

IW 7696177



THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office

October 17, 2018

THIS IS TO CERTIFY THAT ANNEXED IS A TRUE COPY FROM THE
RECORDS OF THIS OFFICE OF THE FILE WRAPPER AND CONTENTS
OF:

APPLICATION NUMBER: *09/608,126*
FILING DATE: *June 30, 2000*
PATENT NUMBER: *6,839,751*
ISSUE DATE: *January 04, 2005*

By Authority of the
Under Secretary of Commerce for Intellectual Property
and Director of the United States Patent and Trademark Office



W. Montgomery
W. MONTGOMERY
Certifying Officer

PART (2) OF (3) PART(S)



US006424624B1

(12) **United States Patent**
Galand et al.

(10) **Patent No.:** US 6,424,624 B1
(45) **Date of Patent:** *Jul. 23, 2002

(54) **METHOD AND SYSTEM FOR IMPLEMENTING CONGESTION DETECTION AND FLOW CONTROL IN HIGH SPEED DIGITAL NETWORK**

(75) **Inventors:** Claude Galand, La Colle sur Loup; Pierre-Andre Foriel, Cagnes sur Mer; Aline Fichou, La Colle sur Loup, all of (FR); Marcus Enger, Hirschhorn (DE)

(73) **Assignee:** Cisco Technology, Inc., San Jose, CA (US)

(*) **Notice:** This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

5,629,927 A	5/1997	Waclawsky et al.
5,787,071 A	7/1998	Basso et al.
5,790,522 A	8/1998	Fichou et al.
5,815,492 A	9/1998	Berthaud et al.
5,898,691 A	4/1999	Liu
5,912,894 A	6/1999	Duault et al.
6,011,776 A	1/2000	Berthaud et al.
6,091,708 A *	7/2000	Matsunuma 370/233
6,108,304 A *	8/2000	Abe et al. 370/232
6,118,791 A	9/2000	Fichou et al.

OTHER PUBLICATIONS

The ATM Forum Technical Committee, Traffic Management Specification Version 4.0, Apr. 1996.

* cited by examiner

Primary Examiner—Wellington Chin
Assistant Examiner—Saba Tsegaye
(74) *Attorney, Agent, or Firm*—Cesari and McKenna, LLP

(21) **Appl. No.:** 09/167,786

(22) **Filed:** Oct. 7, 1998

(30) **Foreign Application Priority Data**

Oct. 16, 1997 (EP) 97480070

(51) **Int. Cl.⁷** G06F 11/10; H04L 1/16

(52) **U.S. Cl.** 370/231; 370/253; 370/400; 709/235

(58) **Field of Search** 370/229-235, 370/400, 419, 420, 463, 252, 253, 522; 709/235, 239

(56) **References Cited**

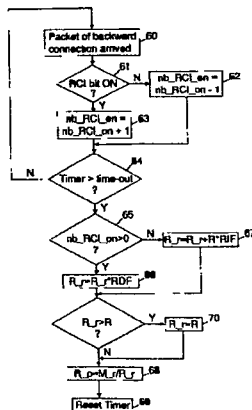
U.S. PATENT DOCUMENTS

5,115,429 A *	5/1992	Hluchyj et al.	370/231
5,313,454 A	5/1994	Bustini et al.	
5,426,640 A *	6/1995	Hluchyj et al.	370/235
5,436,891 A *	7/1995	Grossman et al.	370/231
5,497,375 A *	3/1996	Hluchyj et al.	370/235

(57) **ABSTRACT**

This system is made to perform congestion detection and flow control in high speed digital packet switching network (22) carrying discardable and non-discardable traffic. Forward traffic received at a destination system over a first connection from a source system is monitored. If a congestion-indicating bit is detected in a received packet, a backward congestion indicator is set in packets flowing from the destination system to the source system over a second connection. The source system integrates the number of backward congestion indicators received over successive periods of time using a count-up, count-down counter. Specific congestion control actions are taken at the source system as a function of the counter state at the end of each of the successive periods of time. The congestion control actions may include increasing or decreasing the bandwidth allocated to discardable traffic intended to be delivered over the first connection.

15 Claims, 8 Drawing Sheets



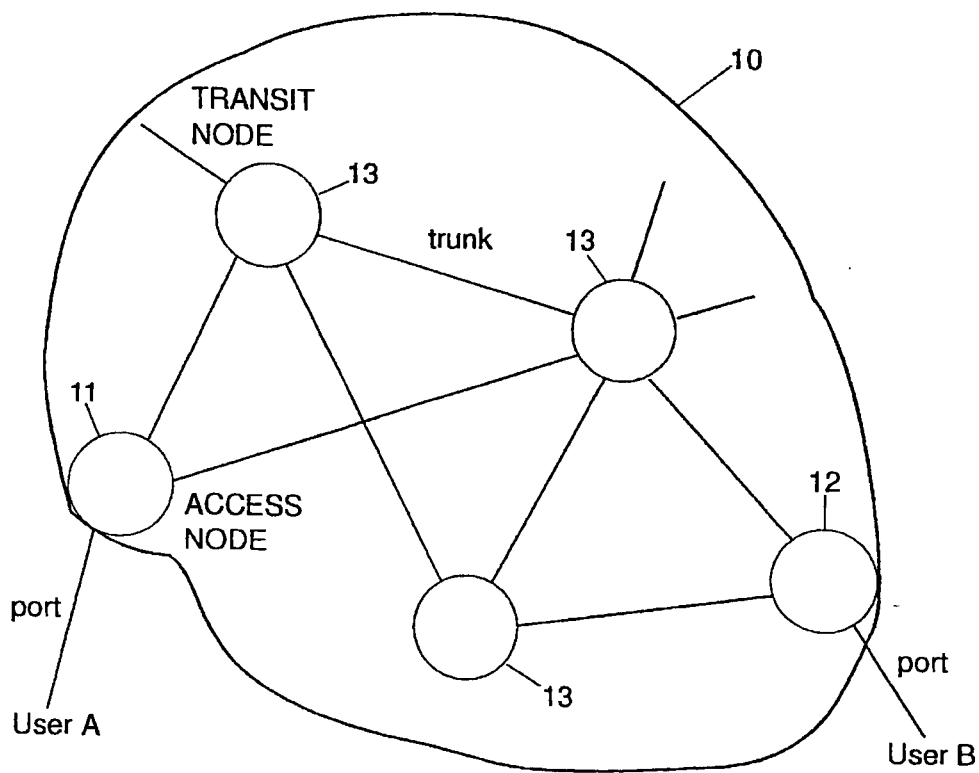


FIG.1

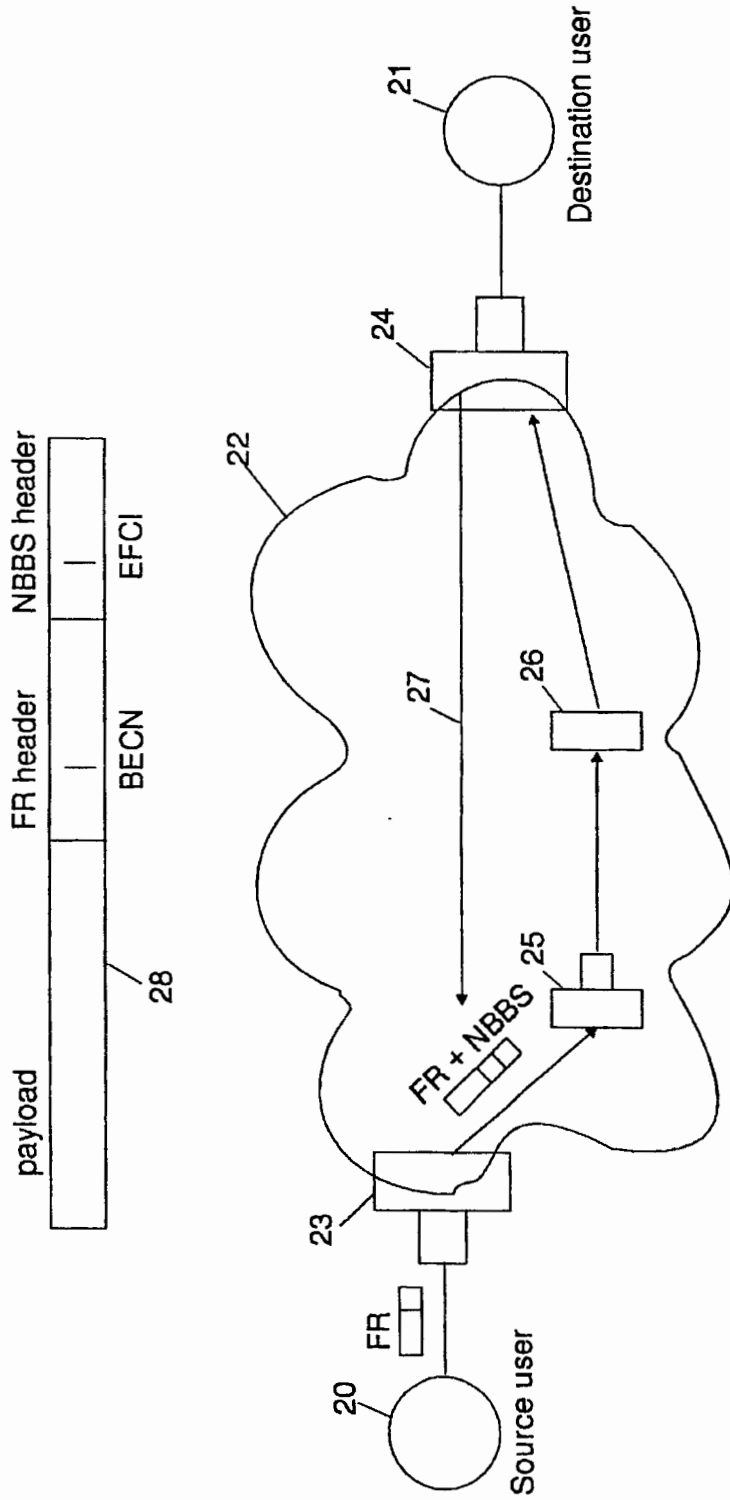


FIG.2

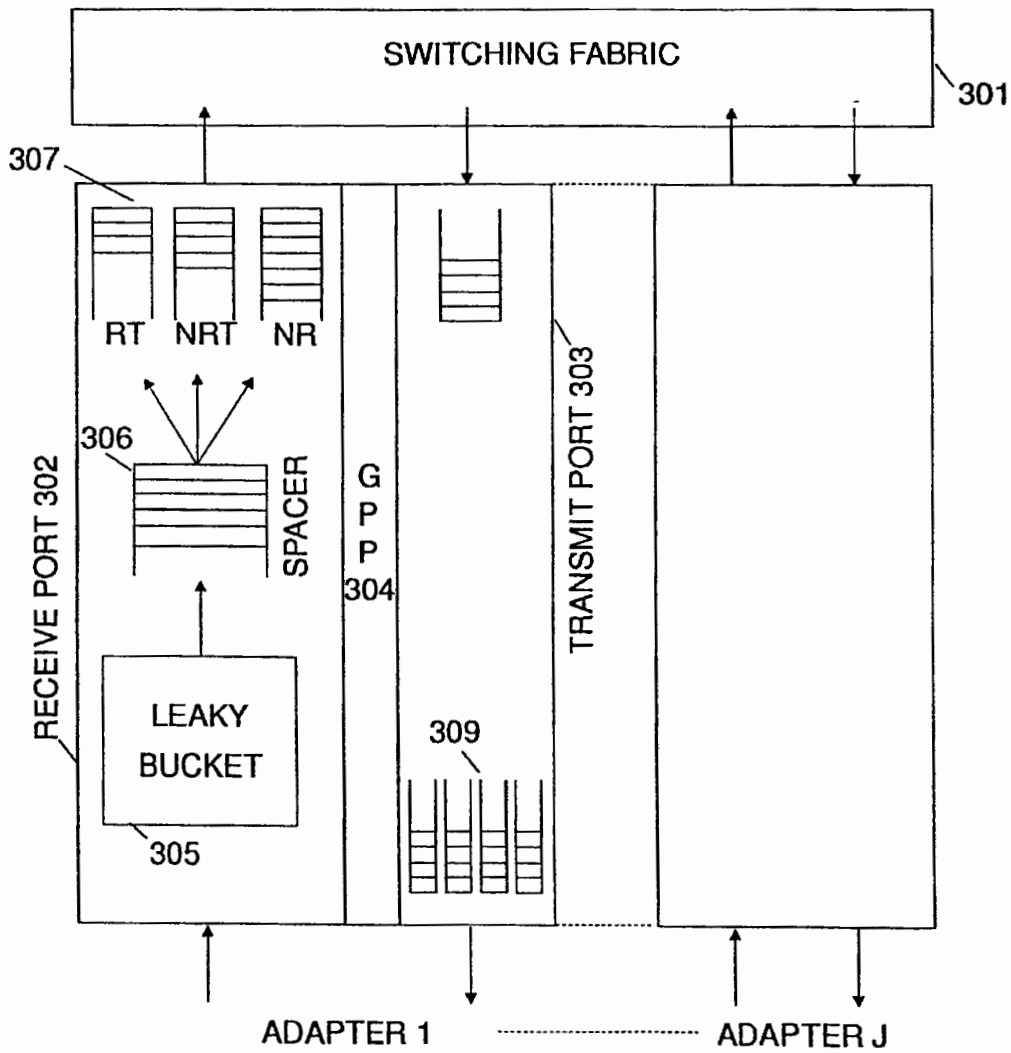


FIG.3

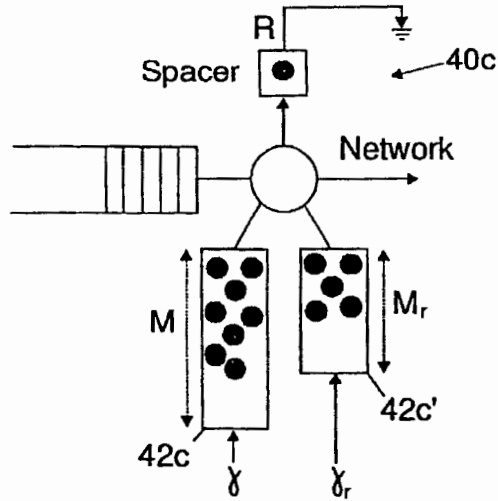


FIG. 4c

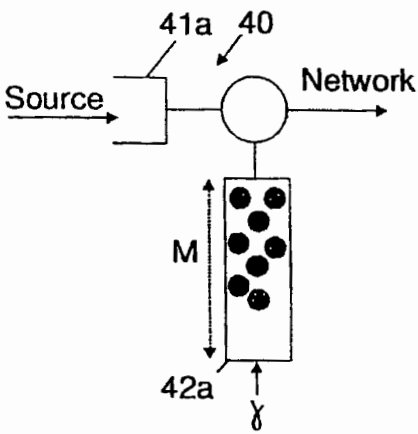


FIG. 4a

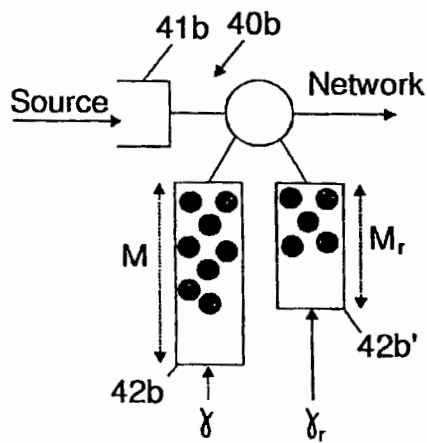


FIG. 4b

FIG. 4

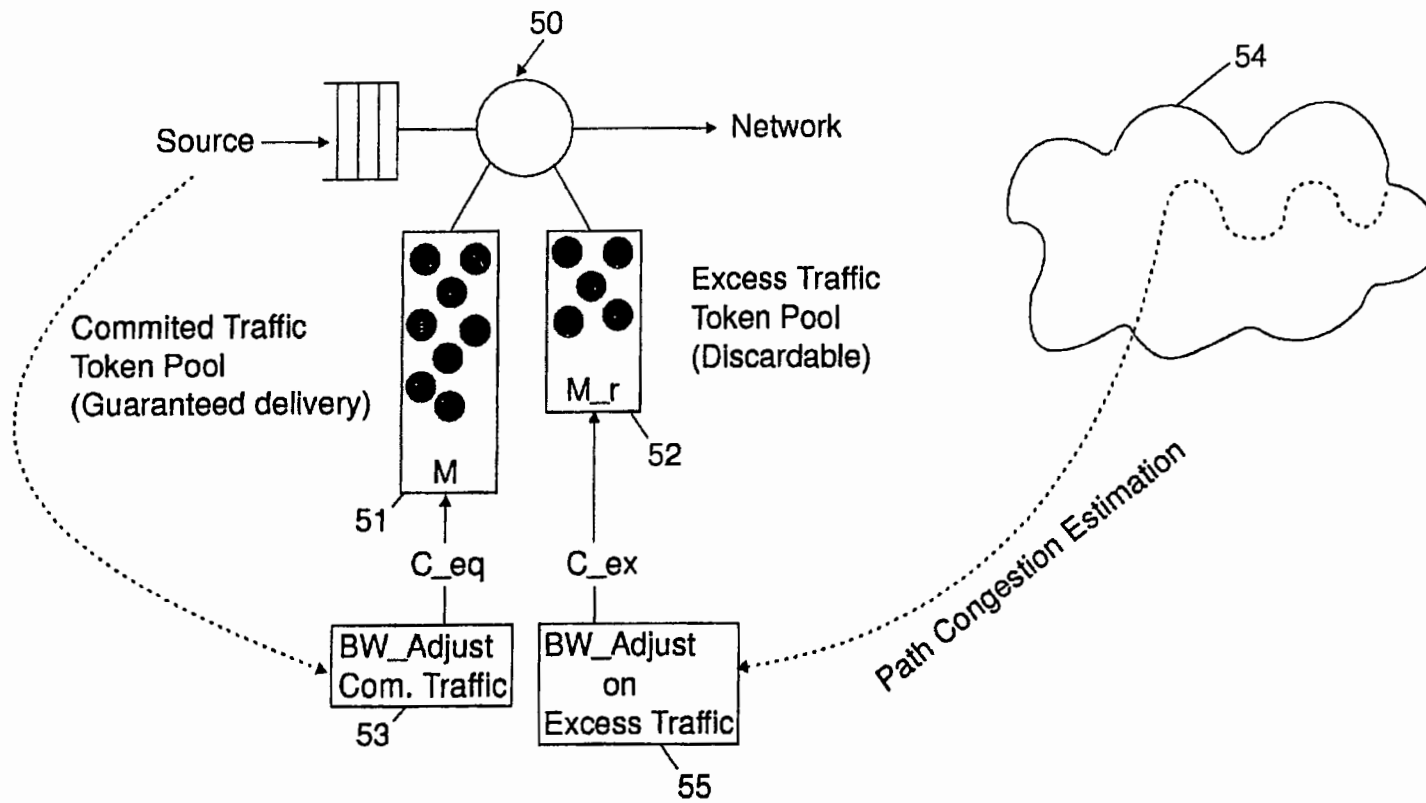


FIG.5

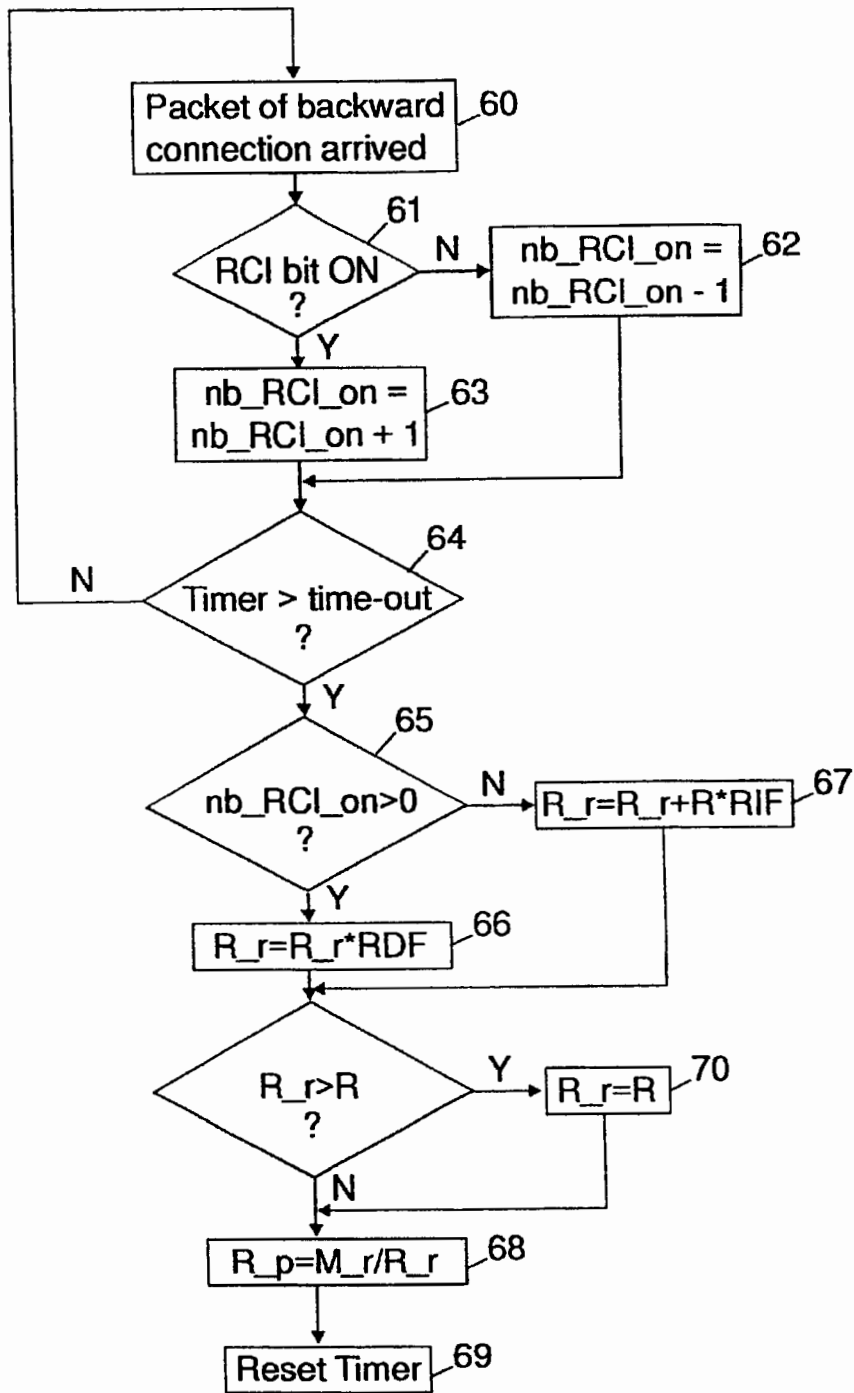


FIG.6

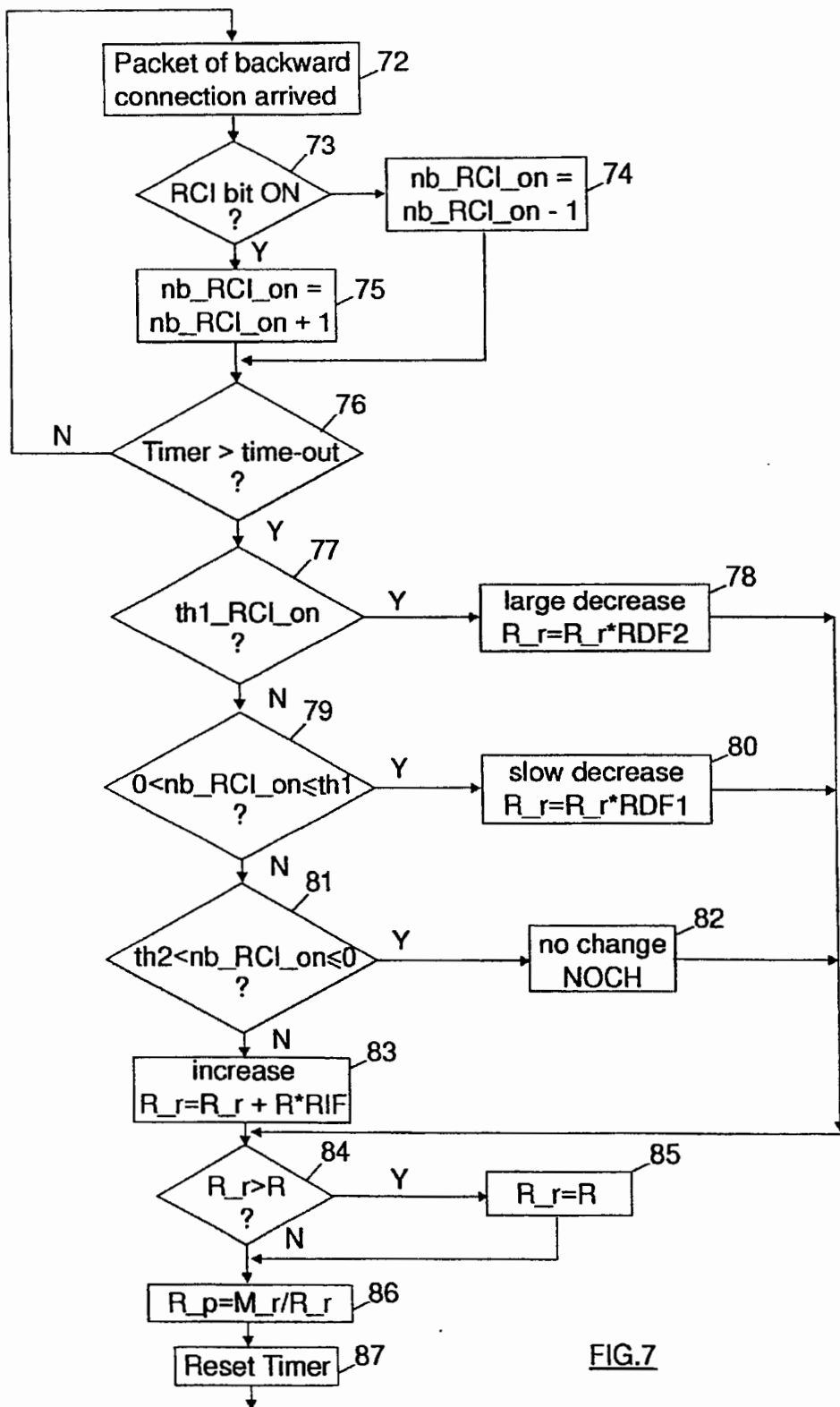


FIG. 7

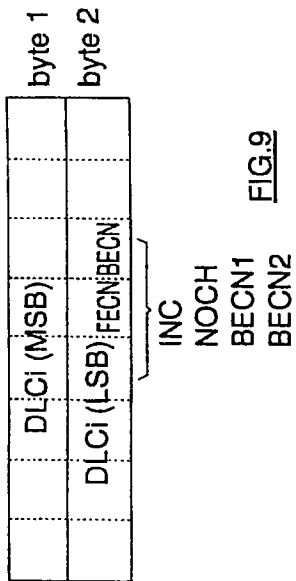


FIG.9

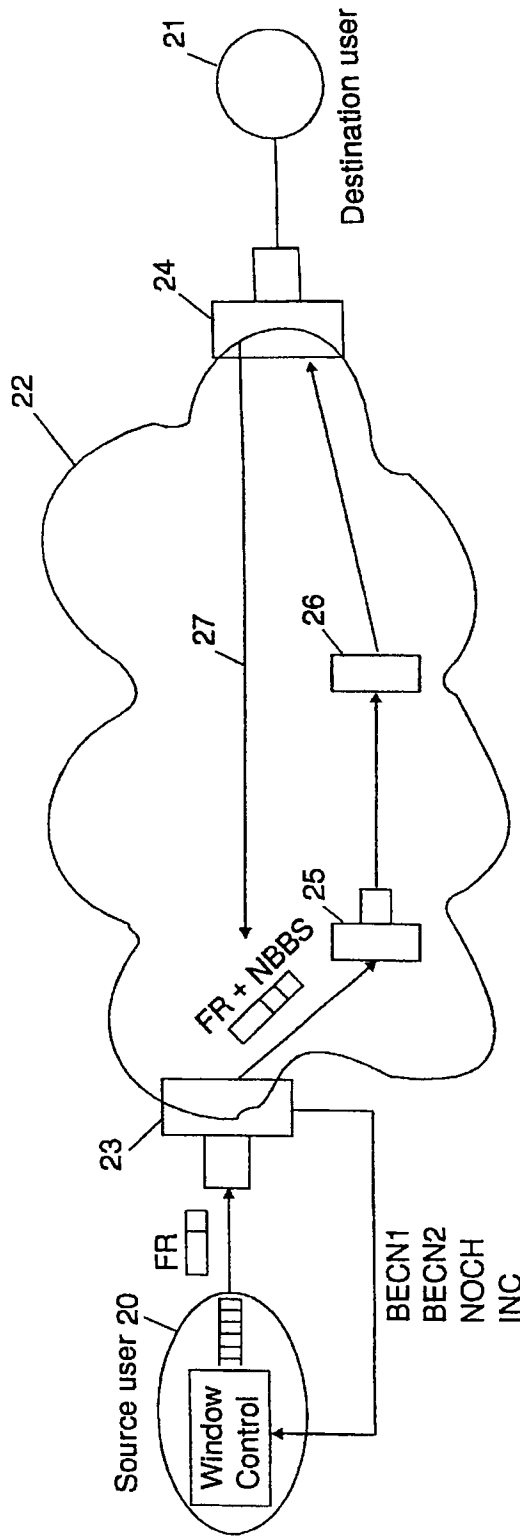


FIG.8

**METHOD AND SYSTEM FOR
IMPLEMENTING CONGESTION
DETECTION AND FLOW CONTROL IN
HIGH SPEED DIGITAL NETWORK**

FIELD OF THE INVENTION

This invention relates to congestion detection and flow control in high speed packet switching networks and more particularly to methods and apparatus for implementing congestion detection and flow control for low priority traffic with optimized cost efficiency.

BACKGROUND ART

Modern digital networks often operate in a multimedia environment and interconnect, upon demand, very large numbers of users and applications through fairly complex digital communication network topologies.

Due to the variety of users' demands and the growth of distributed applications, network traffic is consuming more bandwidth, becoming more non-deterministic and requiring more connectivity. These changes have been the driver for the emergence of fast packet switching network architectures in which data, voice and video information are digitally encoded, chopped into fixed or variable length packets (also named "cells in ATM or Asynchronous Transfer Mode networks) and transmitted through a common set of nodes and links interconnected to constitute the network communication facilities.

The need for efficient transport of mixed traffic streams on very high speed lines (sometimes referred to as links or trunks), means, imposes a set of performance and resource consumption requirements including very high throughput, very short packet processing time, the flexibility to support a wide range of connectivity options and efficient flow and congestion control. Congestion is generally defined as a condition during which network performance is degraded due to saturation of network resources such as communication links, processor cycles, memory buffers, etc.

One of the key requirements for high speed packet switching networks is reduction of end to end delay in order to satisfy real time delivery constraints and to achieve the necessary high nodal throughput for the transport of voice and video. Increases in link speeds have not been matched by proportionate increases in the processing speeds of communication nodes. The fundamental challenge for high speed networks is to minimize the processing time and to take full advantage of the high speed/low error rate technologies. Most of the transport and control functions provided by the new high bandwidth network architectures are performed on an end to end basis.

One basic advantage of packet switching techniques (as opposed to so-called circuit switching techniques) is that it allows statistical multiplexing of the different types of data over a line, which optimizes the utilization of transmission bandwidth. One drawback, however, is that packet switching introduces delays and jitters which might be detrimental for transmission of isochronous data, like video or voice. Methods have been proposed to control the network in such a way that delays and jitters are bounded for every new connection that is set up across the packet switched network.

Such methods are described, for instance, in a published European Application number 0000706297 and include establishing a path through the network high speed lines and nodes, via an entry node port of said network, making optimal use of the available transmission bandwidth of the network along the path to the indicated destination.

Because different type of traffics need to be treated differently to maintain their usefulness at a destination, choices have to be made among the different types by assigning different specific priorities. In other words, when a source terminal requests a connection to a destination terminal via the network (i.e., a call is set-up), a quality of service (QoS) is assigned to the call in terms of maximum permissible delay (T_{13} max) and packet loss probability (P_{loss}).

The QoS and traffic characteristics (e.g., peak data rate, mean data rate and average packet length) are used to compute the amount of bandwidth (i.e. equivalent capacity or C_{eq}) to be reserved on every line on the route or path assigned to the traffic between the source terminal and the destination terminal, in order to guarantee a packet loss probability which is smaller than the loss probability (P_{loss}) that has been specified for the connection. However, in operation, the network traffic must be controlled dynamically which means that some packets may have to be dropped or discarded within the network to avoid traffic congestion.

In practice, it is common to reserve bandwidth for high priority packets (e.g. so-called Real Time or RT traffic) allowing such packets are transmitted in preference to lower priority packets derived from discardable traffic (e.g. Non Real Time or NRT traffic or more particularly Non Reserved or NR traffic). Lower priority packets may be sent at rates greater than their declared rate to dynamically take advantage of any bandwidth remaining after all the higher priority traffic has been served. This remaining bandwidth can vary widely depending on the actual activity of the high priority traffic sources. It is therefore of considerable importance to manage the low priority traffic so as to optimize the use of the widely varying left-over bandwidth in the network while avoiding any congestion which would reduce network throughput. This obviously requires providing the network (and eventually also the sources) with congestion detection and flow control facilities.

Various mechanisms for controlling the flow of NR traffic have been proposed. In particular, an Available Bit Rate (ABR) flow control mechanism has been proposed for Asynchronous Transfer Mode (ATM) networks. ABR flow control is based on use of a particular flow control cell, the so-called Resource Management or RM cell. RM cells are used to collect congestion information from network node switches along connection paths and to send such information back to the traffic sources. While ABR flow control seems to be very efficient, it is complex to implement. End systems must generate RM cells periodically, provide scheduling for RM cells to be sent among data cells, and shape their traffic in response to congestion indications conveyed by received RM cells. Intermediate systems (switches along the paths) must be able to differentiate RM cells from regular data cells, extract RM cells and update these with congestion information. These complexities limit the cost effectiveness of this solution.

Moreover, the above solution requires that all so-called non-reserved (i.e. low priority) sources connected to a network be treated using the ABR mechanism. In fact, if a mix of ABR sources and non-ABR sources are connected to the network, the ABR sources will be disadvantaged as compared to the non-ABR sources which need not be capable of sending RM cells. The customer must therefore update the hardware of all the end systems before using ABR support, which is an additional drawback from an engineering standpoint.

Finally, there is no description, in the ABR standard, of a policing function which could be used to protect the network from misbehaving sources or from non ABR sources.

3

Other mechanisms have been proposed, with flow control which can be used on ATM or PTM (Packet Transfer Mode, including variable length packets) traffic and offer good performance. These flow control mechanisms add complexity to network equipment. Access nodes or ports need to store tables of values and must have the capability of adding a time-stamp to the data packets or to specific control packets. The overhead on the lines is also increased as at least a time stamp must be added to some transmitted packets.

A further improvement was disclosed in U.S. Pat. No. 5,313,454 which made the system transparent to the user (source) by providing an internal congestion avoidance method. To that end, congestion is identified throughout the network and transferred by setting an indicator in the packet header. Then congestion indications are used in the destination node to generate a rate control message which is fed back to the entry node. This prior art still adds overhead to the feedback flow if smooth and flexible congestion control is sought. Otherwise, the flow regulation would be quite rigid and basic.

These functions are generally configured at connection setup and remain static. A more flexible solution is necessary to be able to use the available bandwidth left by the reserved traffic while avoiding high packet loss inside the network.

SUMMARY OF THE INVENTION

The present invention is a method for performing congestion detection and flow control operations for data traffic, including both discardable and non-discardable traffic, in a high speed digital packet switching network including access and transit nodes interconnected by links or trunks. Any source end-user attached to said network via an entry access node can request its traffic to be transported toward a destination end-user also attached to said network via an exit access node. So-called in-going (or forward) and return (or backward) paths are set from the entry node to the exit node and, in the opposite direction, from the exit node to the entry node. The paths might include network transit nodes.

The method includes the step of monitoring the data flow in each transit node in the forward path from the entry node to the exit node for detecting traffic congestion in the transit node. When flow congestion being detected therein, a Congestion Indication (CI) bit is set in a first predefined header field of data packets transported on the forward path to the exit node. Data packets entering the exit node are monitored. Where a set CI bit is detected, a congestion indication is fed back to the entry node by setting a Return Congestion Indication (RCI) bit in a second predefined header field in the data packets of the traffic of the backward path; RCI bits in packets received in the entry node are integrated over a predefined period of time by adding or subtracting one unit depending on the binary value of each received RCI bit. At the end of each of the predefined time periods, the value of the integrated RCI indication is checked. The communication bandwidth assigned to discardable traffic on the forward path, is adjusted as a function of the value of the integrated RCI indications.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a schematic representation of a high speed digital network wherein the invention shall be incorporated.

FIG. 2 is a basic representation of a network implementing the invention.

FIG. 3 is a schematic representation of a network node in which a preferred embodiment of the invention can be implemented.

4

FIG. 4 (consisting of FIGS. 4A, 4B and 4C) shows leaky bucket arrangements to be used in the invention.

FIG. 5 is a schematic representation of the invention as implemented with a leaky bucket.

FIG. 6 is a flow-chart of the algorithm used in the invention.

FIG. 7 is a flow-chart of an improved algorithm for providing enhanced operation of the invention;

FIG. 8 is a schematic representation of a further improvement enabling source cooperation in the flow control mechanism.

FIG. 9 is a detailed representation of the header used in the implementation according to FIG. 8.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT OF THE INVENTION

FIG. 1 is a general block diagram of a packet transmission system 10 showing access nodes/ports 11 and 12 and intermediate or transit nodes (13), the nodes being interconnected by links or trunks. The links or trunks may provide permanent or selectively enabled (dial-up) connections between pairs of nodes. A network transit node may be attached to one or several access (entry or exit) nodes.

Each network node includes input/output adapters (including) buffers interconnected by a switch fabrics, together with data processing facilities providing data communication and network control services in the network node. Data packets (including control data) received on an input adapter may be selectively routed on one or more of the outgoing communication links or trunks through the output adapters. Routing decisions are made in response to information in header sections of the packets. The network node also provides additional services such as calculation of new paths between entry and access nodes, the provision of access control to packets entering the network, and the provision of directory services and topology database maintenance.

A network access node may operate either as a source (entry) of digital data to be transmitted to another end node, or as a data sink (exit node), or both. User A, for instance, acting as a data source utilizes an access node 11 to access the packet switching network 10. The access node translates the user's data into packets formatted for transmission on the network and also generates headers used to route said packets through the network. At call set-up, the entry node processing facilities calculates a path or route through the network from the source node (entry node) to the destination node (exit node). To avoid overload on any of the links on the path, the path is selected using an algorithm that ensures that adequate bandwidth is available for the new connection, while optimizing the overall throughput within the network.

The act of selecting a path for a particular connection implies the allocation of network resources to users in order to guarantee their Quality of Service (QoS) requirements. Various quality levels of service may be specified, some of them in order to satisfy real-time delivery constraints, others related to non real time data traffic transfer. To that end, the origin node computes a path to the destination node that is capable of carrying the new connection and providing the level of service required by the new connection. The Path Selection process uses data describing the current traffic load in the entire network (nodes and links). Such data are stored in a topology database located in each node of the network. If no suitable path can be found to meet all requirements, the connection is rejected. Once, the origin node has found a

5

suitable path, a set-up message is generated which traverses the selected route, updating the resource allocations (including bandwidth occupancy) for each link visited by the set-up message.

To maintain high network throughput, a path is selected and resources are reserved only at the time of the connection establishment. The Path Selection process takes into account various constraints which come both from the user (quality of service requirements, user's traffic characteristics) and from the current network topology and bandwidth allocation. In addition, the algorithm maximizes the network throughput by choosing a path with the least number of hops and which tends to achieve an even distribution of the traffic among the links. Once an appropriate path has been selected, the network connection establishment process takes place, and only then are the resources along the path reserved.

Accordingly, the connections established for different sources may require different levels of bandwidth, different delay guarantees, and/or different loss probabilities. Real time signals such as voice or video, for example, require being assigned higher priority levels than non-real time signals.

As already mentioned, such connection characteristics are negotiated along with the bandwidth requirements and enforced by assigning higher priorities to the transmission of real time signals and, in case of congestion occurring, discarding non-reserved packets before discarding reserved ones. In other words network management must distinguish between guaranteed delivery traffic and so-called discardable traffic. Nevertheless, optimizing the transmission of lower priority traffic is important.

Therefore and due to the bursty nature of data traffic, traffic should be continuously monitored and a mechanism provided for adjusting low priority traffic or any excess traffic in excess, and controlling the assigned bandwidth dynamically. For instance 85% of the total link bandwidth may be reserved for committed traffic, which leaves 15% of said bandwidth for dynamic assignment.

The present invention is a method and apparatus for optimizing the transmission of lower priority traffic while avoiding congestion in the network.

The design of adequate congestion schemes is extremely difficult for the following reasons:

Overhead: A congestion scheme should not increase the traffic too much, in particular when the network is congested. Even proposals of positive feedback schemes, i.e. schemes that give feedback when the network is congested, do not resolve the problem totally as they also consume resources.

Fairness: When the demands of all users cannot be satisfied, the share of satisfied demands should be fairly distributed. But neither the definition of "fairly" is trivial, nor the meaning of what is considered to be fair is invariable.

Responsiveness: The quality of the available resource is often changing dynamically. A congestion scheme has to react quickly to changes by asking users to increase or to decrease their sending rates. On the other hand, in order to maintain good throughput, only persistent congestion should be taken into account. Short term loading of the queue due to traffic bursts should be differentiated from a persistent state of congestion.

Bad environments: Under congestion, packets can be dropped, changed or delivered out of order. Despite such problems, the scheme has to continue to work.

6

Consideration of the totality: The final aim of a congestion scheme is to optimize the overall network performance. Schemes that optimize the throughput of one user or one trunk, do not always maximize the overall performance.

Complexity: In high speed environments there are often only a few clock cycles to process a packet in a node. Therefore, a flow control scheme has to be simple. This criteria can also be called Implementability.

Misbehaving sources: Some congestion control schemes suppose that all users are able and willing to cooperate. These schemes do often not work with misbehaving users. Greedy users can degrade the QoS for other users or can even drive the network into long term congestion.

The aim of the system of this invention is to check the traffic by detecting congestion in any node of the network and then monitor the traffic flow accordingly on a port-to-port basis, while optimizing the network by complying with the requirements to avoid the drawbacks as indicated in the prior listed criteria, e.g. by minimizing traffic overhead, and yet enabling a smooth and flexible congestion control; being insensitive to misbehaving source users, and yet enabling a control of behaving sources at almost no additional cost; being perfectly implementable and rather not complex.

FIG. 2 is a schematic representation of a network implementing the invention. It shows a connection set between a source end-user 20 and a destination end-user 21 through a digital communication network 22. The network includes an entry access node/port 23 through which the source user 20 is attached to the network 22, while the destination user 21 is attached to the network 22 via exit access node/port 24. Assume that, at call set-up, the source end-user's request for a connection had been executed by setting a forward path via transit nodes 25 and 26. Since the traffic works both ways, Assume that the return path between user 21 and user 20 was established as indicated by arrow 27 also through a series of transit nodes (not shown in the figure). Consider first traffic originating at user 20. As already mentioned, in a packet switching system, source traffic is split into packets including a data payload and a header containing control information. Packets may either be of fixed length as in Asynchronous Transfer Mode (ATM) of operation or be of variable length as in Frame Relay (FR) mode of operation. According to this invention, the packets flows are monitored throughout the forward path, in each intermediate transfer node on the path. As soon as traffic congestion is detected in one of the nodes, a predefined header field is marked or set to indicate congestion has been detected. When a congestion indication is detected in exit node 24, each packet in the return path 27 is marked further congestion.

Again, we must say that this invention applies whatever be the type of traffic, be it organized in Asynchronous Transfer Mode (ATM) or in Frame Relay (FR) mode. But the mode of operation selected herein to describe the invention is the FR type (see FIG. 2). Accordingly, each packet of data provided by the source (20) traffic includes a payload section and a Frame Relay header section (FR header)(see packet 28). Now, for the purpose of this invention, each packet entering access node 23 is provided therein with a header. The fields selected for both marking congestion indication (CI), through a so-called Explicit Forward Congestion Indication (EFCI) in any network node along the forward path going through nodes 25, 26, . . . , and for Returning said Congestion Information (RCI), have herein been selected to be in the header and in said Frame Relay (FR) header, respectively. Also, and for optimizing traffic efficiency, these

fields have been limited to be one-bit long each in the best mode of implementation of this invention. Naturally, those field locations and lengths selection shall by no means be considered as limiting the scope of this invention. A person skilled in the art understands that longer fields shall enable carrying more detailed flow congestion information while increasing network complexity and increasing operating cost, as shall be explained further in this description.

As already known in the art of digital communication, and disclosed in several European Applications (e.g. Publication Number 0000719065 and Application Number 95480182.5) each network node basically includes input and output adapters interconnected via a so-called node switch. Each adapter includes a series of buffers or shift registers where the node transiting packets are stored. Traffic monitoring is generally operated via preassigned buffer threshold(s) helping monitoring shift register queues, as shall be described with reference to FIG. 3.

Represented in FIG. 3 is a simplified block diagram showing a network node with the corresponding facilities used to implement the invention. Basically, it includes a series of adapters labeled adapter i with $i=1, 2, \dots, j$ (j being the number of adapters), and a high speed packet switching fabric (301). Each adapter is connected to a network link/trunk. Each adapter includes a receive port section (302), a transmit port section (303) and processing facilities herein designated as General Purpose Processor (GPP) (304). The packets arriving on the node via the receive port are oriented toward the appropriate node adapter (transmit section) to finally reach the destination user through the preassigned network connection path. Switching between an adapter receive port and an adapter transmit port is performed via the switching facilities (301). To that end, the adapters processing means determine the routing operations to be performed, by using indications provided within the data packet headers. As represented in FIG. 3, the adapters include queuing facilities for queuing the packets prior to or subsequent to their launch on the switch (301).

As mentioned, at connection set-up time, two paths, one for each direction are computed between the original access node and the destination access node attached to the calling source user and the destination user, respectively. The connection set-up and bandwidth reservation process operate to adjust, if the call has been accepted, an access control device (e.g. leaky bucket 305) according to the network connection characteristics. The leaky bucket may be provided with traffic shaping capabilities; e.g., spacer 306. A more detailed description of both possible implementations and operations of the leaky bucket means shall be provided later with reference to FIG. 4.

Once exiting the leaky bucket/spacer means, the data packets are oriented toward different queuing elements (307) based on the assigned priorities. In the preferred embodiment of this invention, three different priorities have been assigned, i.e. in decreasing priority order: one for so called Real Time (RT) traffic, one for Non Real Time (NRT) traffic, and the third one for Non Reserved (NR) traffic. Traffic congestion is declared when predefined threshold(s) are reached in the transmit adapter port (303) queues (309).

Now returning to FIG. 2, let's for the moment assume one single threshold level for denoting congestion in a network node has been assigned to the system. As soon as this threshold is reached in anyone of the network nodes of the in-going (forward) path between source (20) and destination 21 (e.g. in node 25), a congestion indication EFCI bit is set to "1" in the header of the packet actually issuing the queue, down to exit node 24. In the node 24, and prior to removing

the header from the received packet, the congestion indication EFCI bit is monitored. When an EFCI bit at binary level "1" is detected, the destination port (24) marks each packet in the backward flow, i.e. from port 24 acting now as a source for user 21 back to original source user's port 23, with the Returned Congestion Information (RCI), by setting to "1" the BECN bit in the FR packet header (see 28). When the source port receives a packet with the BECN bit at binary level "1", a predefined flow control action is taken. For ATM/ABR traffic the RCI bit may be selected in the RM cell.

Several flow control operating algorithms might be defined but efficient algorithms should preferably provide a mechanism for dropping the transmission rate on the congested path in a single large step and then enable slowly going back to full rate step by step.

Thus, in the preferred mode of implementation of this invention, the action to be taken upon reception of a packet indicating congestion, i.e. with the RCI (BECN) bit set (i.e. at binary level "1"), is based on an additive increase and a multiplicative decrease of the considered sending rate, and this control action is performed in the entry node 23 directly under network control rather than source control, therefore neutralizing any burden due to a misbehaving source user. The parameter for the increase is called Rate Increase Factor (RIF) and is expressed as a quantile of predefined peak rate. The factor for the decrease is called Rate Decrease Factor (RDF) and is a quantile of the actual rate.

Now as far as implementation is concerned various flow regulating devices are already known in the art. They are based on so-called usage parameter control (UPC) made to prevent connections from sending with different characteristics than those negotiated in the traffic contract, whether this is done on purpose or not.

One of these flow regulating systems utilizes a so-called leaky bucket (see FIG. 4 representing three different leaky bucket implementations (4a; 4b and 4c)) including an admission shift register (e.g. 41 a) wherein the data packets are buffered and a so-called leaky bucket mechanism to control their passing further to the network.

The Leaky Bucket label describes a family of algorithms with the same basic principle, based on the consumption of credits designated as tokens (see FIG. 4). The tokens are generated at a rate γ and can be accumulated in a token pool or bucket (42a) with the size M . When the token pool is full, no more tokens are generated.

Each packet generated at a source has to pass the Leaky Bucket (see 40a) before entering the network and needs a certain number of tokens to pass. The number of tokens can correspond to the packet size in bytes or for ATM, one token can correspond to one cell. When no more tokens are available, the packet is dropped rather than being passed.

A full token pool means that a source may send a burst of size M at peak rate without losing a packet in the Leaky Bucket. On the other hand, an empty token pool means that a source may still send at rate γ and all packets arriving at higher rate are dropped.

The Leaky Bucket is determined by the parameters (γ, M). An appropriate setting of these parameters can be used to enforce the usage parameters. For example, a low burst tolerance will be translated by a small M and the peak rate will be an upper bound for γ .

Various extensions to the basic Leaky Bucket represented in FIG. 4a, exist. One of the most used is the extension to allow violation tagging. As shown in FIG. 4b, two token pools (42b, 42b') are used in that case. The second pool (42b') is used to allow excess packets to enter the network, but

marked with lower priority. The parameters γ_r and M_r are used for the second pool, called the red token pool. Accordingly marked packets are called red packets and the tokens are called red tokens. The traffic in the first pool is denoted green and is controlled by green tokens (see pool 42b).

For each packet, the green token pool is first checked to see if enough green tokens are available. If there are not enough green tokens, the red token pool is considered. If there are enough red tokens, the packet is assigned a lower priority before being sent into the network, otherwise the packet is dropped.

A third version of the leaky bucket may be implemented, ensuring Traffic Control, Priority Control and Traffic Shaping. It is a Leaky Bucket linked with an entry buffer and a third pool: the spacer (see FIG. 4c). Conceptually, the spacer receives the tokens from the leaving packet and is emptied at peak rate R . To send a packet immediately, two conditions must simultaneously be true. The spacer must be empty and there must be enough tokens in the green or the red pool (42c42c'). In other words, a packet may not leave the system until the spacer is empty, even when there are enough red or green tokens. In this way, the transmission rate can be controlled and can never be higher than R .

The implementation often differs from the concept, as the spacer is implemented as a scheduler. The scheduler calculates the next possible departure time after the last packet with the peak rate R and attributes this time to the current packet.

This kind of mechanism is only applied to delay-insensitive traffic classes.

The general question for Leaky Bucket algorithms concerns the rate γ . Which rate should be used for the reconstitution of the tokens and which bandwidth should be reserved for the connection? This domain is called Resource Management and is widely documented in the literature. The theory allows calculation, starting from the available buffer space and the traffic descriptor, of all the parameters of the Leaky Bucket. The calculated parameters guarantee a very low loss probability with a certain interval of confidence for the data sent into the network.

With these kinds of arrangements the network may be designed and managed to assign, say 85% of a link nominal capacity to reserved traffic and then dynamically monitor and control the remaining bandwidth and assign it to so-called excess traffic. The invention enables control of excess traffic at the network entry node to avoid network congestion without having to rely on traffic sources behavior.

This principle is schematically represented in FIG. 5. A double token pool (51,52) is again used for controlling the traffic entering the network through leaky bucket mechanism 50 (possibly including a spacer not shown). The token pool 51 is used to manage committed traffic whose delivery has been guaranteed at rate C_{eq} equal to the equivalent capacity that has been reserved for the considered connection in the network, or with a token rate which fits the minimum guaranteed bandwidth. Any bandwidth (BW) adjustment through leaky bucket control (53) for committed traffic is then left to the source. But for the excess traffic which by nature is discardable, it shall be handled under the path congestion mechanism of this invention using the EFCI/BECN indications provided by the communication network (54) to the network access node involved, to adjust the Excess Traffic (ET) token rate (55) and therefore adjust the excess capacity (C_{ex}) at network entry without involving the data source, by adjusting the token pool refill rate to R_r as required, to dynamically optimize bandwidth utilization while avoiding network congestion.

In operation, for said excess/discardable traffic, the port algorithm may be designed so that the parameters will be adjusted according to the information of experienced congestion (RCI). The above mentioned adjustments might be performed periodically, or integrated.

The algorithm as implemented in the source and destination ports of the preferred embodiment of this invention is summarized below. It should be remembered that the action to be taken for packets with the RCI (e.g. BECN) bit set to one, is based on additive increase or multiplicative decrease of the sending rate. The parameter for the increase (i.e. the Rate Increase Factor (RIF)) is expressed as a quantile of the peak rate, while the Rate Decrease Factor (RDF) is a quantile of the actual rate. A new variable, i.e. the red token Refill rate (R_r) is introduced. This rate is only used to calculate the increase and decrease and is translated into the Refresh period (R_p) at the end of the action per packet. The rate R_r is not a real send rate. It is an upper bound for a possible mean send rate, due to its integration into the leaky bucket behavior.

The source port algorithmic steps for the adjustment of the leaky bucket parameters are as follows:

1. When congestion is detected after integration of the RCI indications, the red token pool size (M_r) is normalized to the maximum packet size (e.g. 2 KB). The refresh period is adjusted by conforming to the new M_r value (which is initially set to the time to send M_r with peak rate). The red token refill rate (R_r) is derived from the red token pool size and the refresh period by the calculation $R_r = M_r / R_p$.
2. For each congestion detected after integration of the congestion indications in packets, the rate is decreased by the factor RDF (0.85 for example) until the lower bound (5 kbps for example) is reached:
If $(R_r * RDF) > 0.005$ Then R_r is set to $R_r * RDF$;
(the symbol * indicating a multiplication)
3. For status of no congestion, after integration of the congestion indication, the rate is increased by the addition of the RIF multiplied by the peak rate until the peak rate is reached:
If $R_r + (R * RIF) < R$ Then R_r is set to $R_r + (R * RIF)$;
where R is the actual access rate,
4. The refresh period is derived from the red token refill rate: $R_p = M_r / R_r$;

In other words, the system first starts with a check whether a congestion control action is needed. In case of a positive answer, the red rate in the leaky bucket (i.e. R_r) of the source entry node (23) is decreased by the rate decrease factor (RDF) and the refresh period in the leaky bucket (R_p) is set to (M_r / R_r) wherein M_r designates the size of the red token pool in the leaky bucket. Otherwise, the red rate in the leaky bucket is increased by $R * RIF$, with R being the actual access rate and RIF the rate increase factor and then the R_r period in the leaky bucket is refreshed. In either case, R_r is limited to R .

Where the destination port (e.g. see 24 in FIG. 2) detects an EFCI bit set at binary value "1", it has to notify the transmit side of same network exit node adapter of the experienced congestion with the noted frequency. The adapter transmit side will then mark all the packets of the connection travelling in the opposite direction (see 27 in FIG. 2) with the congestion indicator, i.e., will set the RCI bit in all the packets sent on the backward connection. In this case, the destination port (24) does not smooth the information. In fact, the value of the EFCI bit of the last packet received in the destination port (24) before the information is sent to the transmit side is decisive for the value of the RCI bit for the connection.

Thus, in the above described implementation no smoothing algorithm was applied on the receive side to convey the information back. This can be explained by the aims of the algorithm: protect the network, i.e. avoid packet loss and use available bandwidth, even when the bandwidth is available for a short time period.

When congestion is experienced, the source port adapter has to be notified immediately to slow down the red token rate in order to avoid excessive packet loss. That means, on the receive side, that when a port adapter receives only the last packet with the EFCI bit set and all the others did not have the EFCI bit set, it has nevertheless to convey the information about the detected congestion back to the source port. On the send side two options can be used. The port adapter may change the parameters of the leaky bucket to limit bandwidth use regardless of the number of RCI bits received at binary level "one". As an alternative, the port adapter may integrate the RCIs to smooth the leaky bucket changes. This smoothing phenomenon is particularly interesting as it enables substantial improvement in the congestion control with a minimal overhead.

The algorithm for implementing the inventive process is represented in FIG. 6. At connection set-up, two parameters of the connection (in each direction), one for the number of set RCIs bits (nb_RCI_on), and the second used as a timer, are initialized. The timer is set to zero and started, while the nb_RCI_on counter is set to zero.

Each time a packet is received in the backward direction (60), the RCI bit is extracted and tested (61). If this RCI bit is OFF, the nb_RCI_on counter is decremented by one unit (62), else, it is incremented (63). Then the timer indication is checked versus a predefined value considered as time-out indication (64). As long as the time-out is not reached, integration over the RCI bit goes on, with the process looping back to step (60). When the timer indicates a time-out, the integrated RCI bit indication is considered. In a preferred embodiment, the congestion status is evaluated as follows: if nb_RCI_on is higher than zero as indicated by test (65), then congestion is declared and the actual red rate (R_r) is decreased as already indicated, by the predefined rate decrease factor (RDF) (see 66). Otherwise no congestion is declared and the red rate in the leaky bucket is incremented by $R \cdot RIF$ (67), with R being the actual access rate and RIF the rate increase factor and then the refresh period in the leaky bucket is amended to $R_p = M_r / R_r$ (68). In both cases the timer is reset (69) and R_r is limited to R (70).

With the use of the timer, the flow control proposed integrates more or less depending on the data flow: if the backward data flow is important there will be a lot of "up-to-date" RCI information to process. To avoid multiple changes in the leaky bucket parameters and useless processing, integration is performed. If the backward data flow is low, each RCI indication is important and the integration is low.

The solution described with reference to FIG. 6 provides a valuable improvement over prior art while minimizing overhead traffic. It can still be improved by providing more precise leaky bucket regulation, assuming several different levels of congestion and different RDF/RIF functions of these levels.

One implementation of such an improved algorithm is represented in FIG. 7. The first series of operations (i.e. 72, 73, 74, 75 and 76) are identical to those described with reference to FIG. 6 (i.e. 62, 63, 64, 65 and 66), but once time-out is declared then the count of nb_RCI_on based on packets received over the feedback path is compared to

achieve $R_r = R_r \cdot RDF2$ with $RDF2$ being the largest predefined rate decrease factor. Otherwise, if the test (77) is negative, a second test is performed (79) to check whether nb_RCI_on is between zero and th1. If this is the case, then a decrease at slower rate ($RDF1$) is implemented (80) and R_r is made equal to $RDF1 \cdot R_r$; otherwise a third test is performed (81) versus a negative threshold th2. If the nb_RCI_on is between zero and th2, then the rate is kept unchanged (82), otherwise it is increased (83) to $R_r + RIF \cdot R$, with a limitation to the peak rate level (84, 85) and the token pool refresh period is refreshed to M_r / R_r ; then the timer is reset (87).

Such an improved method and system enables a smoother rate regulation from the network entry access node and further enables driving the source user assigned window adaptation if required. To that end, the rate variations defined here above: i.e. Increase (INC) (83), no change (NOCH) (84), slow decrease (BECN1) (80) and large decrease (BECN2) (78) shall be coded with two bits in the input access node and fed back to the source, as represented in FIG. 8. These two bits may be selected in the two byte long Frame Relay header as defined in the ANSI Standards (see FIG. 9). In a preferred mode of implementation of this invention, these two bits have been selected to be the third and fifth bits of the Least Significant byte (byte 2). INC, NOCH, BECN1 and BECN2 are therefore coded accordingly. With these two bits, the process is thus improved to control the NCP window feeding the source queue as illustrated in FIG. 8.

The four rate variations defined in FIG. 7, steps 78, 80, 82 and 83, while being used in the flow control operated in the network access node 23 in response to the network 22 congestion integration over the considered connection, are also used for source control. This additional control is performed over the NCP window of source user 20. Several window controls may be defined, but in the preferred mode of implementation of this invention, the window adjustments have been made as follows:

- if BECN1 (slow decrease), then the window is adjusted to:
window=window-n (with $n=1$).
- if BECN2 (large decrease), then window=window/p (with $p=2$).
- if NOCH (no change), then window=window.
- if INC (large increase), then window=window+q (with $q \geq 1$).

Consequently, the single bit added to the overhead in the backward path within the network 22 enables smooth flow control directly from the considered network entry access node thanks to judicious integration of congestion indicators in said access node. Further smoothing for congestion control is possible where combined with multiple thresholding. In addition multiple thresholding operation may be coded into the entry access node and fed back to the source user system to enable further regulating the source generated data flow if required without being bound to non-behaving sources. The method is therefore quite convenient in that it enables a smooth and flexible congestion control releasing the network from any unnecessary additional overhead.

The preferred embodiment of the invention as described herein was limited to two bits for the overall forward and backward congestion indications to limit the overhead on the links. But some improvements to the invention can still be provided by using these bits in a slightly different way.

In effect, when using one threshold only to detect congestion and set and reset the EFCI bit in the packet header, a typical threshold phenomenon can occur with oscillations around this threshold.

These oscillations could be avoided by a second "unset-threshold". When a packet arrives and the queue size is greater than the set-threshold, the EFCI bit is set in all the packets. Only when the queue size gets below the unset-threshold, the EFCI bit won't be set anymore. This "hysteresis" avoids oscillations around one congestion indication threshold.

However, the oscillations might in some cases be ignored if the packets are not discarded at this threshold, they are only equipped with an additional information. Knowing that the information is not translated immediately into the RCI field, but with a certain frequency, it does not matter a lot if there is a mixed sequence of EFCI and non-EFCI packets. In the worst case the source would receive one wrong 'no congestion' information. The buffer in the trunks is supposed to support these short time periods of wrong information.

Once more the delay criterion has to be considered talking about two different thresholds. When the queue size becomes smaller this means that the input rate is smaller than the output rate. When the whole queue is empty, the link would be under-utilized. The unset threshold should therefore be reasonably high, to inform the sources as soon as possible that the congestion situation does not exist anymore. Knowing the already added delay we would tend to fix the unset-threshold higher than the set-threshold, which implies a certain complexity in implementation (the trunk has to keep track of the increase or the decrease of the queue size).

For implementation reasons an unset-threshold would add supplementary delay because it would be lower than the set-threshold, i.e. it would need more time to decrease the queue size to reach the unset-threshold. This added delay would decrease bandwidth utilization as the sources would start later to recover from the experienced congestion.

A simulation performed at T1 link speed to evaluate the impact of two thresholds on the system as compared with one threshold provided the following results:

Criterion	Two thresholds	one threshold	difference
LINK UTILIZATION	82.53%	82.04%	0.49%
PACKETS LOSS AT TRUNK	0.0165%	0.0206%	0.0041%

In conclusion, a second threshold in the forward direction could avoid periods where some packets are marked and others are not. This could theoretically increase fairness and reactivity of the sources. Simulations showed, that there is only a minor difference between the use of one or two thresholds. The oscillation periods are very short as the queue growth has always a clear direction upward or downward.

Some improvements to the invention efficiency might be obtained by focusing on a number of parameters in the implementation.

For instance, a modification in the way the packets are EFCI marked would lead to higher bandwidth utilization. If, instead of being marked in the receive port, when they arrive on the adapter the packets are marked just before they are sent away (i.e. after buffering) the responsiveness of the system would be improved.

Responsiveness of the system might also be optimized by a careful selection of the Rate Decrease Factor (RDF) and Rate Increase Factor (RIF) of the algorithm. For instance, the RDF and RIF values were evaluated for a range of trunk speeds from 128 Kbps to 44.74 Mps. The values optimized

for a T1 trunk gave reasonable throughput for all tested speeds with RDF=0.85 and RIF=0.002. In other words, on each intervention the rate shall be decreased to 85% of its old value, or increased by a constant of 0.002*R. The decreases are quasi exponential and the increase almost linear. These values are key points in terms of responsiveness of the system. The aim of the so called excess traffic controlled by the invention is to use even shortly available bandwidth. Therefore the RIF should be preferably selected high enough to enable the system to adapt in few steps to a higher rate. On the other hand, a too high value would increase the rate too fast, allowing the sources to send too much packets when congestion occurs before its notification reaches the source, leading finally to packet loss at the congested trunk.

On the other hand, a too small value would be inappropriate if the number of sources is low.

Therefore a compromise value should be selected carefully, based on the possible number of connections involved.

The RDF value should also be selected carefully to enable optimized adaptation to less available bandwidth.

The decrease of the rate must be done as fast as possible, because the congestion is always detected too late, due to the various delays the propagation of the information suffered from. The exponential character of the decrease provides this fast response, in 1 sec, i.e. after 10 adjustments, the rate is at 20% of its original value.

A local optimization for each trunk speed makes not always sense as computer networks often integrate lines with different transmission speeds.

The red packet discard threshold has a direct impact on the ability to accept bursts and on the queue size, which is translated into added delay for the packets. There are four main ideas behind the value of the red packet discard threshold:

1. Use the available buffer to accept bursts Each trunk is equipped with 256 KByte buffer space for the NRT and NR traffic classes. This buffer can provide a great flexibility for the adapter to accept data bursts.
2. Protect the green packets by leaving them enough space above the red packet discard threshold. Even by discarding red packets, there may be the situation where there are bursts of green packets, leading to queue size values over the red packet discard threshold. It has to be assured, that the queue size never exceeds the available buffer size, to avoid the discarding of green packets. Therefore the red packet discard threshold has to be inferior to the available buffer space.
3. Assure maximum delays for low speed trunks. A queue size of 200 KB implies a sending a delay of 12.5 s at 128 Kbps. This can lead to time-out events for NBBS messages and higher level protocols. The queue size has therefore to be reasonably small to assure maximum sending delays of less than 1 sec per congested trunk at low speed.

With the above solution provided by this invention for monitoring and controlling congestion, a fairly stable mechanism is made possible. It is nearly insensitive against packet loss and reacts to congestion before packets are lost, and, most of all, it is not impacted by non-compliant sources since it is implemented within the network, with adjustments of the traffic source being made possible on top of the basic input node control, both being independent from each other which is quite convenient in case of misbehaving source users. Additionally, the mechanism operates with nearly no added overhead, and can be easily implemented in presently available networks.

What is claimed is:

1. Method for performing congestion detection and flow control operations over data traffics including both discardable traffic and non-discardable traffic, in a high speed digital packet switching network including access and transit nodes interconnected by links or trunks, and wherein for any source end-user attached to said network via an entry access node and requesting its traffic to be vehiculated toward a destination end-user also attached to said network via an exit access node, a connection is established which includes so-called in-going (or forward) and returning (or backward) paths set from said entry node to said exit node and in the opposite direction from said exit node to said entry node respectively, which paths might include network transit nodes, said method including:

monitoring the data flow in each transit node over the forward path from said entry node to said exit node for detecting traffic flow congestion in said monitored transit node and in case of flow congestion being detected therein, setting so-called Congestion Indication (CI) bit in a first predefined so-called header field of data packets on the involved forward path down to the exit node;

said method being further characterized in that it includes:

monitoring the incoming data packets entering said exit node, and in case of a set CI indication being detected therein, feeding this indication back to said entry node by setting a Return Congestion Indication (RCI) bit in a second predefined header field in the data packets of the traffic of said backward path;

monitoring the packets received in said entry node over said returning (backward) path and integrating, in said entry node, said RCI bits indications over a predefined period of time, said integration meaning adding or subtracting one unit depending whether said RCI bit is detected to be at binary value one or zero, respectively, said integration producing an integrated RCI indication;

monitoring said predefined time period, and when said time period is over, checking said integrated RCI indication; and,

adjusting the communication bandwidth assigned to said discardable traffic over said forward path, from said entry node to said exit node in a predefined manner, according to said integrated RCI indications.

2. Method for performing congestion detection and flow control operations over data traffics according to claim 1, further characterized in that said communication bandwidth adjustment includes comparing, in said entry node, said integrated RCI indication bits to at least one predefined threshold levels and adjusting the transmission rate over the involved forward path from said entry node to said exit node, accordingly in a predefined manner.

3. A method for performing congestion detection and flow control operations according to claim 2, further characterized in that said thresholding indications are also fed back to the involved source end-user to enable further controlling the flow of behaving end-user.

4. A method for performing congestion detection and flow control operations according to claim 1, 2 or 3 wherein said non-discardable traffic addresses so-called committed traffic whose delivery is guaranteed while said discardable traffic is accepted on the network path as excess traffic to optimize network bandwidth occupation.

5. A system for performing congestion detection and flow control operations over data traffics in a high speed digital

packet switching network including access and transit nodes interconnected by links or trunks, each node including adapters with receive and transmit sections respectively attached to node entering and exiting links or trunks and switching means for transferring data packets from receive to transmit adapter sections, and wherein for any source end-user attached to said network via an entry access node and requesting its traffic to be vehiculated toward a destination end-user also attached to said network via an exit access node, so-called forward and returning (or backward) paths are set from said entry node to said exit node and in the opposite direction from said exit node to said entry node, respectively, which paths might include network transit nodes, each said end-user's traffic being either qualified as high priority level committed traffic whose delivery is guaranteed and assigned predefined transmission rate limits whereby a predefined amount of total transmission bandwidth is being reserved to it accordingly, or qualified low priority level discardable excess traffic and assigned whatever bandwidth is left, said system being characterized in that it includes:

means in said adapter transmit section for dispatching the data packets, each packet including a so-called payload section and a so-called header section, toward output queues based on said priority levels;

means for monitoring the data flow in said output queues in each node over the forward path from said entry node to said exit node, for detecting traffic congestion in said monitored queues and in case of flow congestion being detected therein, setting a so-called Congestion Indication (CI) bit field selected for carrying Explicit Forward Congestion Information (EFCI) in a first predefined header section of forward data packets, on the involved path down to the exit node;

means for monitoring the incoming data packet entering said exit node, and in case of a set CI indication being detected therein setting a Return Congestion Indication (RCI) bit in a second predefined header section field in the data packets of the traffic flowing over said backward path;

means in said entry node for monitoring the packets received from said backward path and for integrating monitored RCI bits over a predefined time-out period;

means for monitoring said time period and at time-out indication, comparing said integrated RCI indication to at least one predefined threshold level and for controlling accordingly the bandwidth adjustment means in the involved receive adapter section of said entry node for adjusting the communication bandwidth assigned to said discardable traffic over said forward path, from said entry node to said exit node in a predefined manner.

6. A system for performing congestion detection and flow control operations over data traffics in a high speed digital packet switching network according to claim 5 further characterized in that said means for monitoring the data flow in said output queues in each node over the forward path from said entry node to said exit node, for detecting traffic congestion in said monitored queues involves means for monitoring at least one predefined queue threshold level.

7. A system for performing congestion detection and flow control operations over data traffics in a high speed digital packet switching network according to claim 6 wherein a set and a reset threshold levels are predefined, whereby the EFCI is set in all packets when a packet arrives and the queue size is greater than the set-threshold and only when

17

the queue size gets below the unset-threshold the EFCI won't be set anymore.

8. A system for performing congestion detection and flow control operations over data traffics in a high speed digital packet switching network according to claim 6 or 7 characterized in that said bandwidth adjustment means includes:

leaky bucket means assigned to discardable traffic data and provided with so-called red token pool means sized at a predefined value M_r , and having a token refill rate R_r , a red token peak rate R and a token refresh period R_p ;

means for decreasing R_r to $R_r * RDF$, RDF being selected among at least one predefined Rate Decrease Factor smaller than one, for each packet received with a congestion indication set, and for setting the red token refresh period R_p to M_r / R_r , wherein M_r is the size of said red token pool; and,

means for increasing R_r by a predefined amount (RIF) of the peak rate R .

9. For use in a packet switching network wherein a first connection is established to enable transmission of packets from a first node to a second node and a second connection is established to enable transmission of packets from the second node to the first node, wherein said second node monitors each packet received on the first connection for the presence of congestion indicators and responds to such indicators by setting a congestion indicator in packets later transported on the second connection and wherein packets may be designated as low priority or high priority packets, a congestion control method implemented in the first node comprising the steps of:

integrating the number of congestion indicators detected in packets received on the second connection over a predefined period of time; and

adjusting the bandwidth allocated to low priority traffic on the first connection as a function of the results of the integrating step.

10. A method as set forth in claim 9 wherein the bandwidth adjusting step comprises the further steps of:

comparing the results of the integrating step to at least one predefined threshold level; and

adjusting the bandwidth in a first predetermined manner if the results exceed the threshold level and in a second predetermined manner if the results don't exceed the threshold level.

11. A method as set forth in claim 10 including the further step of notifying the source of the traffic on the first connection if the predefined threshold level is exceeded to enable the source to further control the flow of traffic to the first node.

12. A method as set forth in any of claims 9-11 wherein the step of adjusting the bandwidth comprises the step of, if the integration results exceed a first level, decreasing the currently allocated bandwidth to a value $R * RDF$ where R is a predefined peak rate and RDF is a predefined Rate Decrease Factor or, if the integration results do not exceed the first level, of incrementing the currently allocated bandwidth by an amount equal to $R * RIF$ where RIF is a predefined Rate Increase Factor.

18

13. For use in a packet switching network wherein a first connection is established to enable transmission of packets from a first node to a second node and a second connection is established to enable transmission of packets from the second node to the first node, wherein said second node monitors each packet received on the first connection for the presence of congestion indicators and responds to such indicators by setting a congestion indicator in packets later transported on the second connection and wherein packets may be designated as low priority or high priority packets, a congestion control system implemented in the first node comprising:

a timer that times out at the conclusion of successive predetermined periods of time;

an integrating circuit which maintains an integration result reflecting the number of congestion indications detected in packets received on said second connection;

congestion detection logic responsive at the conclusion of each of the successive predetermined periods of time to obtain the current integration result and to reset the integrating circuit; and

bandwidth control logic which adjusts the bandwidth allocated to low priority traffic on the first connection as a function of the obtained current integration result.

14. A system as set forth in claim 13 wherein the bandwidth control logic further comprises:

a leaky bucket mechanism for receiving low priority traffic intended for the first connection, the leaky bucket mechanism maintaining a red token pool of predetermined size and having a predefined token refill rate, a predefined red token peak rate R and a predetermined token refresh period, said mechanism including logic for decreasing the token refill rate by a predetermined factor for each packet received on the second connection with a congestion indicator and for setting the red token refresh rate equal to M / R_r where M is the size of the red token pool and R_r is the token refill rate.

15. For use in a packet switching network wherein a first connection is established to enable transmission of packets from a first node to a second node and a second connection is established to enable transmission of packets from the second node to the first node, wherein said second node monitors each packet received on the first connection for the presence of congestion indicators and responds to such indicators by setting a congestion indicator in packets later transported on the second connection and wherein packets may be designated as low priority or high priority packets, a computer program product for use in the first node comprising a computer usable medium having computer readable program code embodied therein for performing congestion control at the first node, the computer readable program code comprising code for integrating the number of congestion indicators detected in packets received on the second connection over a predefined period of time, and adjusting the bandwidth allocated to low priority traffic on the first connection as a function of the results of the integrating operation.

* * * * *



US006279113B1

(12) **United States Patent**
Vaidya

(10) **Patent No.:** US 6,279,113 B1
(45) **Date of Patent:** Aug. 21, 2001

(54) **DYNAMIC SIGNATURE INSPECTION-BASED NETWORK INTRUSION DETECTION**

OTHER PUBLICATIONS

Mukherjee, Biswanath et al., "Network Intrusion Detection," *IEEE Network*, 0890-8044/94, May/Jun. 1994, pp. 26-41.

(75) **Inventor:** Vimal Vaidya, Fremont, CA (US)

* cited by examiner

(73) **Assignee:** Internet Tools, Inc., Fremont, CA (US)

Primary Examiner—Robert Beausoleil
Assistant Examiner—Scott Baderman

(*) **Notice:** Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(74) **Attorney, Agent, or Firm**—Luce, Forward, Hamilton & Scripps LLP

(21) **Appl. No.:** 09/090,774

(22) **Filed:** Jun. 4, 1998

Related U.S. Application Data

(60) Provisional application No. 60/078,759, filed on Mar. 16, 1998, and provisional application No. 60/078,328, filed on Mar. 17, 1998.

(51) **Int. Cl.**⁷ H04L 9/00

(52) **U.S. Cl.** 713/201; 713/154; 709/229

(58) **Field of Search** 713/201, 200, 713/202, 151, 152, 153, 154-176, 177, 178; 709/229; 707/9

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,278,901	*	1/1994	Shieh et al.	713/200
5,414,833	*	5/1995	Hershey et al.	713/201
5,557,742		9/1996	Smaha et al.	395/186
5,720,033		2/1998	Deo	395/186
5,727,146		3/1998	Savoldi et al.	395/187.01
5,948,104	*	9/1999	Gluck et al.	713/200
5,991,881	*	11/1999	Conklin et al.	713/201
6,035,423	*	3/2000	Hodges et al.	714/38
6,088,804	*	7/2000	Hill et al.	713/201

(57) **ABSTRACT**

A signature based dynamic network intrusion detection system (IDS) includes attack signature profiles which are descriptive of characteristics of known network security violations. The attack signature profiles are organized into sets of attack signature profiles according to security requirements of network objects on a network. Each network object is assigned a set of attack signature profiles which is stored in a signature profile memory together with association data indicative of which sets of attack signature profiles correspond to which network objects. A monitoring device monitors network traffic for data addressed to the network objects. Upon detecting a data packet addressed to one of the network objects, packet information is extracted from the data packet. The extracted information is utilized to obtain a set of attack signature profiles corresponding to the network object based on the association data. A virtual processor executes instructions associated with attack signature profiles to determine if the packet is associated with a known network security violation. An attack signature profile generator is utilized to generate additional attack signature profiles configured for processing by the virtual processor in the absence of any corresponding modification of the virtual processor.

20 Claims, 12 Drawing Sheets

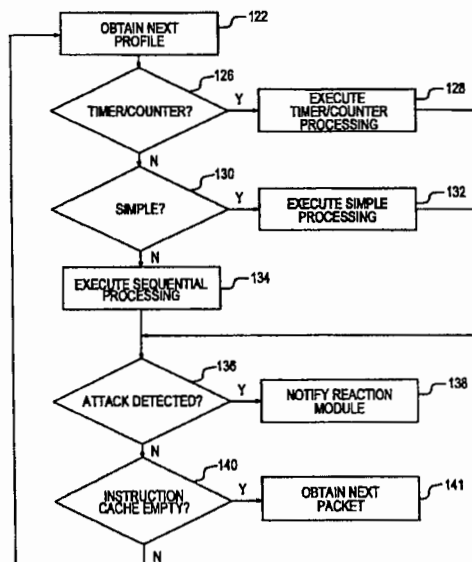
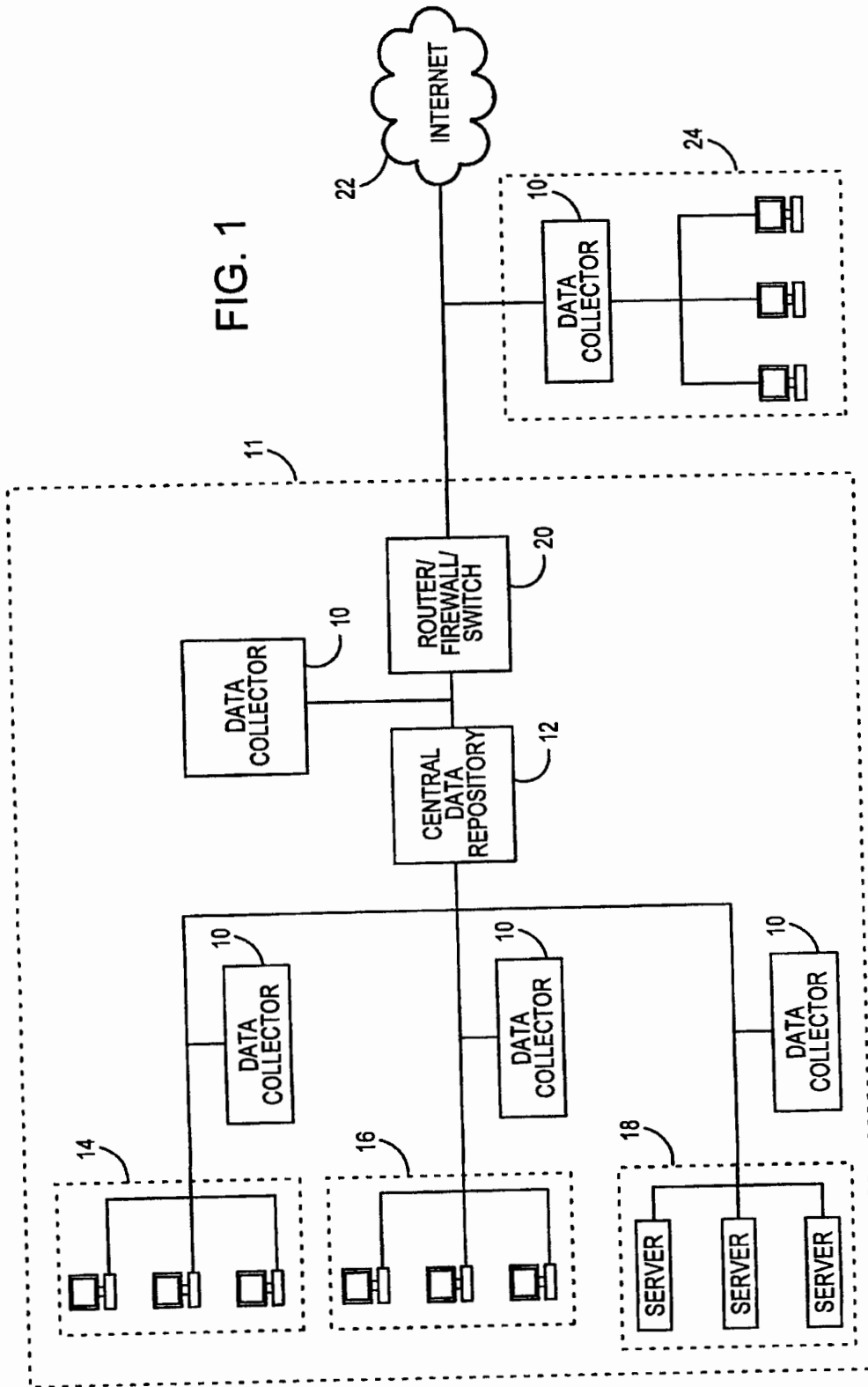


FIG. 1



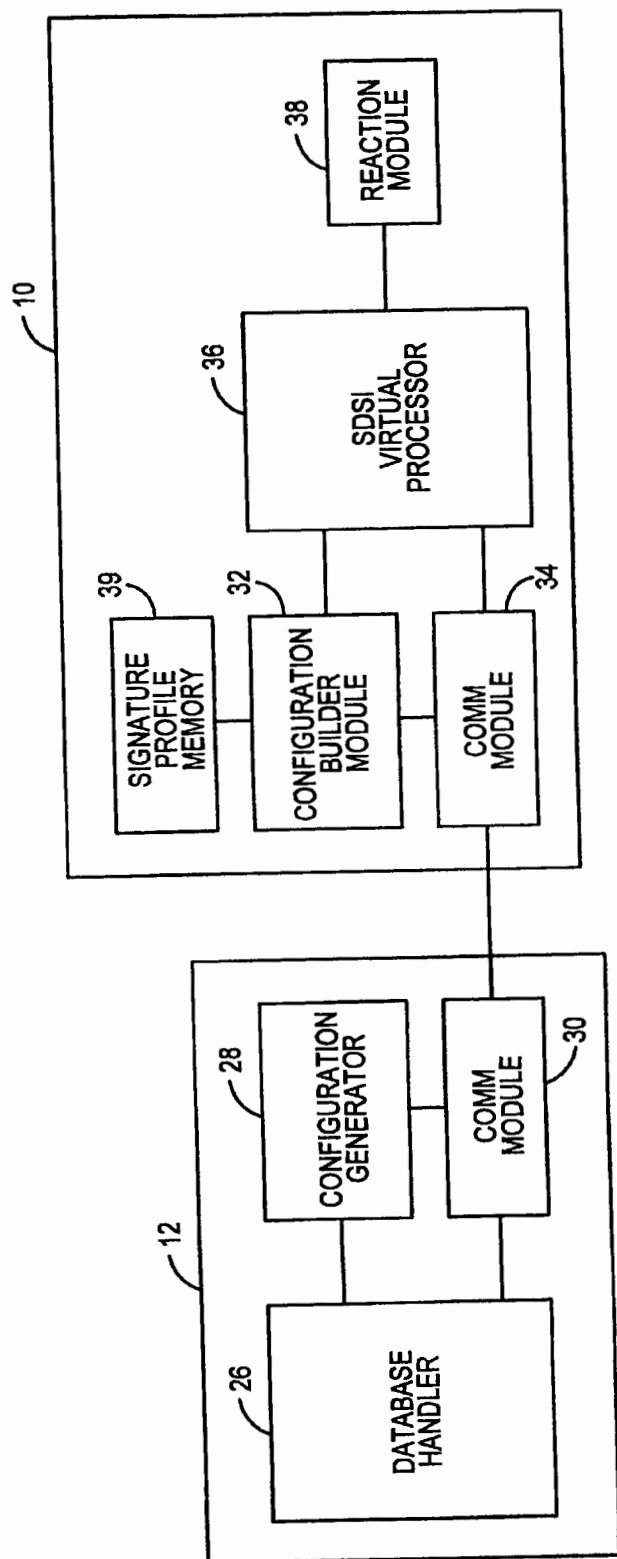


FIG. 2

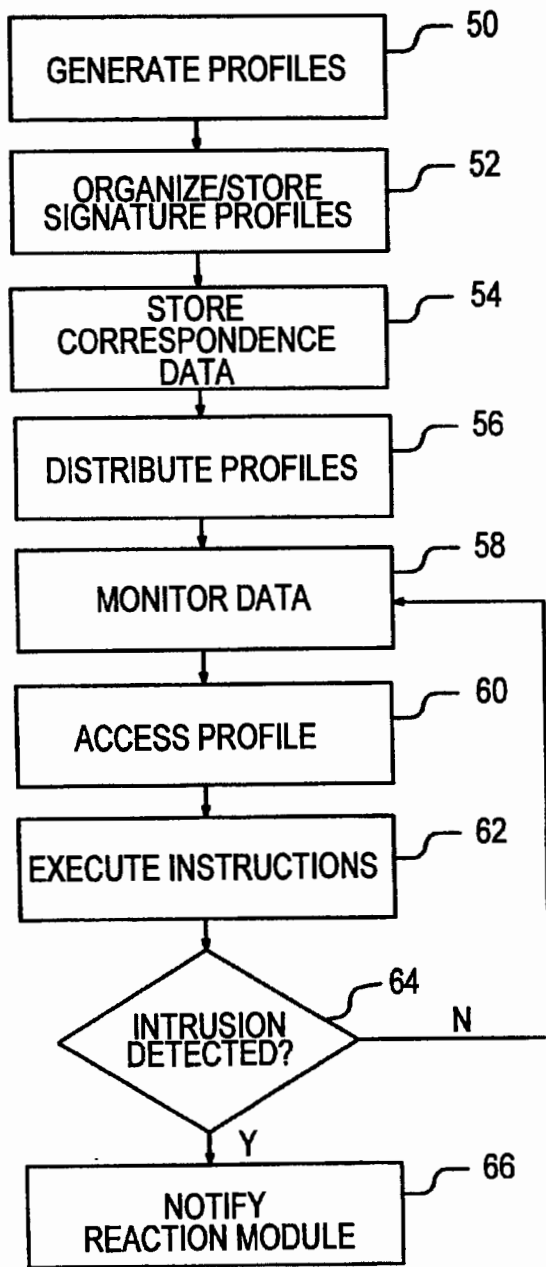


FIG. 3

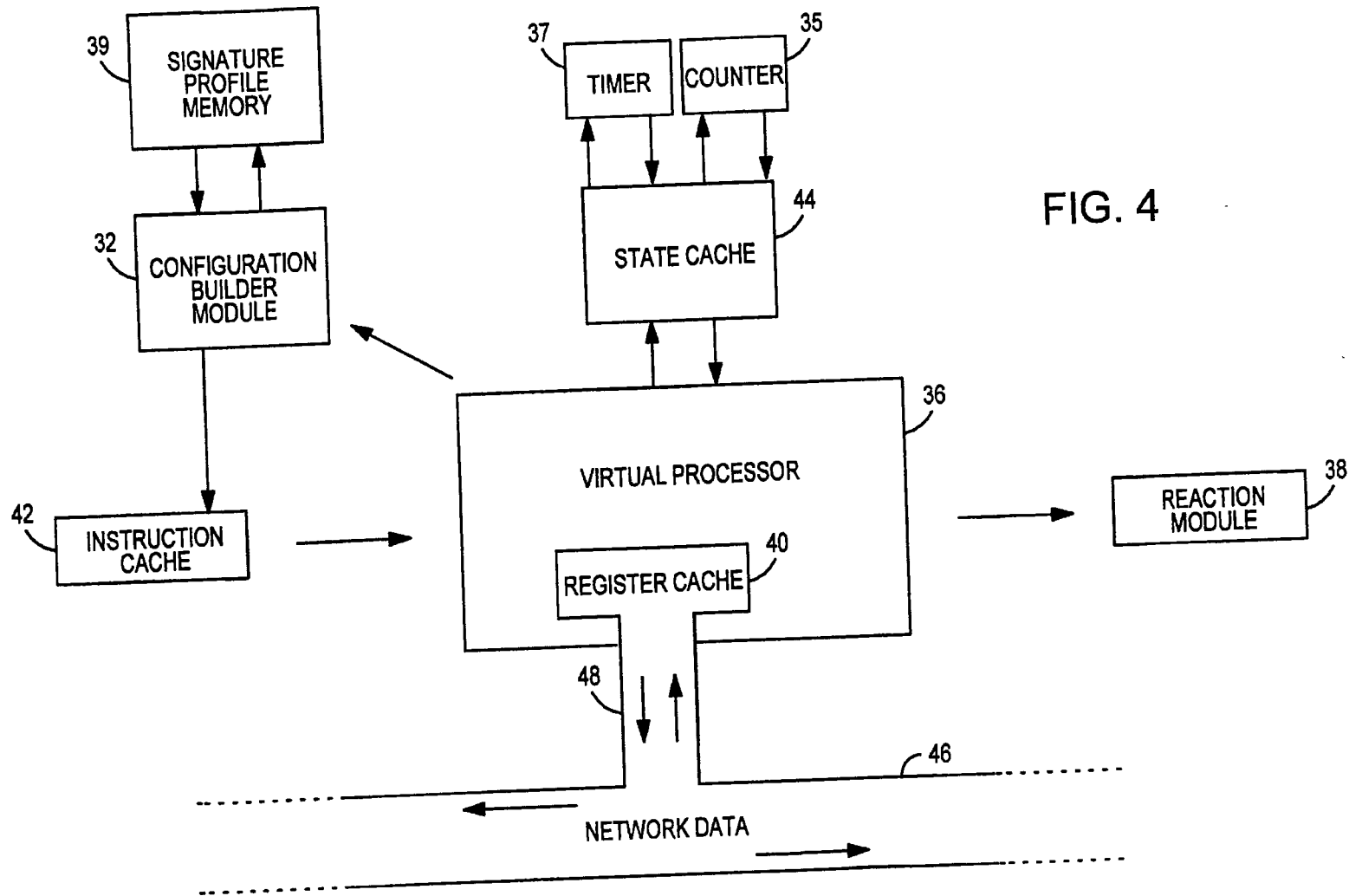


FIG. 4

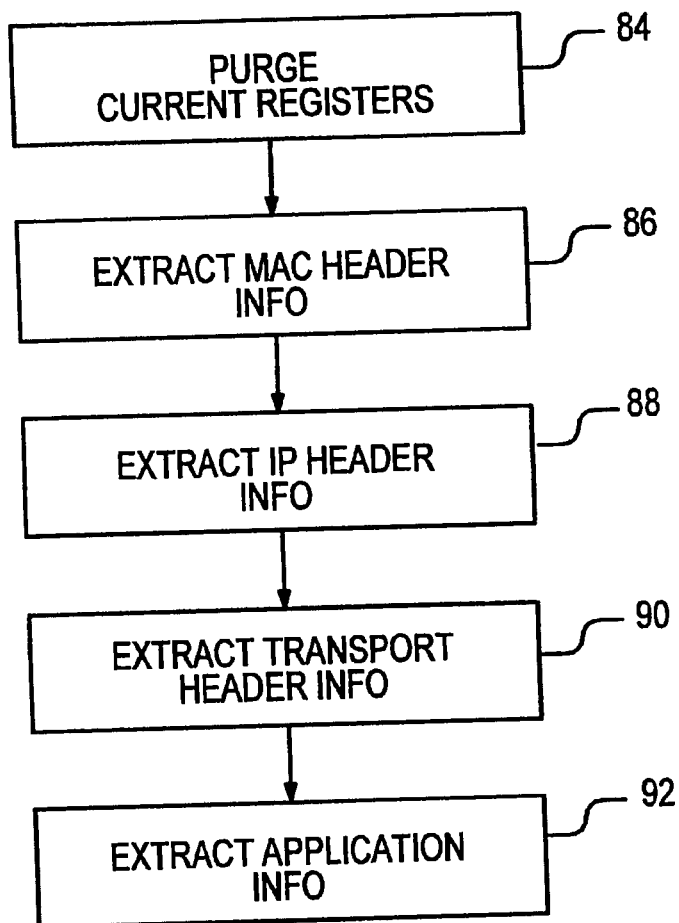


FIG. 5

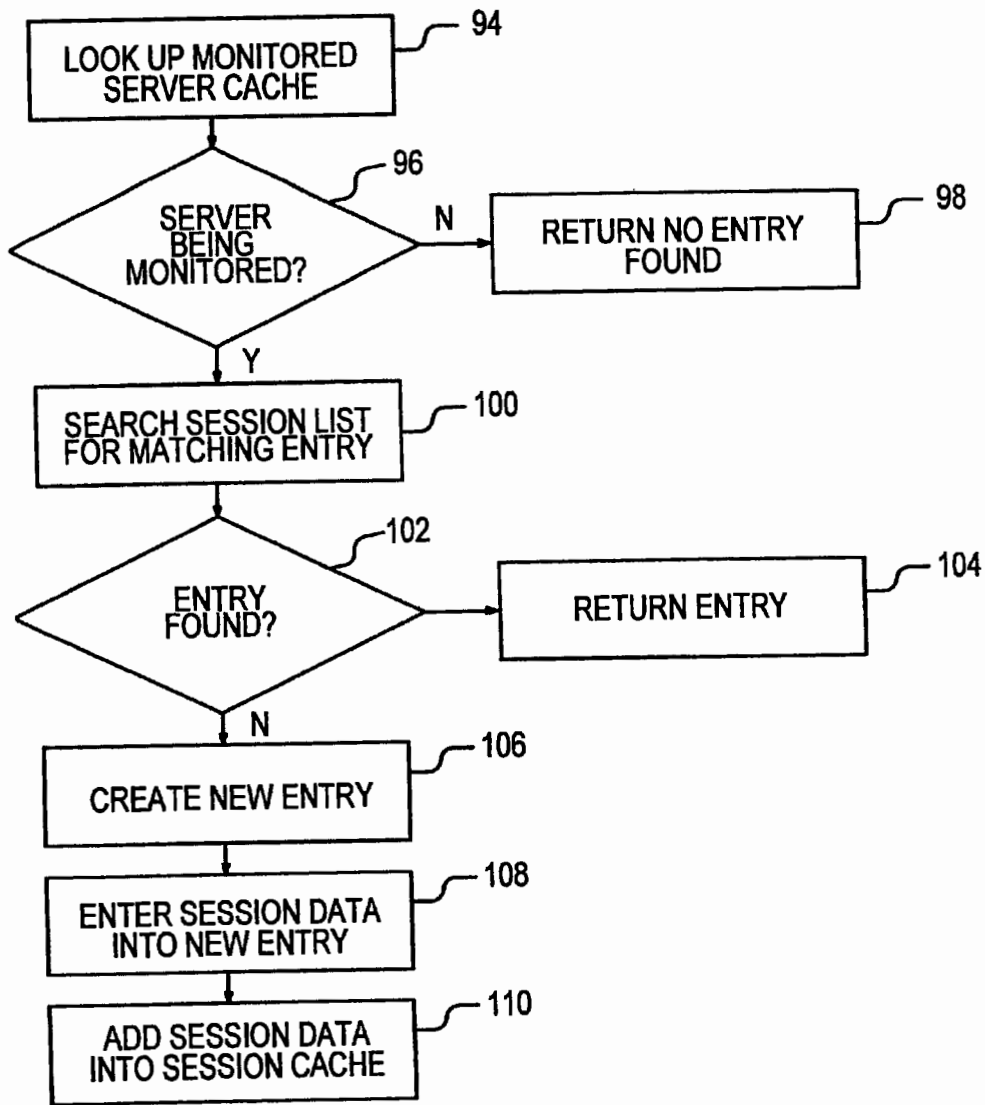


FIG. 6

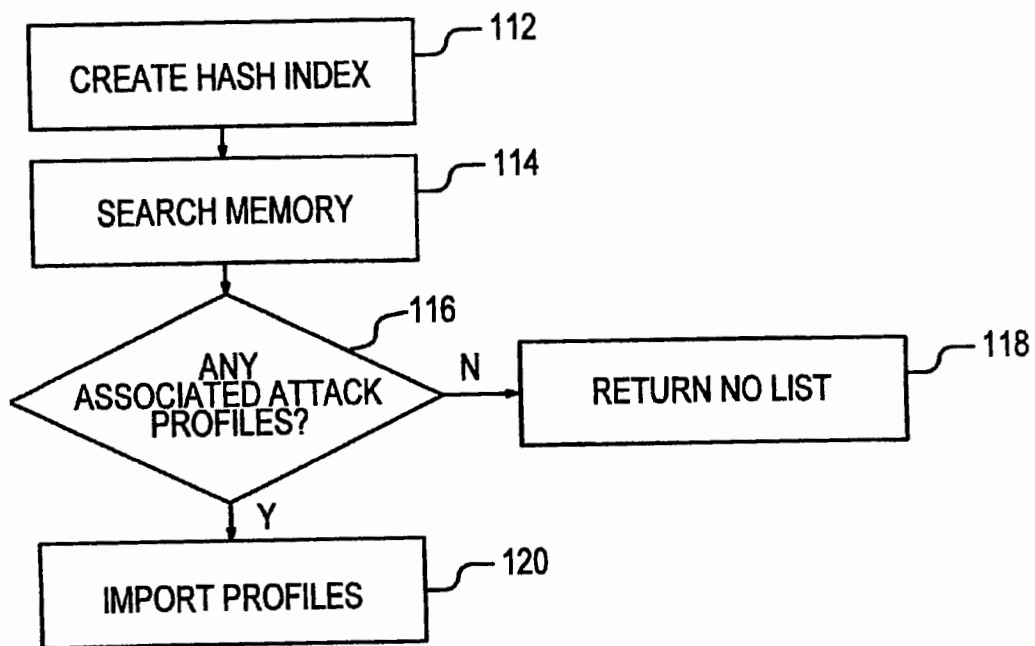


FIG. 7

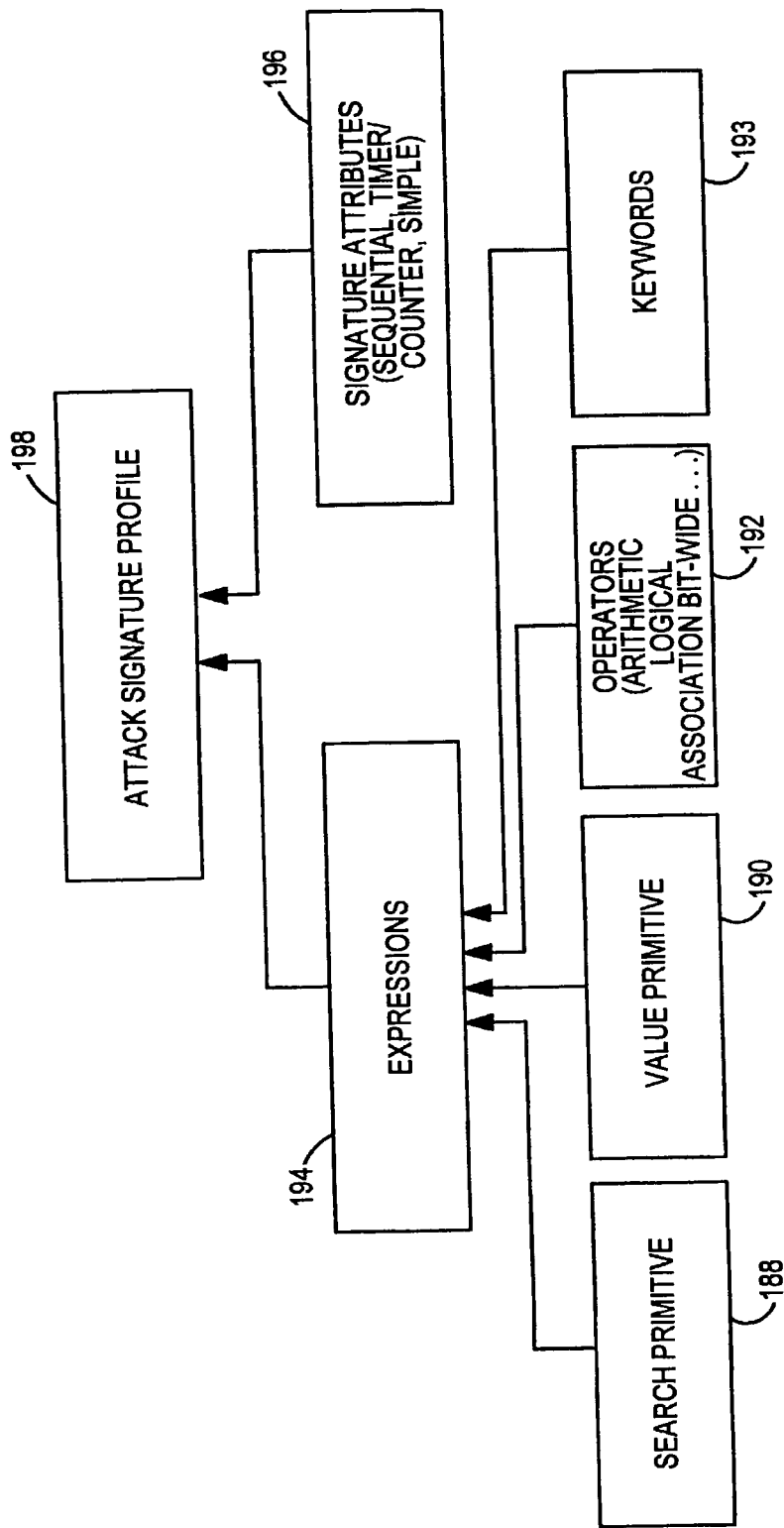


FIG. 8

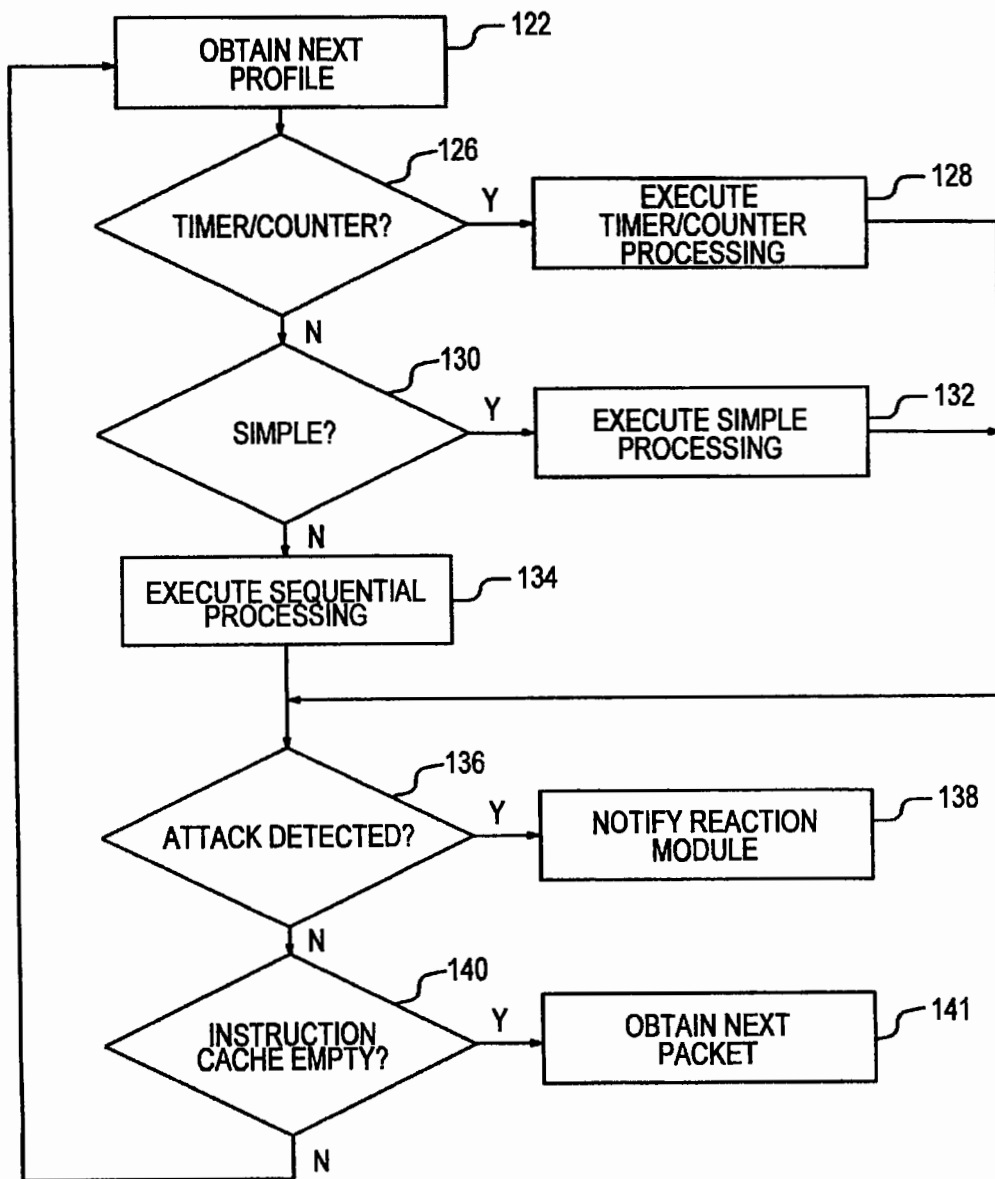


FIG. 9

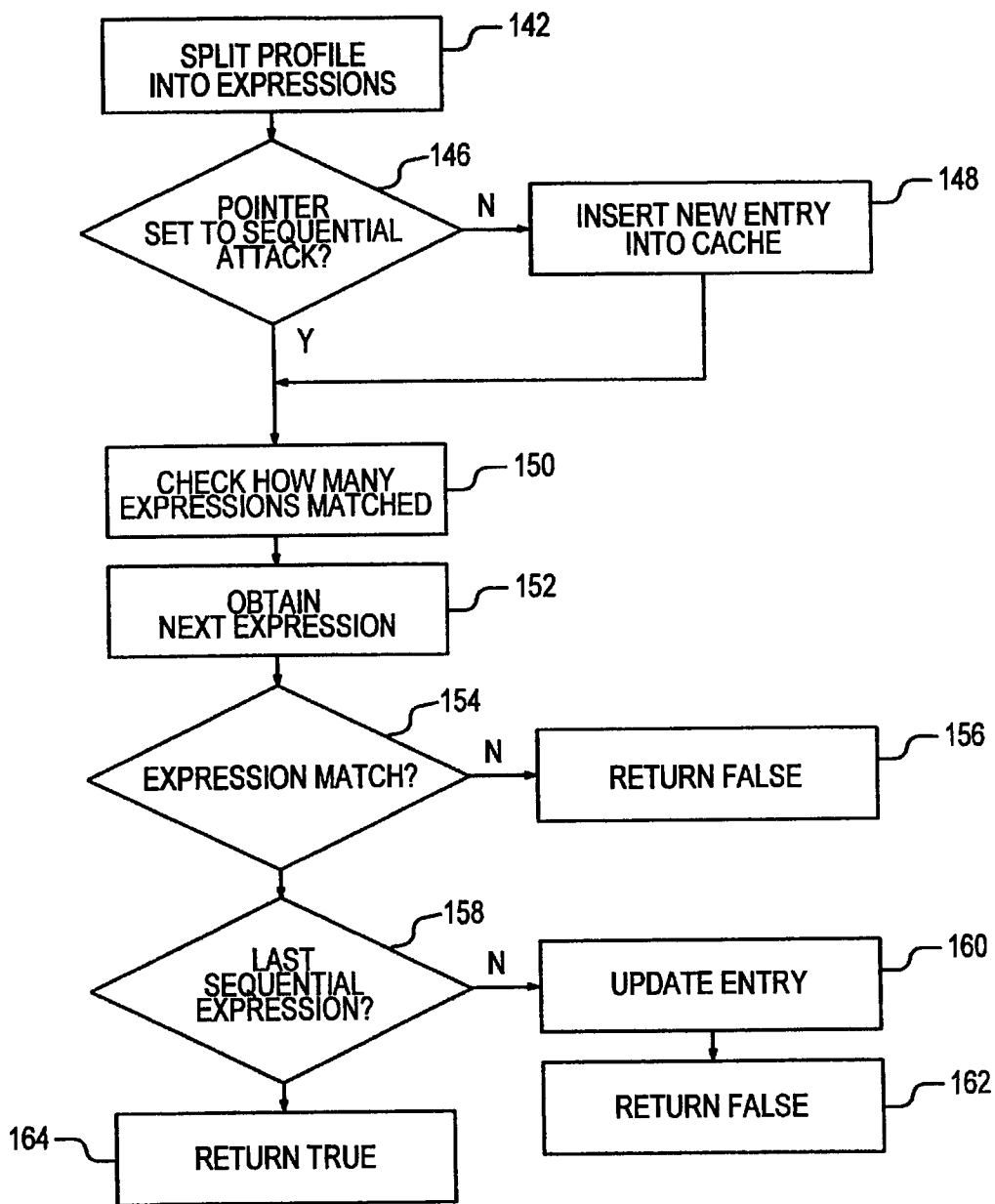


FIG. 10

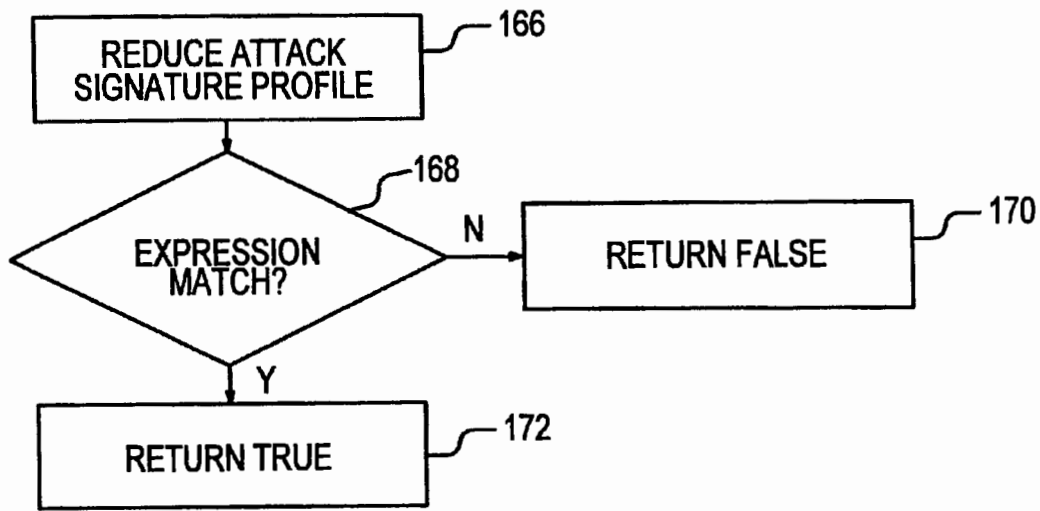


FIG. 11

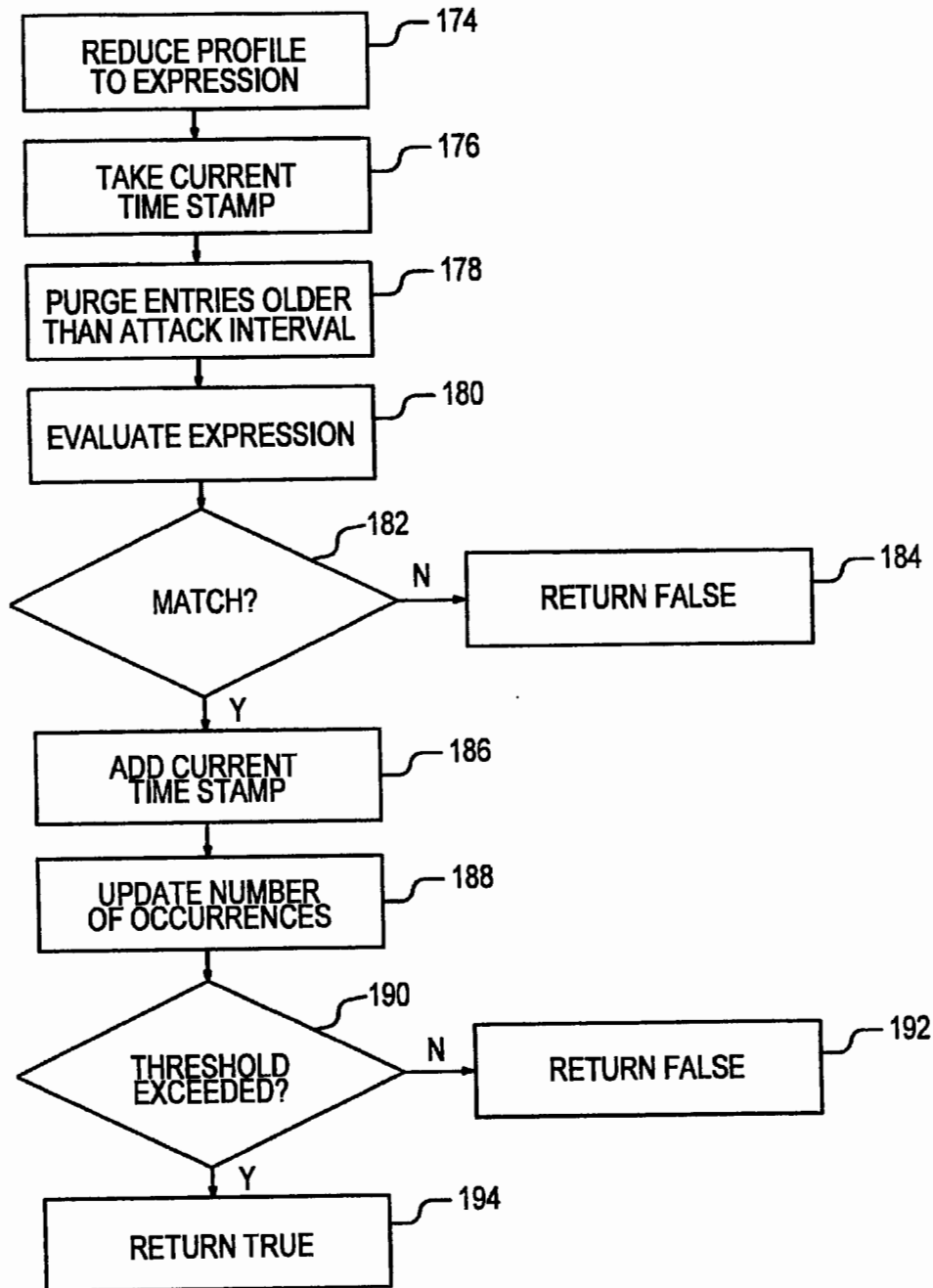


FIG. 12

**DYNAMIC SIGNATURE
INSPECTION-BASED NETWORK
INTRUSION DETECTION**

This application claims the benefit of U.S. Provisional Application No. 60/078,759, filed Mar. 16, 1998, and U.S. Provisional Application No. 60/078,328, filed Mar. 17, 1998.

TECHNICAL FIELD

The present invention relates generally to a method and system for providing security on a communication system and, more particularly, the invention relates to detecting intrusion attempts into system resources by monitoring for attack signatures.

DESCRIPTION OF THE RELATED ART

Computer networks enable multiple communication devices such as computers, fax machines, and modems to communicate with each other. In systems which employ a client-server computing model, server devices can generally be viewed as being a service provider and client devices are consumers of the services. Instead of each device on a network being self-sufficient, resources are contained in servers, which extend capabilities throughout the network. Client devices access the resources necessary to perform functions from the servers. For instance, a user might use a client application to obtain a compound document, perhaps an annual sales report containing spread sheet graphs and explanatory text, where part of the document is located on a first server (the text) and another part is located on a second server (the graphs).

Although the client-server system can provide an efficient means for managing resources of a computer system, significant security issues arise regarding control of access to sensitive material stored on the servers. Large corporate networks often include servers storing sensitive material, access to which must be closely regulated. Often the set of client objects which are permitted access to a particular server application will change over time. A significant need remains for a security system which regulates access to certain objects on a computer system and which provides the flexibility to allow for the changing requirements of security of the system.

U.S. Pat. No. 5,720,033 to Deo describes a security platform for networked processors which limits access to system resources by implementing a rules based system for types of access of security interests to one or more served application programs. The platform provides rule sets, each of which associates an access type with a subject. An example of a subject is a particular user. Optionally, the rule sets also associate an access type with a set of objects, which are specific system resources to which access is sought. Access demands made by a particular served application are compared to the rule sets to determine whether the access demanded is permissible. The platform permits access by a subject to an object if a rule is found for (a) the access type or (b) an access class to which the access type belongs which defines access between (A) either (i) the subject or (ii) a superclass to which the subject belongs and (B) either (i) the object or (ii) the superclass to which the object belongs.

Although the security platform described above provides a partial solution to the network security problem by enabling detection of unauthorized access attempts which are based in the application layer of the OSI model, the security platform is unable to detect network intrusions

based in lower levels of the OSI model. The security platform might be unable to detect an attempt to deliver a malicious data packet capable of causing a malfunction in a network object upon delivery because the security platform regulates access to a network object based on the identity of a subject. Consequently, a subject which is authorized to access a network object can deliver a malicious packet to that network object without being detected. The security platform described above is designed for access control to an object residing on a particular UNIX server. However, the platform is ineffective for detection of network security breaches unrelated to access control, such as transmission of malicious data packets.

U.S. Pat. No. 5,727,146 to Savoldi et al. describes a source address security system for both training and non-training objects, wherein network access to a port is secured by monitoring the source address of packets that are sent as a device attempts to transmit to the port over the network. If the source address of a packet matches an authorized source address assigned to the port, then the device is permitted to access the network. The source address security system requires that the address of all devices authorized to access a network be known so that the source address of a device which has transmitted a particular packet can be compared to source addresses of all authorized devices to determine if the device in question is permitted to access the network. Only if the source address of a device is known to the security system will the device be allowed to access the network.

A static signature database intrusion detection system (IDS) overcomes some of the above described limitations by providing a static signature database engine which includes a set of attack signature processing functions, each of which is configured to detect a specific intrusion type. Each attack signature is descriptive of a pattern which constitutes a known security violation. The system monitors network traffic by sequentially executing every processing function of a database engine for each data packet received over a network. Each processing function of the database engine is integrally associated with a corresponding attack signature making it problematic to incorporate new attack signatures into an existing static signature database. An entirely new database engine must be constructed in order to incorporate a new attack signature. This limitation also results in the built-in IDS not being able to allow addition and customization of new signatures. Furthermore, a built-in database IDS suffers from performance loss due to the sequential execution of every processing function for each packet received over the network. The IDS performance degrades further as more signatures are added to the database engine because of the resulting delay caused by the sequential processing by the static database engine.

What is needed is a network intrusion detection system which provides efficient extensibility to include newly discovered network attack signatures and which allows modifications to recognize new attack signatures without substantially affecting performance of the network intrusion detection.

SUMMARY OF THE INVENTION

A dynamic signature inspection-based network intrusion detection system and method include a processor which is configured such that it is mutually independent from configuring storage of attack signature profiles. In a preferred embodiment, the processor may be implemented either as a virtual processor in software or as an actual hardware

processor. The mutual independence of the processor from the attack signature profiles allows additional attack signature profiles to be integrated into the intrusion detection system without requiring any corresponding modification of the processor. The mutual independence of the processor from the attack signature profiles also enables the system to allocate processing requirements of network monitoring for attack signatures among various sites on the network according to a distribution of network objects in order to maintain high performance of the dynamic signature inspection-based network IDS.

The dynamic signature-based network IDS includes multiple attack signature profiles which are each descriptive of identifiable characteristics associated with particular network intrusion attempts associated with network objects located on the network. Network intrusion attempts include unauthorized attempts to access network objects, unauthorized manipulation of network data, including data transport, alteration or deletion, and attempted delivery of malicious data packets capable of causing a malfunction of a network object. The attack signature profiles can include generic attack and/or customized attack signature profiles for particular network objects on the network. Customized attack signature profiles can be added to a set of generic attack signature profiles without having to modify the processor, thereby facilitating efficient customization of the IDS.

The attack signature profiles are organized into sets of attack signature profiles which are assigned to network objects based on security requirements of the network objects, and these sets of signature profiles are stored in a signature profiles memory. The signature profile memory of a network defines the network data signaling patterns which constitute network intrusion attempts with regard to that network. Association data is stored in the signature profile memory and corresponds each of the network objects to associated set or subset of signature profiles, such that multiple sets of signature profiles are assigned to the set of network objects.

Data transmitted over the network is monitored by a data monitoring device to detect data addressed to the network objects. Upon detecting data addressed to one of the network objects, a set of signature profiles corresponding to that network object is accessed from the signature profile memory based on the association data. At least one attack signature profile from the set of profiles is processed by the processor to determine if the data addressed to the network object is associated with a network intrusion.

In a preferred embodiment multiple data collectors, each of which includes a data monitoring device, an attack signature profile memory, and a processor, are deployed at multiple sites in different segments of the network. A network configuration generator assigns sets of attack signature profiles to each data collector based on the network objects located on the network segment on which each data collector is deployed. A particular data collector monitors network data only for data addressed to the network objects located on that data collector's network segment. By distributing the network monitoring responsibilities among multiple data collectors, high performance of the dynamic signature-based network IDS is maintained. Instead of a single data collector monitoring the entire network data for network intrusion attempts, each data collector only monitors a network segment on which it is located or a point of entry from an open network, such as the Internet.

The dynamic signature-based network IDS employs at least three different types of attack signature profiles: a

sequential, a simple, and a timer-counter based attack signature profile. A simple attack signature profile provides instructions to the processor which, when executed, can detect a single occurrence of an event associated with a network intrusion attempt. If processing of a simple attack signature profile reveals an occurrence of the event, a network intrusion attempt has been detected.

A sequential attack signature profile directs the processor to sequentially execute a series of instructions on data which constitutes at least a portion of an application session. The series of instructions is configured to detect a corresponding sequence of events which collectively are associated with a network intrusion attempt. Upon detecting each event associated with each instruction, the processor stores data indicative of the occurrence of that event in a state cache. The state cache is accessed by the processor to determine whether the entire series of events associated with the network intrusion attempt has occurred.

The timer-counter based attack signature profile directs the processor to execute an instruction which is configured to detect a particular event. The instruction is executed on each packet associated with an application session. A time stamp entry and a counter entry is made for each event detected by executing the instruction. If the number of times the event occurs within a predetermined time interval exceeds a preselected threshold, a network intrusion attempt has been detected.

An advantage of the present invention is that all seven layers of the OSI model are monitored and so an attack based in any of the layers can be detected. Another advantage is that the mutual independence of the processor and the attack signature profile enables efficient customization of the IDS according to the security requirements of a network. Yet another advantage of the present invention is the high performance which the IDS is able to provide on large networks by allocating network monitoring responsibilities to multiple monitoring devices at multiple sites on the network.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram showing a network on which a network IDS according to the current invention is deployed.

FIG. 2 is a block diagram of a data repository and a data collector employed in the operation of the network IDS shown in FIG. 1.

FIG. 3 is a process flow for an operation of a network IDS shown in FIG. 2.

FIG. 4 is a schematic diagram illustrating the operation of a virtual processor shown in FIG. 2.

FIG. 5 is a process flow of a method for building a register cache during the operation of the virtual processor shown in FIG. 4.

FIG. 6 is a process flow for a method of extracting a state cache entry during the operation of the virtual processor shown in FIG. 4.

FIG. 7 is a process flow for a method for building an instruction cache with applicable attack signature profiles.

FIG. 8 is a schematic diagram of the components of an attack signature profiles.

FIG. 9 is a process flow for a method of processing attack signature profile from an instruction cache.

FIG. 10 is a process flow for a method for processing a sequential attack signature profile.

FIG. 11 is a process flow for a method for processing a simple attack signature profile.

FIG. 12 is a process flow for a method for processing a timer/counter based attack signature profile.

DETAILED DESCRIPTION

Referring to FIG. 1, a preferred embodiment of a dynamic network-based signature inspection network Intrusion Detection System (IDS) includes a central data repository 12 and multiple data collectors 10 located on a network such as a Local Area Network 11 (LAN). Although the data collectors 10 are illustrated as stand-alone devices, the function of a data collector can be included on other devices in the network, such as a server or a router/firewall/switch 20. Multiple data collectors 10 are preferred when the LAN 11 includes multiple network objects which the IDS must monitor for network intrusions. As will be discussed in greater detail below, allocating monitoring responsibilities among multiple data collectors 10 in such situations tends to maintain a high performance of the IDS. Two of the data collectors 10 are deployed on first and second LAN segments 14 and 16 each of which includes multiple workstations, a third data collector 10 is located on a server backbone 18 of the LAN 11 to monitor network traffic to and from the servers, a fourth data collector 10 is located proximate to the router/firewall/switch 20 to monitor incoming data to the LAN 11, and a fifth data collector monitors incoming data to a remote network 24.

The data repository 12 polls the data collectors 10 to obtain network security data, which the data repository 12 handles. The data repository 12 also provides an interface for an administrator of the IDS to establish a configuration of network objects on the LAN 11 and to distribute attack signature profiles to the data collectors 10 based on the network configuration. The attack signature profiles are adapted for detecting network data patterns associated with network intrusions which include unauthorized attempts to access network objects, unauthorized manipulation of network data, including data transport, alteration or deletion, and attempted delivery of malicious data packets capable of causing a malfunction in a network object. The remote network 24 is connected to the LAN 11 and is equipped with a data collector 10 which monitors work stations located on the remote network 24 and transmits network security data specific to the remote network back to the data repository 12. Both the remote network 24 and the LAN 11 are connected to the global communications network referred to as the Internet 22.

Referring to FIG. 2, the data repository 12 includes a database handler 26 which polls the data collectors 10 for intrusion detection data and stores the data for future reference. The database handler 26 also generates reports regarding intrusion detection history. A configuration generator 28 is connected to the database handler to enable the network administrator to define the configuration of network objects on the LAN 11 and the remote network 24. The configuration generator 28 also enables the administrator to define the connection of both the LAN 11 and the remote network 24 to the Internet 22. The network objects include devices such as the servers and workstations, as well as routers, firewalls and switches. Network objects further include applications and files stored in memory within those devices. Based on the network configuration data generated by the configuration generator 28, the database handler 26 assigns sets of attack signature profiles to each data collector 10. A communication module 30 is used by the data repository 12 to transmit and receive data to and from the data collectors 10. For example, the communication module 30 downloads network configuration data to the data collectors 10.

Each data collector 10 includes a communication module 34 for transmitting and receiving information to and from the data repository 12. A configuration builder module 32 assigns a set of signature profiles to each network object and stores data representative of associations between network objects and attack signature profile sets in a signature profile memory 39. The configuration builder module 32 accesses the appropriate attack signature profile sets during operation of the data collector 10 and provides the attack signature profiles to a stateful dynamic signature inspection (SDSI) virtual processor 36. The attack signature profiles include a set of instructions which the virtual processor 36 executes to determine whether a particular data packet is associated with a network intrusion. Although a preferred embodiment of the processor employs the software based virtual processor 36 to execute attack signature profiles, a hardware based processor can be employed in the place of the virtual processor 36. If the virtual processor 36 determines that a network intrusion has occurred, it alerts a reaction module 38, which initiates one of several reactions depending on the nature of the attack. The reaction module 38 can either terminate an application session associated with the network intrusion, trace the session, and/or alert the network administrator of the attack. The reaction module 38 is configured to automatically notify the network administrator via e-mail, fax, an SNMP trap, and/or by pager.

With reference to FIGS. 2 and 3, a method for the operation of the dynamic signature inspection network IDS includes the step 50 of generating attack signature profiles. The attack signature profiles can be generic in that they describe generic network intrusion attempts which are common to most networks, or the attack signature profiles can be generated to be specific to a particular network by, for instance, indicating which network objects are not permitted to access other network objects. In step 52 sets of attack signature profiles are organized according to security requirements of each network object. In step 54, corresponding data that are indicative of which objects corresponds to which sets of attack signature profiles are stored in memory of the data repository 12. As noted above, network objects include servers, workstations, applications, files within applications, and devices such as routers, firewalls and switches.

The configuration generator 28 of the data repository 12 is utilized to establish a configuration of network objects. If more than one data collector 10 is deployed on a network, the configuration generator 28 stores information regarding which objects reside on each segment that a data collector 10 is monitoring and the sets of attack signature profiles required by each data collector. In step 56 the communication module 30 of the data repository 12 distributes the signature profiles to the various data collectors 10 throughout the network. Upon receiving a set or sets of attack signature profiles, each data collector 10 stores the set or sets of profiles it receives from the data repository 12 in its signature profile memory 39.

Each data collector 10 monitors network data in step 58 to detect packets addressed to network objects on the network segment on which the data collector 10 is located. For example, referring briefly to FIG. 1, the data collector 10 located on the first network segment 14 monitors network data for packets addressed to those workstations on the first network segment 14. When the data collector 10 detects a data packet addressed to a network object having an associated attack signature profile set in the signature profile memory 39, the data collector accesses the attack signature profile set in step 60 and processes attack signature profiles

in step 62 to determine if the packet is associated with a network intrusion in step 64. The attack signature profile type can be either simple, sequential or a timer/counter based. If in step 64 the data collector 10 determines that the data packet is not associated with a network intrusion, the data collector continues to monitor data in step 58. If a network intrusion is detected, the reaction module is notified in step 66. The reaction module 38 takes steps to trace the application session associated with the data packet, to terminate the session, and/or to notify the network administrator.

With reference to FIG. 4, the operation of the virtual processor 36 includes monitoring network data 46 to determine whether the data is associated with a network intrusion. A register cache 40 temporarily stores information extracted from a data packet which determines which signature profile(s) will be accessed from the signature profile memory 39. The virtual processor 36 obtains a data packet from a queue and extracts MAC header information, IP header information, transport header information, and application information from the data packet. Extraction of the packet information enables the data collector 10 to detect network intrusions based in the different layers of the OSI model.

The virtual processor 36 uses the extracted packet information to determine to which server and application the packet is addressed. The virtual processor 36 communicates the server/application information to the configuration builder module 32, which accesses the applicable set of attack signature profiles from the signature profile memory 39.

The configuration builder module 32 temporarily stores the applicable attack signature profiles in an instruction cache 42. The virtual processor 36 processes the attack signature profiles to determine whether the packet is associated with a network intrusion attempt. A simple attack signature profile might provide instructions to determine if a data packet, which is addressed to server X for access to application Y, has a source address of user Z. In this example, a network administrator has determined that user Z is not authorized to access application Y on server X. If, upon executing the simple attack profile instructions the virtual processor 36 recognizes that the source address for the data packet is user Z, the virtual processor 36 notifies the reaction module 38, which then takes an appropriate action.

Simple attack signature profiles include only a single expression. In the example above the expression can be described as "is source address user Z?" Two other types of attack signature profiles, sequential and timer/counter based, require sequential execution of an instruction or instructions associated with an attack signature profile.

The sequential attack signature profiles include multiple expressions. For instance, these expressions might include "is source address user Z?" and "is user Z attempting to access file A?" Instructions associated with the first expression are executed on a first packet associated with an application session to determine that the packet has the user Z source address. However, if this first packet does not include information that user Z is attempting to access file A in application X, a subsequent packet associated with the same application session will have to be analyzed to determine if user Z is attempting to access file A. An entry is made into a state cache 44 to indicate that the first expression was satisfied. The state cache 44 satisfies the need for a record to be made indicating which expressions in the sequential attack signature profile have been matched in the current application session.

The next packet which the virtual processor 36 determines to be associated with the same application session will cause the virtual processor 36 to fill the instruction cache 42 with the sequential attack signature profile. The sequential attack signature profile includes information which causes virtual processor 36 to access the entry from a state cache 44 indicating that user Z has accessed application Y on server W. Based on the state cache entry, the virtual processor 36 executes instructions associated with the expression "is user Z attempting to access file A?" If the virtual processor 36 determines that this second packet associated with the application session includes data representative of an attempt to access file A, the second expression is satisfied and an unauthorized access attempt by user Z into file A has been recognized.

A timer/counter based attack signature profile directs the virtual processor 36 to execute instructions associated with a single expression on every data packet associated with a particular application session to determine whether an event has occurred a threshold number of times within a predetermined time interval. For instance, a timer/counter based attack signature profile might direct the virtual processor 36 to execute an instruction associated with the expression "is user Z attempting to access file A?" on every packet associated with a session application Y. The instructions also direct the virtual processor 36 to determine whether the number of attempts user Z makes to access file A exceeds 5 attempts within any 10 minute period. The first packet which the virtual processor 36 recognizes as being associated with an attempt by user Z to access file A causes the virtual processor 36 to activate a timer 37 and to set a counter 35 to one. The timer and counter information are entered into the state cache 44. Each subsequent detection of an attempt by user Z to access file A triggers the virtual processor 36 to access the timer and counter information from the state cache 44 and to determine whether the threshold has been met. If the threshold is met, a network intrusion has been detected and the virtual processor 36 notifies the reaction module 38.

Referring to FIG. 5, a method for building a register cache 40 during the operation of the virtual processor 36 includes purging the packet information in the current register in step 84 upon accessing a data packet from the packet queue. In step 86 the MAC header information is extracted from the packet, in step 88 the IP header information is extracted, in step 90 the transport header information is extracted from the packet, and in step 92 the application information is extracted from the data packet. All of the extracted packet information is entered into the register cache 40. The extracted packet information is utilized to create a session cache entry, which is essentially an application session history, and to access an appropriate set of attack signature profiles. The different types of packet information enable generation of attack signatures profiles which can recognize network intrusions based in the different layers of the OSI model.

Referring to FIG. 6, a method for extracting a session entry in the state cache 44 includes utilizing a server IP address to look up the server in a monitored client/server cache (not shown) in step 94 to determine in step 96 whether the server is being monitored. If the server is not being monitored, in step 98 the virtual processor 36 is alerted that no entry was found for the server. If no entry is found for the server, the server is not being monitored for network intrusions and no further steps are taken. If the network object to which the data packet is directed is a client workstation instead of a server, the virtual processor 36 looks up the

workstation in the client/server cache to determine whether the workstation is being monitored.

If the server is being monitored, in step 100 a session list in the state cache 44 is searched for a matching entry. Application information and the server IP address extracted from the packet into the register cache 40 are used to calculate a hash index, and the hash index is used to search for a matching entry from the session list. In step 102, it is determined whether a matching session entry was found. If a matching session entry is found, the entry is returned to the virtual processor 36 in step 104. The session entry might contain a record of timer/counter expressions executed on packets associated with the application session. For instance, the entry might reflect that within the application session a particular file within the application has been accessed ten times in the past twenty minutes. The virtual processor 36 uses this timer/counter information to determine whether a network intrusion is associated with the particular packet. The state cache 44 is also utilized to create a record of executed expressions in a sequential attack signature profile.

If no session entry is found in step 102, a new session entry is created in the session cache 44 in step 106. Session data, which includes any matches identified by executing attack signature profile instructions on a data packet, are entered into the new session entry in step 108 and the session entry is entered into the state cache 44 in step 110.

Referring to FIG. 7, a method for building the instruction cache 42 includes the step 112 of creating a hash index based on the server IP address and the application information in the register cache 40. Alternatively, if the network object being monitored is a workstation, the hash index can be created using an IP address of the workstation. In step 114 the hash index is used to search the signature profile memory 39 for a set of attack signature profiles corresponding to the server and application associated with the packet information in the register cache 40. In step 116 it is determined whether the server and application associated with the packet information correspond to a set of attack signature profiles. If the search reveals no corresponding profile, the virtual processor 36 is informed of the negative search result in step 118 and no further steps are taken with regard to executing attack signature profile instructions on the data packet. If the search identifies a corresponding profile, the attack signature profiles signatures are imported into the instruction cache in step 120.

With reference to FIG. 8, an attack signature profile 198 can be represented as at least one expression 194 in combination with a signature attribute 196, wherein the expressions can be composed of search primitives 188, value primitives 190, and operators 192. In a preferred mode, the expressions also include keywords 193. An example of an expression might be as follows: (IP AND S1 and (V1>200)), wherein "IP" is a keyword referring to a packet utilizing IP/TCP protocol, "S1" is a search primitive referring to user A, "AND" is a conjunctive operator, ">200" is an operator for indicating a value greater than 200, and "V1" is a value primitive referring to a packet length. Taken together, the entire expression describes a data packet which utilizes IP/TCP protocol, has a source address of user A and which has a packet length of greater than 200 bits.

The attribute 196 of an attack signature can be either sequential, timer/counter based, or simple. A simple attack signature attribute indicates that a attack signature profile consists of a single expression with an instruction is executed by the virtual processor 36 only once. A timer/counter based signature indicates that a single expression

instruction is executed sequentially on each data packet associated with an application session until either the session is terminated or an intrusion is recognized. The timer 37 is used to enter a time stamp into a state cache entry each time an execution of a timer/counter expression instruction detects an event associated with an application session. The counter 35 logs and tracks the number of events within the predetermined time interval each time an event is detected by an execution of the timer/counter based instruction. Upon each execution of the timer/counter based instruction, a state cache entry associated with the application session being monitored is referenced to determine whether previous executions of the timer/counter based instruction together with the present execution have caused the threshold number of events to be reached within the predetermined time interval.

The sequential signature attribute refers to multiple expressions which are sequentially executed on successively transmitted data packets associated with an application session. If each of the expressions detects the event it was designed to detect, a network intrusion has been detected.

A more formal description of an attack signature in a loose BNR parsing grammar follows:

```

Pattern      := Hex or ASCII string of characters
Offset       := integer
Protocol     := one of the communication protocols, ie. MAC-layer
              Network-layer, Transport-layer, or Application-layer
Extract_Type:= Byte, Word, Long Word or String
Header_Field:= Predefined keywords for communication
              protocol header fields
Variable_Name:= ASCII character string Name
SP           := <Pattern, Offset, Protocol> . . . Search Primitive
VP          := <Extract_Type, Offset, Protocol> . . . Value Primitive
OP          := <Logical> | <Arithmetic> | <Bit-wide> |
              <Association> | . . . Operators
Basic_Expression := <SP> | <OP> | <Header_Field> | <SP OP SP>
              | <SP OP VP> | <SP OP Header_Field>
Assignment := <Variable_Name> "=" <Basic_Expression>
Complex_Expression := { (<Basic_Expression> OP <Basic_Expression> ) . . . }
Expression  := <Complex_Expression> | <Complex_Expression> ";"
              { (<Assignment> ";" ) . . . }
Signature_Attributes := <Simple> | <Counter-Timer-Based> |
              <Sequential-occurrence>
Attack_Signature := <Signature_Attribute> { <Expression> . . . }
    
```

With reference to FIG. 9, a method for processing attack signature profiles includes obtaining an attack signature profile from the instruction cache 42 in step 122. As previously noted, the attack signature profiles in the instruction cache 42 were accessed from the signature profile memory 39 based on the IP address of the server to which the packet was addressed and the application in the server to which the packet was directed. It is not necessary that the monitored network object be an application within a server. The object could be any network object, such as a particular server, a workstation, a firewall or a router, or a particular file within an application of the workstation.

In step 126 the virtual processor 36 determines if the attack signature profile has a timer/counter based attribute. If the attack signature profile has a timer/counter based attribute, in step 128 the virtual processor 136 executes timer/counter processing. If the profile's attribute is not timer/counter based, and if in step 130 the virtual processor 36 determines that the attack signature profile has a simple attribute, the virtual processor 36 executes simple signature processing in step 132. If the signature attribute is neither simple nor timer/counter based, the virtual processor 36

11

executes sequential processing in step 134. Although only simple, sequential, and timer/counter based attributes have been discussed, other signature attributes can be incorporated into the present invention.

In step 136 the virtual processor 36 determines if the execution of the attack signature has revealed a network intrusion. If the data collector 10 recognizes a network intrusion, in step 138 the reaction module 38 is notified. If no attack has been detected, in step 140 the virtual processor 36 determines if the instruction cache 42 is empty. If the instruction cache is not empty, the virtual processor 36 returns to step 122 and accesses the next attack signature profile. If the instruction cache 42 is empty, the next packet in the queue 48 is obtained in step 141 to extract packet information into the register cache 40.

Referring to FIG. 10, a method for processing a sequential attack signature profile includes the step 142 of splitting the attack signature profile into expressions. As previously discussed, a sequential attack signature profile is composed of multiple component expressions which are sequentially evaluated to determine if each expression matches a data packet associated with a particular application session. In step 146 the virtual processor 36 determines whether a pointer is set to the sequential attack signature profile in the state cache 44. If the pointer is not set to the sequential attack signature profile, in step 148 an entry is made in the state cache 44 so that a pointer is set to the sequential attack signature profile and the entry parameters are initialized. In step 150, the virtual processor 36 references a state cache entry 44 to determine how many of the expressions have already been matched to data packets associated with the currently monitored application session.

In response to the state cache entry, the virtual processor 36 obtains the next sequential expression from an expression list in step 152. For example, an attack signature profile might include expressions A, B, and C. Expression instruction A was executed and found to match a first packet associated with an application session and expression instruction B was executed and found to match a second packet associated with the application session. Upon receiving a third packet associated with the application session and after referencing the state cache entry to obtain the information that expressions A and B have been matched, the virtual processor 36 obtains the third expression to determine if it matches the third packet. It should be noted that expressions A, B, and C need not be found to match three consecutive data packets associated with an application session. Rather, expression A must be found to match a packet which precedes a packet found to match expression B or C, and B must be found to match a data packet which precedes a packet found to match expression C.

In step 154, after executing an expression instruction, the virtual processor 36 determines whether the expression matches the data packet. If the expression does not match, the virtual processor 36 returns a false value in step 156. If the expression matches, a determination is made in step 158 whether the expression was the last sequential expression. In step 160, the virtual processor 36 updates the entry in the state cache 44 to reflect the match of the expression to the data packet if it is determined that the executed expression is not the last sequential expression and in step 162 the virtual processor returns a value of false. If the expression is the last sequential expression, in step 164 the virtual processor 36 returns a value of true to indicate that a network intrusion has been detected.

The processing of a simple attack signature profile is similar to the processing of a single expression of a sequen-

12

tial attack signature. Referring to FIG. 11, the attack signature profile is reduced to an expression in step 166. After executing the expression instruction, the virtual processor 36 determines whether the expression matches a data packet associated with an application session in step 168. If the expression matches the packet, in step 172 the virtual processor 36 returns a value of true and the reaction module 38 is notified of a network intrusion. If the expression does not match, the virtual processor 36 returns a value of false in step 170.

With reference to FIG. 12, a method for processing a timer/counter based attack signature profile includes the step 174 of reducing the profile to an expression. In step 176 the virtual processor 36 utilizes the timer 37 to make a current time stamp for the data packet being evaluated. Entries in the state cache 44 that are older than an attack interval are purged from the state cache 44 in step 178. Purging stale entries involves comparing a time interval between time stamps associated with entries and the current time. If the actual time interval associated with an entry is greater than the attack signature time interval, that entry is purged from the state cache 44.

In step 180 the expression is evaluated to determine in step 182 if the expression matches the packet currently being analyzed. If the expression does not match, the virtual processor 36 returns a value of false in step 184. If the expression matches the packet, the virtual processor returns a value of true and adds the current time stamp to the application session entry in the state cache 44 in step 186. In step 188 the counter 35 is utilized to update the number of events recognized by execution of the timer/counter expression instruction on data packets associated with the current application session. A determination is made in step 190 whether, after the number of event occurrences has been updated, the threshold number of events has been detected within the predetermined time interval. A value of false is returned in step 192 if the threshold has not been reached. If the threshold has been reached, in step 194, the virtual processor 36 returns a true value to indicate that a timer/counter based network intrusion has been detected.

What is claimed is:

1. A method for detecting network intrusion attempts associated with network objects on a communications network including the steps of:
 - storing a list of attack signature profiles descriptive of attack signatures associated with said network intrusion attempts;
 - storing corresponding data representative of a correspondence between subsets of said attack signature profiles and said network objects such that each network object has a corresponding stored subset of attack signature profiles and more than one subset of attack signature profiles corresponds to network objects;
 - monitoring network traffic transmitted over said communications network for data addressed to one of said network objects;
 - in response to detecting said data addressed to said network object, accessing a subset of attack signature profiles corresponding to said network object based on said correspondence data; and
 - executing at least one attack signature profile included in said subset corresponding to said network object to determine if said data addressed to said network object is associated with a network intrusion attempt.
2. The method of claim 1 wherein said executing step includes utilizing a processor to execute said at least one

13

attack signature profile, the method further comprising the step of generating additional attack signature profiles to be added to said subsets of attack signature profiles in the absence of modifying said processor.

3. The method of claim 2 wherein said generating step includes generating an additional attack signature profile configured to recognize an occurrence of a predetermined threshold number of events within a predetermined time interval, said occurrence of said predetermined threshold number of events within said predetermined time interval constituting said network intrusion attempt.

4. The method of claim 1 wherein said executing step includes determining whether a particular sequence of events occurs which constitutes said network intrusion attempt.

5. The method of claim 1 wherein said steps of storing said list of attack signature profiles and storing said correspondence data include storing said subsets of said attack signature profiles and subsets of said correspondence data at a plurality of sites in different segments of said networks according to a distribution of said network objects on said network.

6. The method of claim 5 wherein said monitoring step includes monitoring network traffic at one of said plurality of sites for data addressed to a subset of said plurality of network objects having corresponding subsets of said attack signature profiles and corresponding subsets of said correspondence data stored at said site.

7. The method of claim 1 further comprising the step of alerting a network administrator if it is determined in said executing step that said data addressed to said network object is associated with said network intrusion attempt.

8. A network-based dynamic signature inspection system for detecting attack signatures on a network comprising:

a data monitoring device configured to detect network data addressed to a first set of network objects, said monitoring device having an input for receiving said data and an output for signaling a detection of said data; signature profile memory including:

- a) attack signature profiles descriptive of network signaling patterns which constitute said attack signatures, each attack signature profile being configured to enable recognition of one of said attack signatures, each attack signature being associated with a known network security violation; and
- b) association data corresponding each of said first set of network objects to an associated subset of said attack signature profiles such that more than one of said subsets of said attack signature profiles corresponds to said first set of network objects; and

processor means, responsive to said detection signaling, for processing an attack signature profile included in a subset of said signature profiles assigned to one of said first set of network objects, reception of a detection signal indicative of a detection by said monitoring device of data addressed to said network object triggering access by said processor means to said subset of said signature profiles assigned to said network object based on said association data.

9. The system of claim 8 further comprising an attack signature profile generator enabled to generate additional attack signature profiles to be included in said subsets of attack signature profiles, said additional attack signature profiles being configured for processing by said processor means in the absence of any corresponding modification of said processor means.

10. The system of claim 9 wherein said attack signature profile generator is further configured to generate said

14

additional attack signature profiles for said first set of network objects based on security requirements of said first set of network objects.

11. The system of claim 9 further comprising a state cache connected to said processor means, said state cache having memory for storage of data representative of attack signature profile processing results.

12. The system of claim 11 wherein said attack signature profile generator is configured to generate a sequential attack signature profile with directions to said processor means to sequentially execute a set of instructions and to store results of each instruction execution in said state cache, a sequential occurrence of events detected by said execution of said instructions being indicative of a known network security violation.

13. The system of claim 8 further comprising an intrusion detection alert mechanism in communicative contact with said processing means, said detection alert mechanism being configured to perform a predetermined act if said processing of said attack signature profile reveals a network intrusion, said predetermined act being one of alerting a network administrator, denying access to said network object, or tracing an application session associated with said network intrusion.

14. A network-based dynamic signature inspection system for detecting attack signatures on a network comprising:

a data monitoring device configured to detect network data addressed to a first set of network objects, said monitoring device having an input for receiving said data and an output for signaling a detection of said data; signature profile memory including:

- a) attack signature profiles descriptive of network signaling patterns which constitute said attack signatures, each attack signature profile being configured to enable recognition of one of said attack signatures, each attack signature being associated with a known network security violation; and
- b) association data corresponding each of said first set of network objects to an associated subset of said attack signature profiles such that more than one of said subset of said attack signature profiles corresponds to said first set of network objects; and

processor means, responsive to said detection signaling, for processing an attack signature profile included in a subset of said signature profiles assigned to one of said first set of network objects, reception of a detection signal indicative of a detection by said monitoring device of data addressed to said network object triggering access by said processor means to said subset of said signature profiles assigned to said network object based on said association data wherein said data monitoring device, said signature profile memory, and said processor means are all contained in a first data collector located on a first network segment on which said first set of said network objects reside, said system further comprising:

a second data collector including a second data monitoring device, a second signature profile memory, and second processor means, said second data collector being located on a second network segment including a second set of said network objects, said second processor means being a duplicate of said first processor means; and

a network configuration generator configured to assign a first plurality of said signature profile subsets to said first data collector based on a configuration of said first

15

set of network objects and to assign a second plurality of signature profile subsets to said second data collector based on a configuration of said second set of network objects.

15. A method for providing network intrusion detection 5 on a network including first and second network objects comprising the steps of:

storing first and second sets of attack signature profiles associated respectively with first and second network objects at a first site on said network, each attack signature profile being configured to detect a network signaling pattern associated with a known network security violation; 10

monitoring network traffic at said first site for data addressed to one of said first and second network objects; 15

upon detecting data addressed to said first network object, accessing said first set of attack signature profiles;

utilizing a processor to execute an attack signature profile from said first set of attack signature profiles; 20

determining whether said execution of said attack signature profile reveals a known network security violation; and

generating additional attack signature profiles configured to be executed by said processor in the absence of modifying said processor. 25

16. The method of claim 15 further comprising the steps of:

deploying a duplicate of said processor at a second site on said network; 30

storing a third set of attack signature profiles associated with a third network object at said second site;

16

monitoring said network traffic at said second site for network data addressed to said third network object; and

executing at least one attack signature profile in said third set of attack signature profiles at said second site upon detecting said network data addressed to said third network object.

17. The method of claim 15 wherein said executing step includes determining whether a predetermined number of events occur within a predetermined time interval.

18. The method of claim 15 wherein said step of utilizing said processor to execute said attack signature profile includes:

translating said attack signature profile into a set of instructions to be sequentially executed to enable recognition of a set of sequentially occurring events which collectively constitute said known network security violation;

sequentially executing said set of instructions; and upon recognizing each of said set of events, storing data representative of an occurrence of said each event.

19. The method of claim 18 wherein said determining step includes determining whether said known security violation has occurred based on said stored data representative of said occurrence of said each event.

20. A computer system comprising:

a plurality of attack signature profiles comprising machine readable data corresponding to attack signatures associated with network intrusion attempts; and

corresponding data comprising machine readable data representative of a correspondence between a plurality of network objects and subsets of attack signature profiles.

* * * * *



US006363056B1

(12) **United States Patent**
Beigi et al.

(10) **Patent No.:** US 6,363,056 B1
(45) **Date of Patent:** Mar. 26, 2002

(54) **LOW OVERHEAD CONTINUOUS MONITORING OF NETWORK PERFORMANCE**

Primary Examiner—Ricky Ngo
(74) *Attorney, Agent, or Firm*—Louis P Herzberg

(75) **Inventors:** Mandis Sadr Mohammad Beigi, Tarrytown; Raymond Byars Jennings, Ossining; Dinesh Chandra Verma, Millwood, all of NY (US)

(73) **Assignee:** International Business Machines Corporation, Armonk, NY (US)

(*) **Notice:** Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) **Appl. No.:** 09/115,438

(22) **Filed:** Jul. 15, 1998

(51) **Int. Cl.⁷** H04L 12/28; H04L 12/56

(52) **U.S. Cl.** 370/252; 709/224

(58) **Field of Search** 370/241, 242, 370/243, 244, 245, 252, 253, 401, 400; 709/224, 235, 223

(56) **References Cited**

U.S. PATENT DOCUMENTS

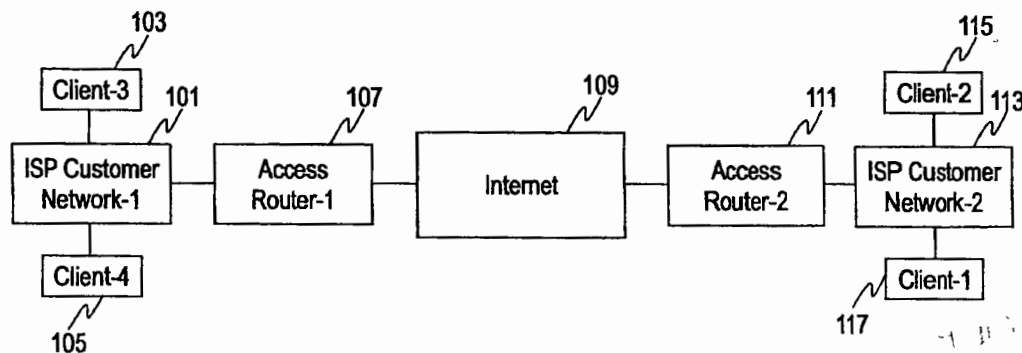
- 5,886,643 A * 3/2000 Diebboll et al. 340/825.08
- 6,058,102 A * 5/2000 Drysdale et al. 370/252
- 6,108,782 A * 8/2000 Fletcher et al. 713/153

* cited by examiner

(57) **ABSTRACT**

A method, apparatus, article of manufacture and computer product for low-overhead continuous monitoring of network performance in an intranet or Internet topology. Probe packets are sent from ingress access routers where they are received and processed by egress access routers. Probe packets are generated by copying every Nth packet being sent by an ingress access router. In the event an access router does not receive the probe packet, the probe packet is discarded through normal network delivery mechanisms. Network delay is determined by subtracting the time that a probe packet was received with the time stamp enclosed in the probe packet. Round trip time is established by reflecting the probe packet back to the originating access router and computing the round trip time. Bandwidth monitoring is achieved by using the number of probe packets received to estimate the expected amount of network traffic to be received. Fault monitoring is accomplished by comparing the number of probe packets received with the number of actual packets received. When the low overhead mechanisms indicate that network delays or faults exist, a heavy weight monitoring protocol is started between two access routers in question.

70 Claims, 12 Drawing Sheets



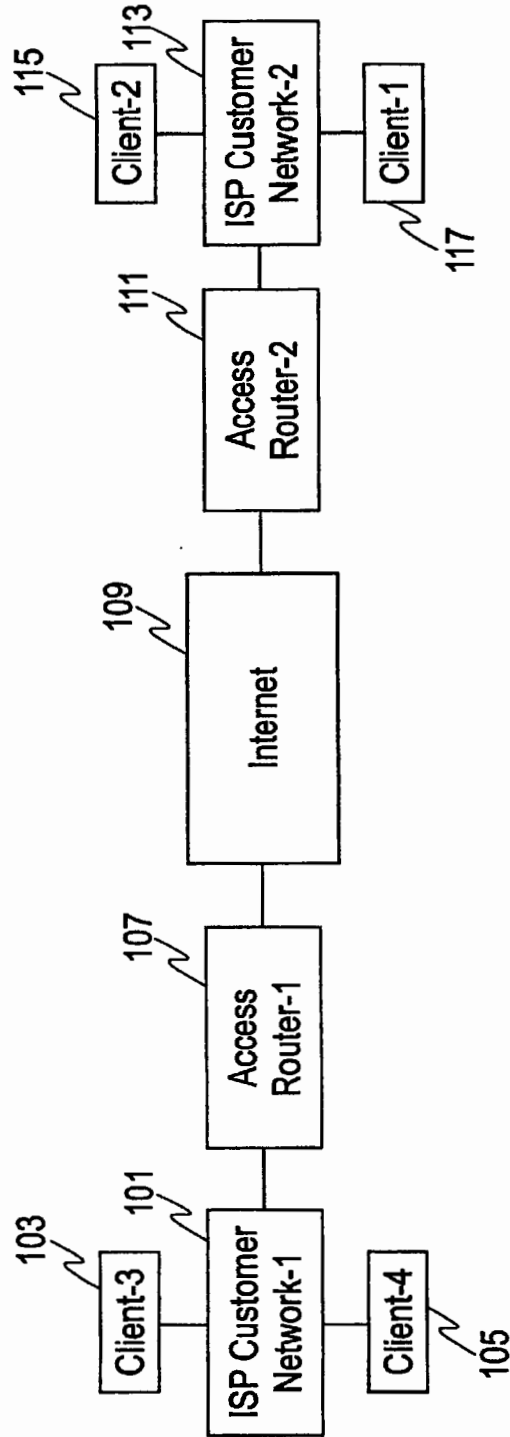


Fig. 1

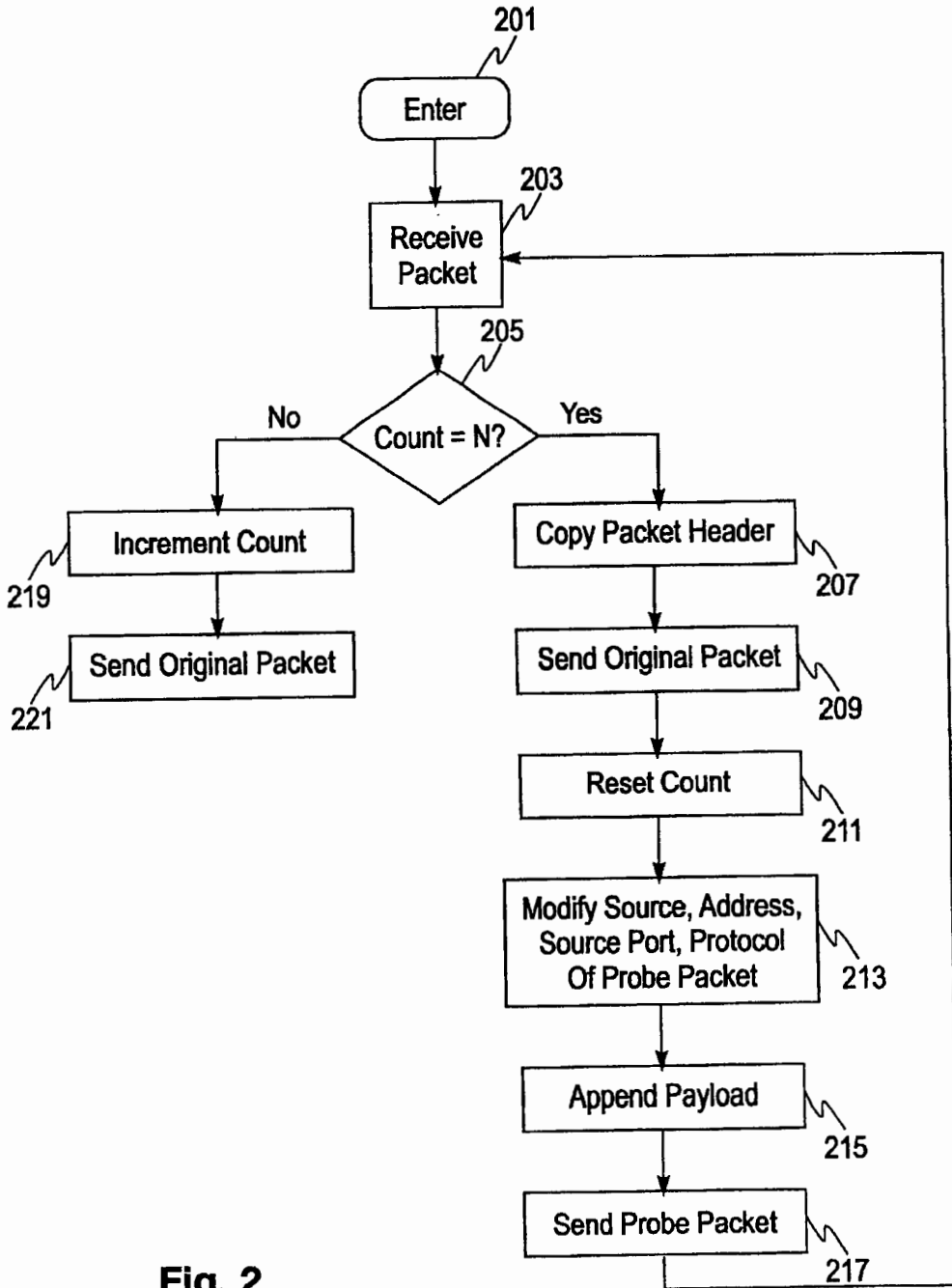


Fig. 2

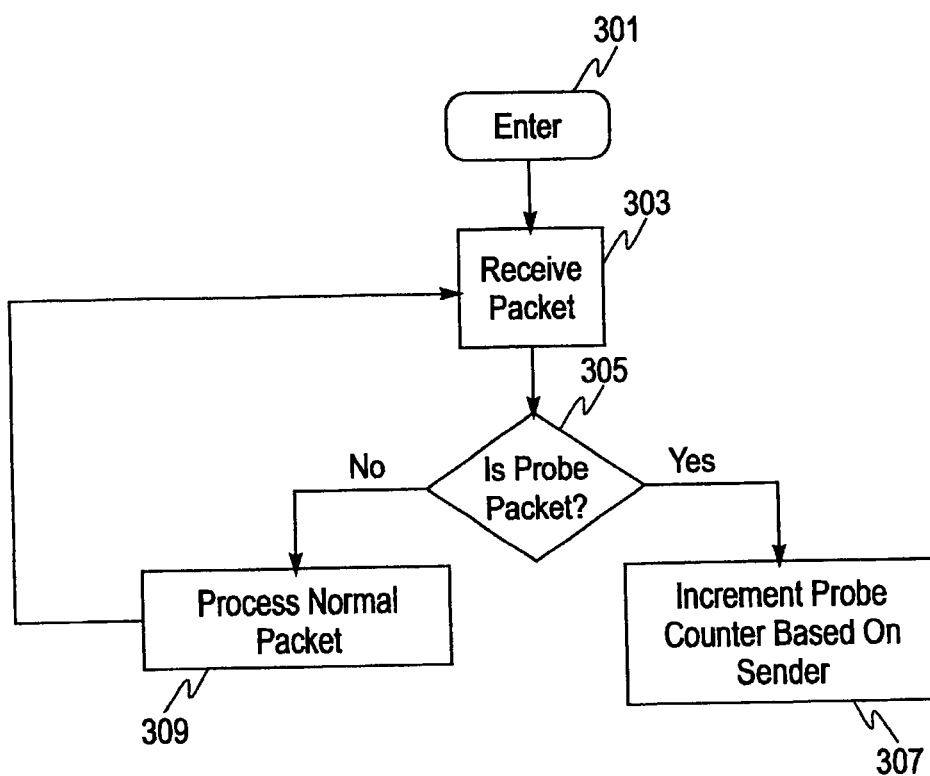


Fig. 3

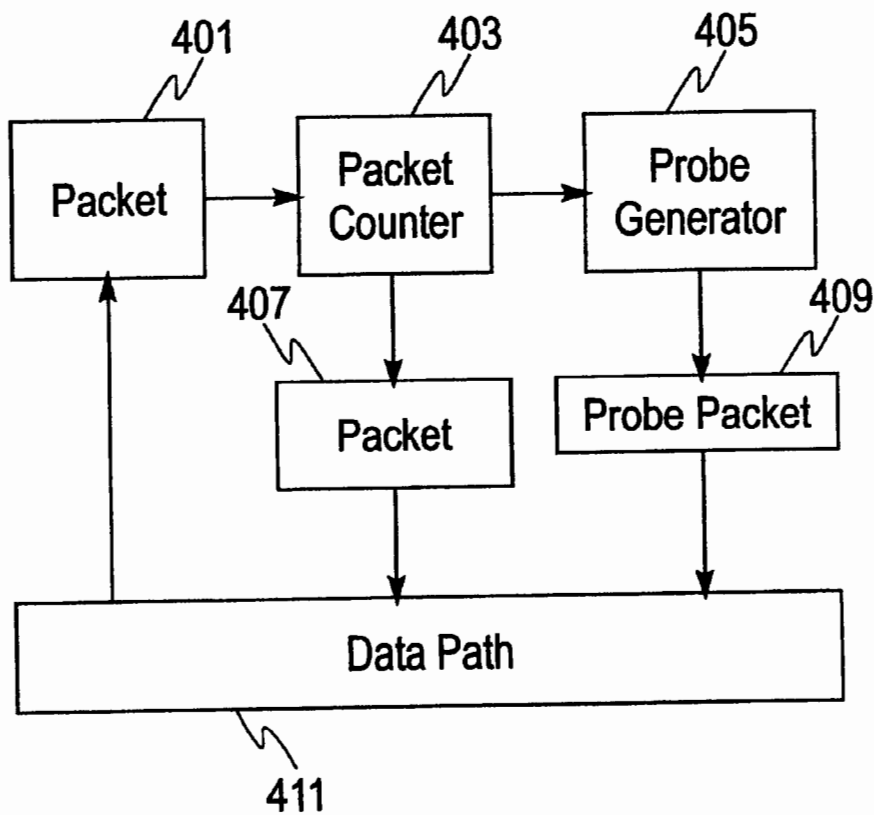


Fig. 4

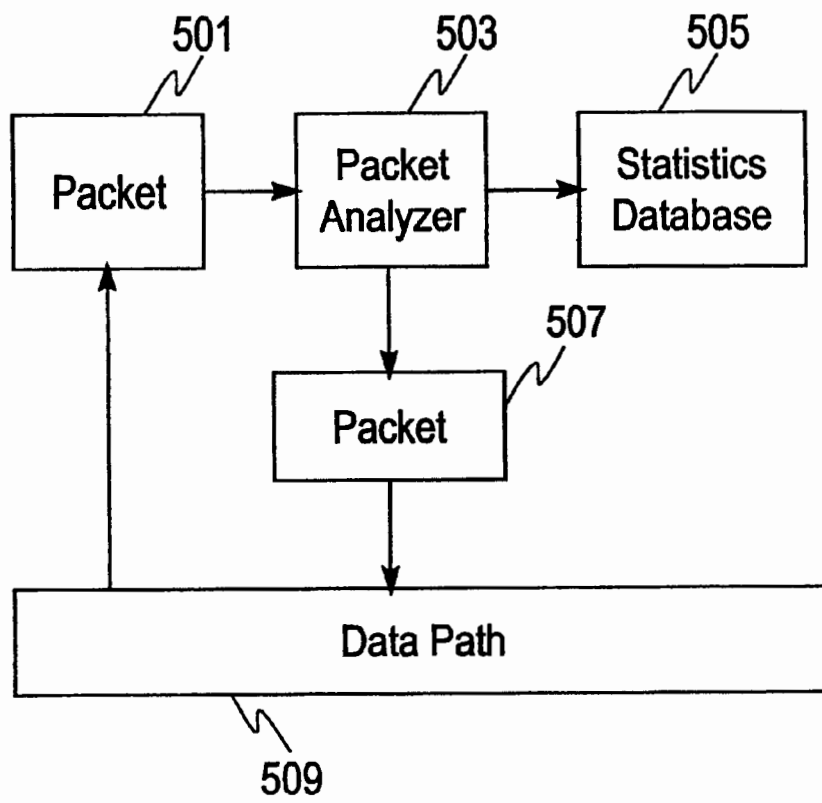


Fig. 5

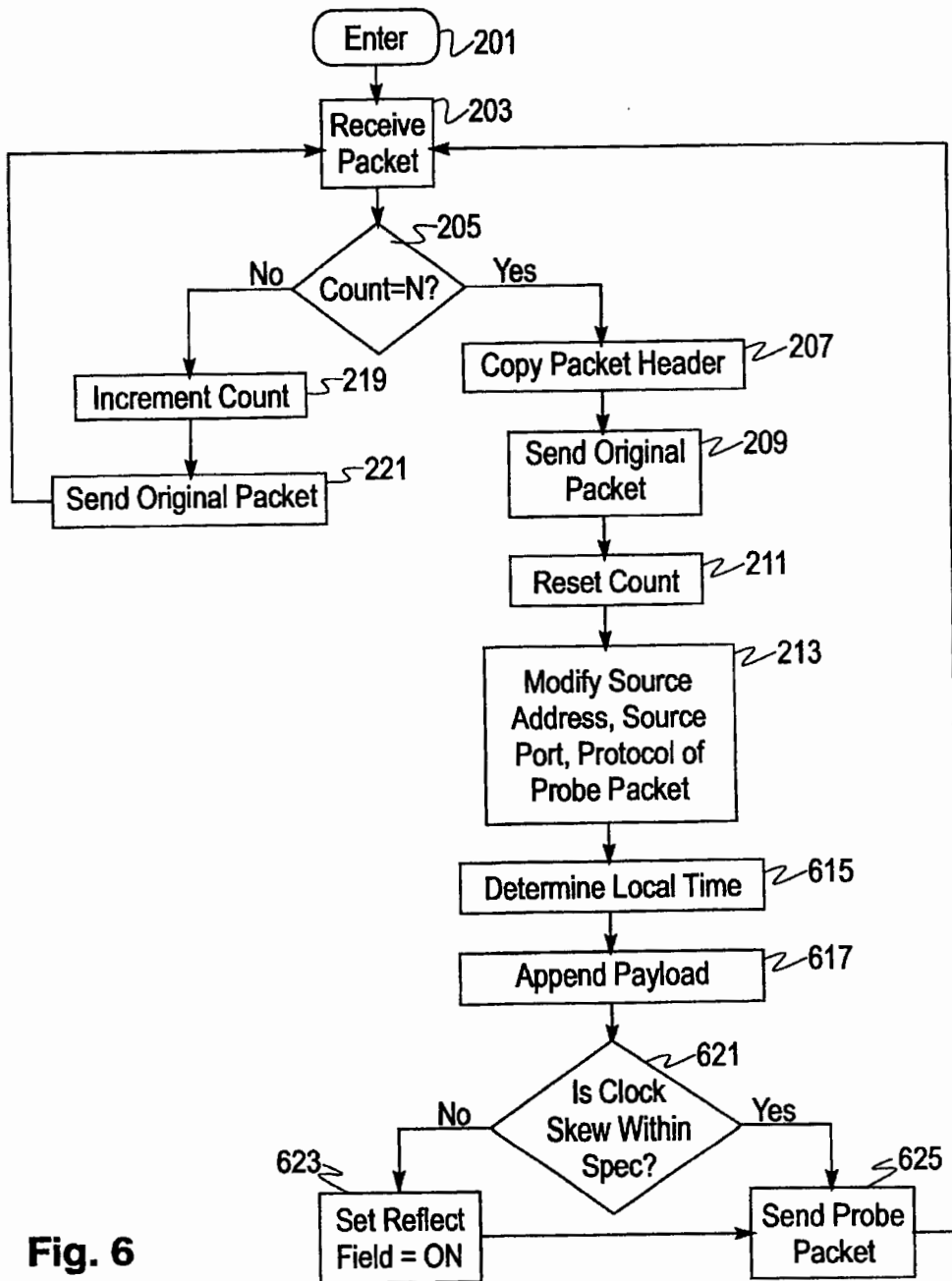


Fig. 6

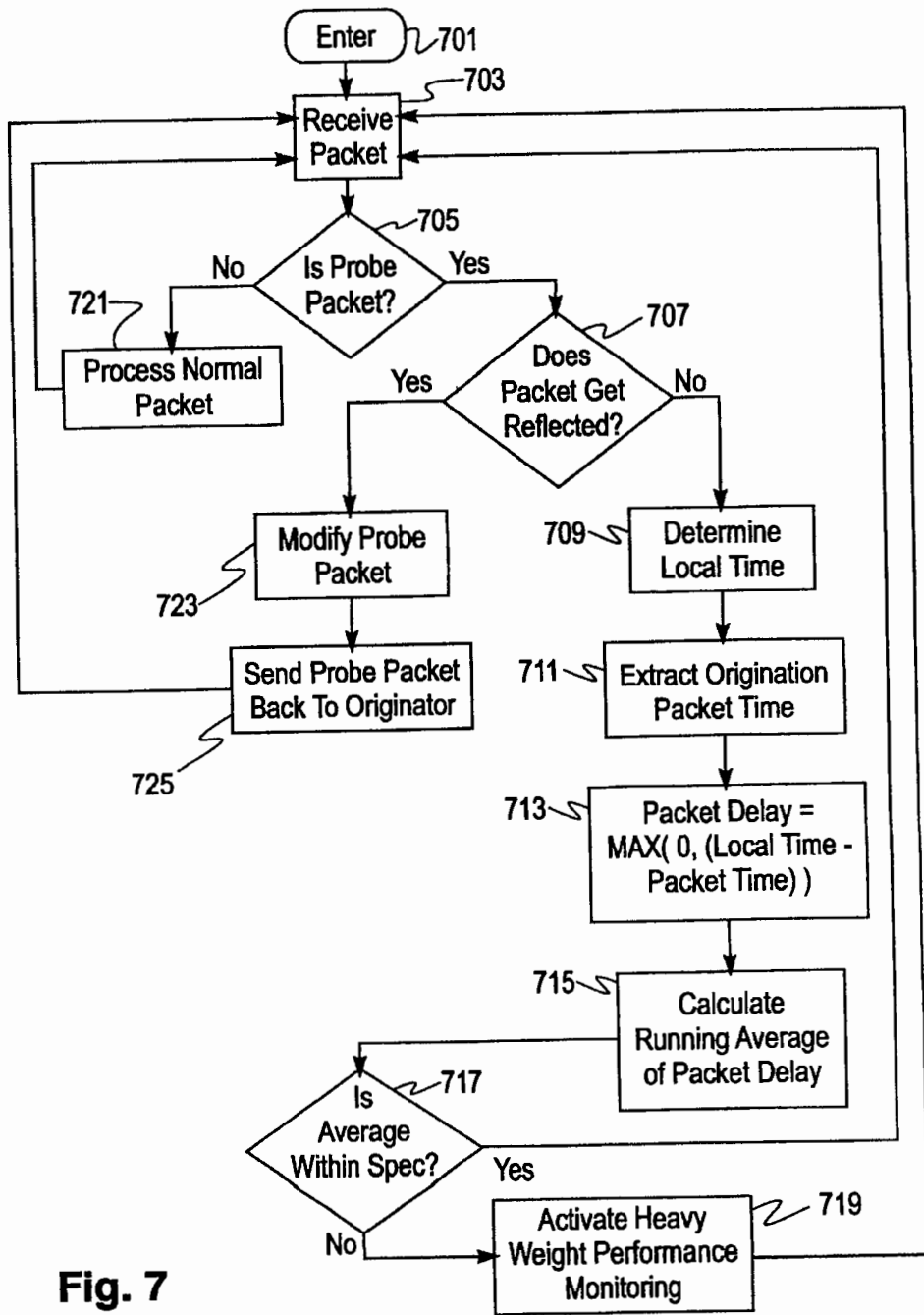


Fig. 7

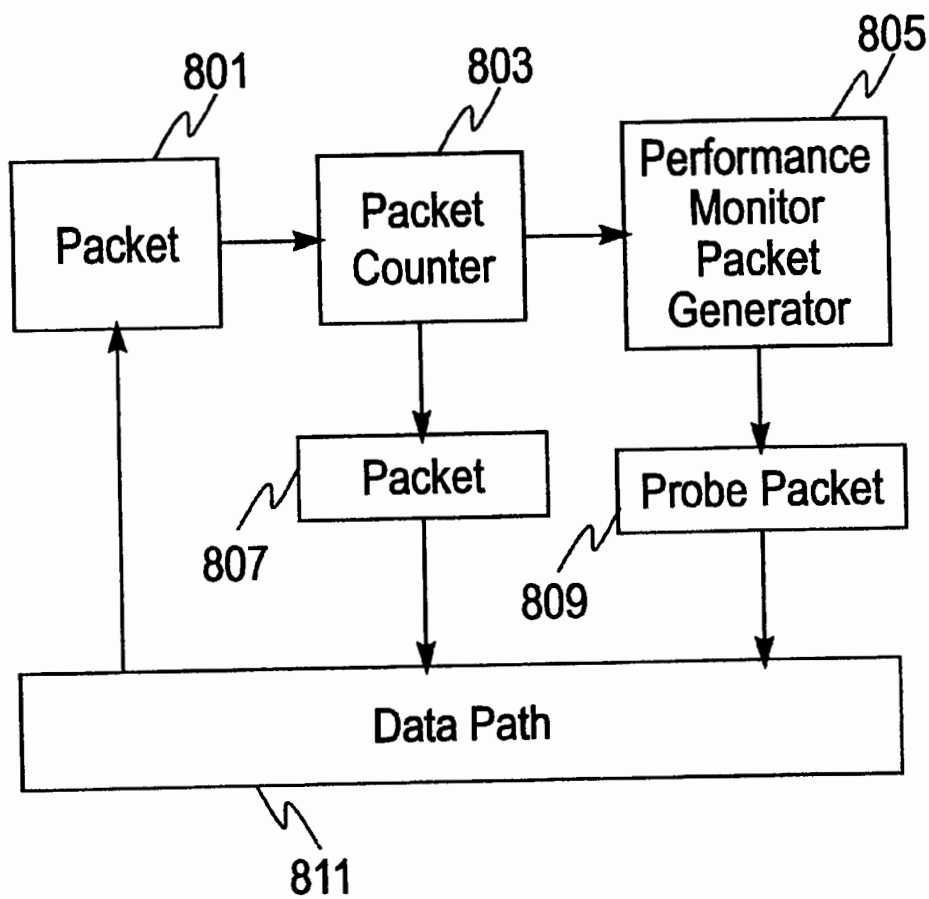


Fig. 8

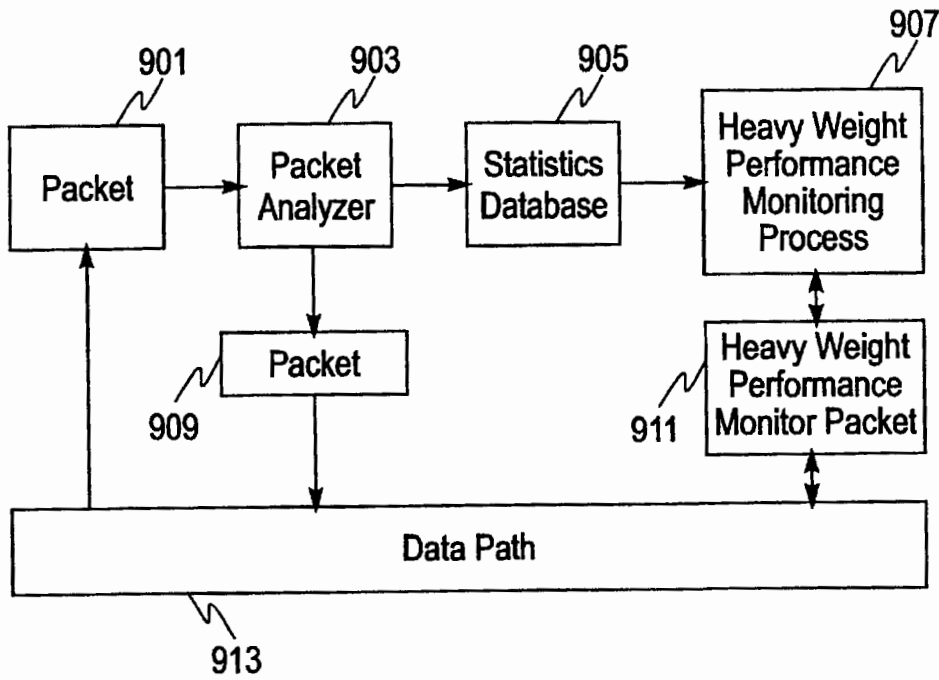


Fig. 9

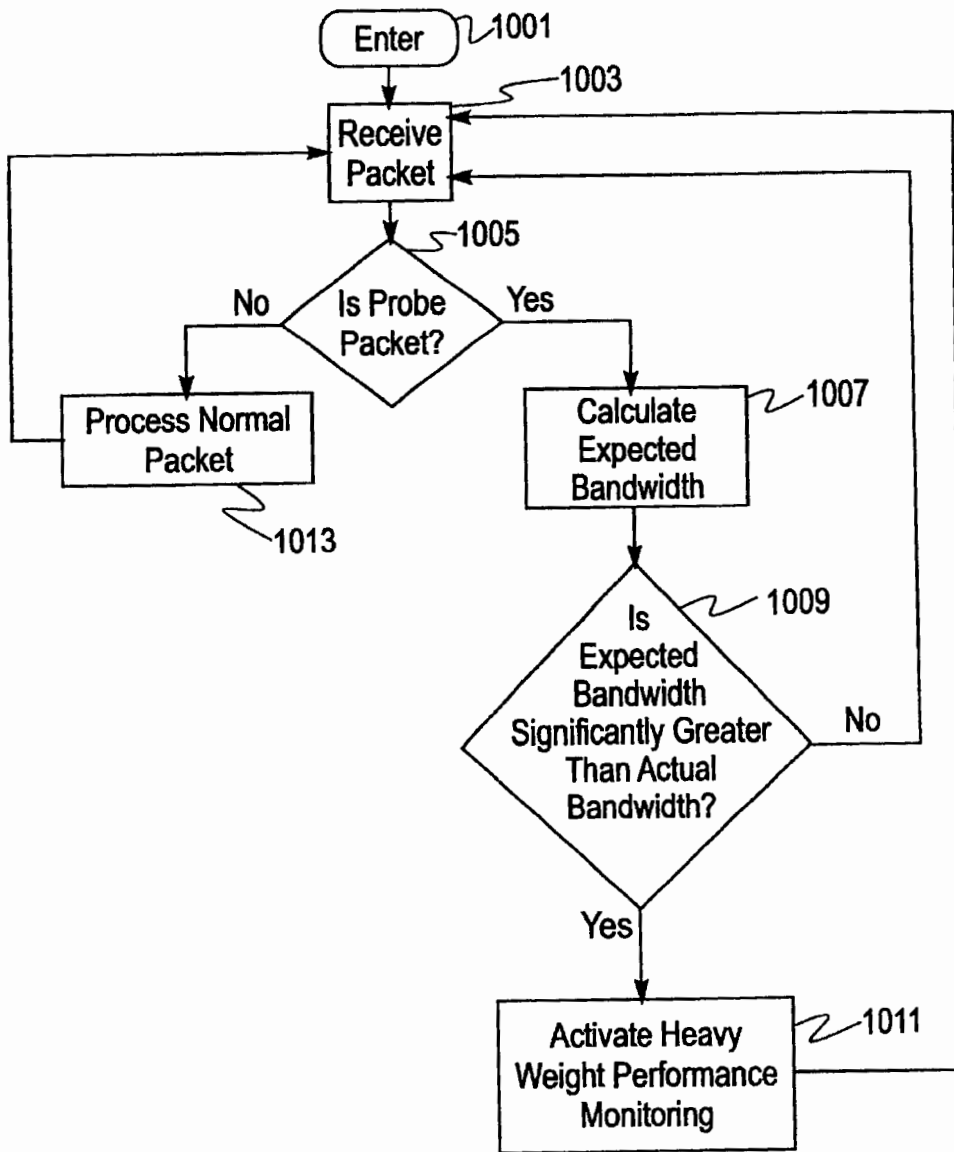


Fig. 10

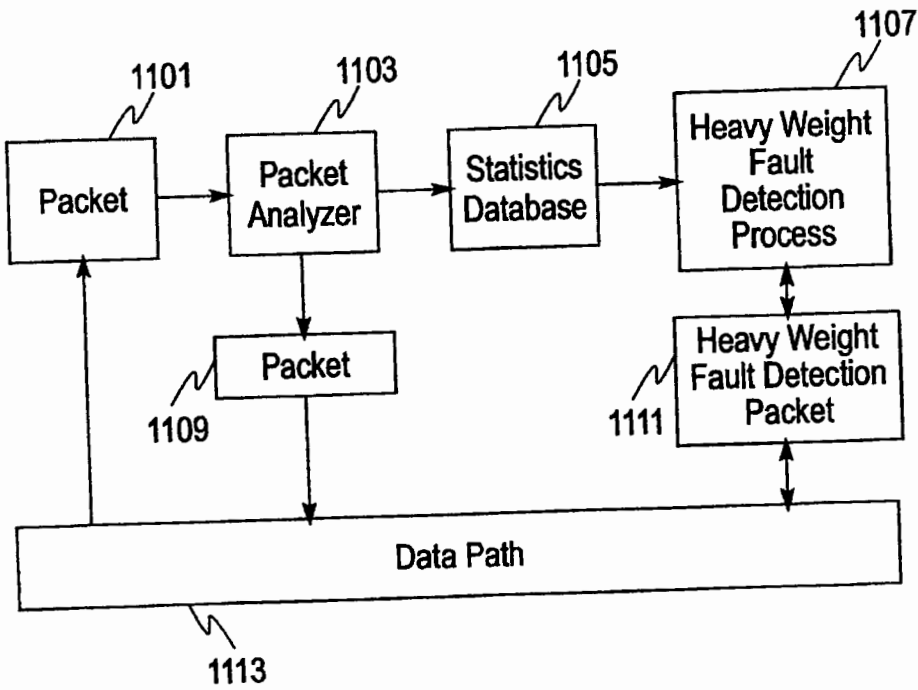


Fig. 11

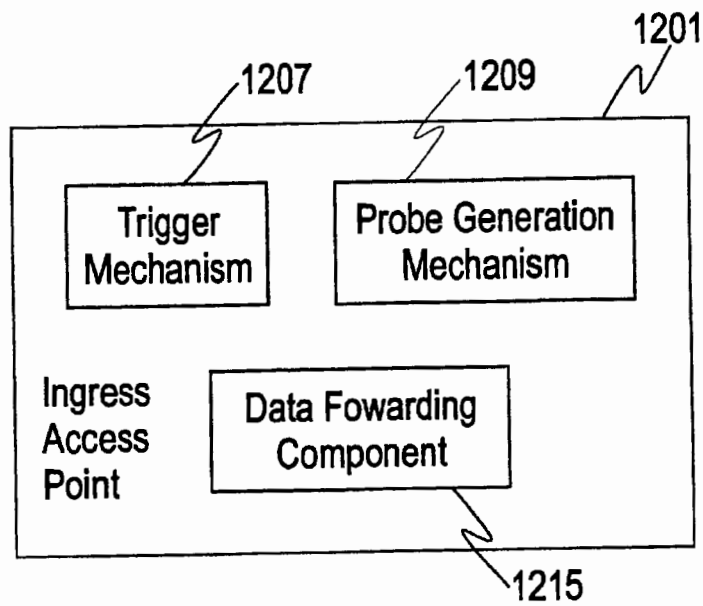


Fig. 12a

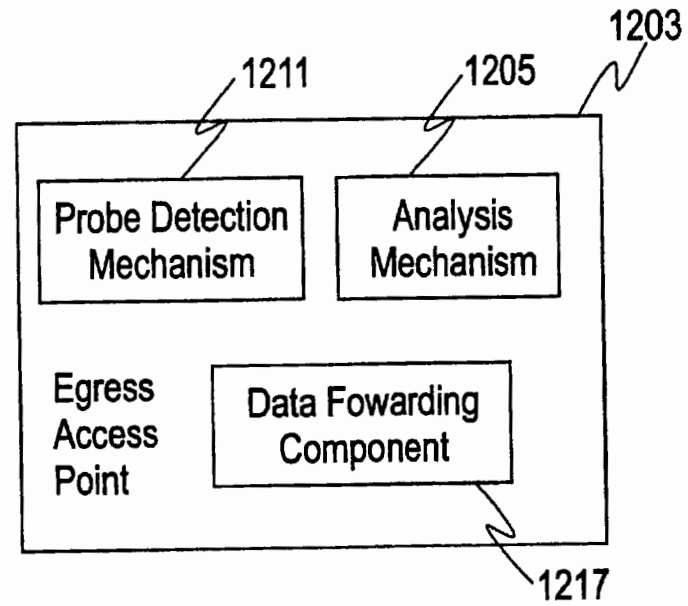


Fig. 12b

LOW OVERHEAD CONTINUOUS MONITORING OF NETWORK PERFORMANCE

FIELD OF THE INVENTION

The present invention is directed to the field of computer networks. It is more specifically directed to monitoring network performance metrics.

BACKGROUND OF THE INVENTION

There is a growing demand for communication via networks. This is especially so, for networks that are based on the TCP/IP protocol. This is both within the context of a corporate intranet as well as the context of an ISP providing Internet services to a group of users. A typical network provider may connect different campus networks belonging to a customer via its backbone network. As a provider of network services, the provider is interested in monitoring and ensuring that the network performs according to the guidelines and limits specified in the service level agreement between itself and the customer. The need for monitoring may be felt by the network operator, the customer or both.

The network operator is typically interested in monitoring the performance of the network, which is in its domain of control. Thus, he would like to measure the performance between all the access points that a customer uses to connect to his network. One way to measure the performance would be to execute a traditional performance measurement tool on a pair of access points.

Current performance monitoring techniques involve the use of 'ping', 'traceroute', 'netperf', data accessed through SNMP, etc. These methods are disadvantageous in that they add extra traffic load in the network between all of the access points. These methods also add additional packet processing on the network routers. These traditional network performance measurement mechanisms tend to be relatively heavy weight and generally rely on sending a continuous sequence of ping packets or other network probes between an end-point to all the other end-points. This process creates a large increased load on the network. The traditional normally high overhead network performance measurement mechanisms are herein referred to as 'heavy weight' mechanisms.

A typical ISP or a corporate intranet backbone has a very large number of network end-points. It is not typical to have a continuous monitoring of network performance between all of the end-points. The problem is compounded by the fact that the access points that may be controlled by the network operator are usually intermediate points in the network, and do not constitute either the source or destination of the traffic.

Consider a network operator with N access points provided to a customer. These access points are intermediaries along the path that a packet takes from the customer network. Performance monitoring of the network operation requires monitoring the $N \times (N-1)$ simplex channels between the pair-wise access points, and determining performance metrics such as delay between these access points. Another useful metric is the determination of the bandwidth that is being used between the different access points. The bandwidth usage is often a component in the price charged to the customer by the operator.

The measurement of bandwidth between the access points is difficult to do for an ISP. An IP packet only contains the final source and destination addresses, and standard routing information only provides the next outgoing interface from

an ingress access router. Thus, determining the egress access router of a packet requires building a duplicate overlay routing infrastructure within the ingress access router. This introduces additional processing delays in the forwarding of an IP packet, and also requires additional space for storage of overlay routing information.

A different issue arises when performance metrics (e.g. delays) are being monitored between the access points. An ISP would like to proactively monitor the delays between two access points belonging to a customer to verify if the delays exceed the desired bounds. Using heavy weight probes to do this monitoring across all access points is not a viable solution due to the additional load generated on the network. However, a continuous monitoring of network performance is desirable to determine the level of service provided and/or to determine if there are any problems between two network access points.

SUMMARY OF THE INVENTION

It is therefore an aspect of the present invention to provide a low overhead method for monitoring bandwidth in a corporate intranet or an ISP controlled portion of the Internet.

It is also an aspect of the invention to provide a low overhead method for performance monitoring in a corporate intranet or an ISP controlled portion of the Internet.

It is a further aspect of the invention to provide a low overhead method for fault monitoring in a corporate intranet or an ISP controlled portion of the Internet.

It is a further aspect of the invention to provide apparatus, a computer product and an article of manufacture comprising a computer usable medium having computer readable program code means embodied therein for causing a computer to perform a low overhead method for a network which includes originating and receiving access routers. The method is such that bandwidth monitoring, performance monitoring and fault monitoring are all possible. In an example implementation of the invention, network probe packets are generated by copying normal outbound packets and modifying these packets such that the destination address remains in tact. Recipient nodes can identify probe packets by their protocol number and UDP header pattern. Probe packets that are sent to nodes that are not aware of the probe packet format, discard these packets through normal network mechanisms.

Still a further aspect of the present invention is an apparatus to monitor network performance characteristic between a first and a second access point in the network. The apparatus including: a trigger mechanism at the first access point for determining a time to generate a probe packet; a probe generation mechanism at the first access point for generating a probe packet based on contents of a data packet and probe generating criteria; a probe detection mechanism at the second access point for detecting the probe packet; and an analysis mechanism to compare the probe packet detected at the second access point to the probe packet generated at the first access point.

In an embodiment of the apparatus the trigger mechanism and probe generation mechanism are located in a path of forwarding packets and/or wherein the probe detection mechanism is located in the path of forwarding packets, and/or the probe detection mechanism performs the additional function of removing probe packets from a data stream, and/or the trigger mechanism and probe generation mechanism are located outside of the data forwarding path of the packets, and monitor the flow of packets through the

3

first access point, and/or the probe detection mechanism is located outside of the data forwarding path of packets, and operates by monitoring the flow of packets through the second access point.

Still a further aspect of the present invention is a method for providing bandwidth measurement between an ingress and an egress access router in a network. The method including the steps of: counting a plurality of packets at the ingress access-router; generating a probe packet whenever a packet count reaches a specified value N; and counting the probe packets received at the egress access-router.

In a particular embodiment of the method the step of discarding the probe packets through normal network mechanisms if an egress edge device is not found at a specified destination address of the probe packet, and/or the step of generating includes probe packets copying every Nth packet and changing the source address of the Nth packet to that of an originating access router; changing the protocol number to a reserved protocol number, attaching a UDP header with a reserved pattern, and attaching a data segment as the UDP payload which includes a probe number and a local time-stamp, and/or the step of the egress access-router identifying probe packets by the reserved protocol number and the UDP header pattern, and/or a delay between two endpoints can be determined through use of a common clock and probe packets, and/or the step of counting a first number of probe packets received, and comparing the first number to a second number of packets that should have been received.

Still a further aspect of the present invention is a method for providing bandwidth accounting between a first and a second ISP access point in a network. The method including: configuring at least one ingress access point to have a first packet count of 'N-in'; the at least one ingress access point keeping track of a second packet count 'N-out' of packets sent into the network; and generating a probe packet whenever 'N-out'='N-in', wherein the probe packets being given a destination address of an Nth packet sent into the network, and being given a source address of an ingress router associated with the at least one ingress point.

Still a further aspect of the present invention is a method for measuring network characteristics between a first and a second router in a network. The method including the steps of: configuring at least one ingress access point on the first router to generate a plurality of probe packets; generating each of the probe packets based on the contents of a next data packet passing through the ingress access point; configuring at least one egress access point on the second router to detect the probe packet; and correlating each of the probe packets received at the egress access point with each of the probe packets sent by the ingress access point to determine the network characteristics between the ingress and egress access points. In some cases the probe packets are generated after a preset number of data packets have passed through the ingress access point.

Still a further aspect of the present invention is to provide an article of manufacture comprising a computer usable medium having computer readable program code means embodied therein for determining network characteristics between a first and a second access point in a network. The computer readable program code means in the article of manufacture comprising computer readable program code means for causing a computer to effect the steps of configuring the first access point as an ingress access point to generate a plurality of probe packets; generating each of the probe packets based on contents of a data packet and preset criteria; configuring the second access point as an egress

4

access point to detect the probe packets; and correlating each of the probe packets received at the egress access point with one of the probe packets sent by the ingress access point to determine the network characteristics between the two access points. In some cases the network characteristics include a round-trip delay between two endpoints and the step of generating includes marking the probe packet with a time of generation, and the computer readable program code means in the article of manufacture further comprising computer readable program code means for causing a computer to effect the steps of the egress access point reflecting a probe packet back to the ingress access point; and the ingress access point comparing the time of generation to a time when the probe was received back.

Still a further aspect of the present invention is to provide a computer program product comprising a computer usable medium having computer readable program code means embodied therein for forming a plurality of probes packets in an network. The computer readable program code means in the computer program product comprising computer readable program code means for causing a computer to effect: marking a protocol field in an header of each probe packet with a reserved protocol number; filling a source port field in a UDP header for each probe packet with the reserved pattern; and filling a destination probe field with the probe number.

In a particular embodiment of the computer program product, the computer readable program code means in the computer program product further comprising the step of using the reserved protocol number at an egress access router to identify each probe packet, and/or the computer readable program code means in the computer program product further comprising the step of extracting each probe packet based on an identification obtained in the step of using.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other aspects, features, and advantages of the present invention will become apparent upon further consideration of the following detailed description of the invention when read in conjunction with the drawing figures, in which:

FIG. 1 shows an example block diagram of two Internet service providers (ISP) and their respective customer networks in which each ISP network has one access router connecting the network to the Internet backbone;

FIG. 2 shows an example flow diagram that illustrates steps taken when sending a probe packet by an access router for determining bandwidth statistics in accordance with the present invention;

FIG. 3 shows an example flow diagram that illustrates steps taken when receiving a probe packet by an access router for bandwidth monitoring in accordance with the present invention;

FIG. 4 shows an example block diagram that illustrates the components and the relationships between them for a process of sending probe packets for bandwidth statistics in accordance with the present invention;

FIG. 5 shows an example block diagram that illustrates the components and the relationships between them for the purpose of receiving probe packets for statistical bandwidth data gathering in accordance with the present invention;

FIG. 6 shows an example flow diagram that illustrates the steps taken when sending a probe packet for delay and performance monitoring in accordance with the present invention;

FIG. 7 shows an example flow diagram that illustrates the steps taken when receiving a probe packet for delay and performance monitoring in accordance with the present invention;

FIG. 8 shows an example block diagram that illustrates the components and the relationships between them for the process of sending probe packets for delay and performance monitoring in accordance with the present invention;

FIG. 9 shows an example block diagram that illustrates the components and the relationships between them for the process of receiving probe packets for delay and performance monitoring in accordance with the present invention;

FIG. 10 shows an example flow diagram that illustrates the steps taken when receiving a probe packet for fault monitoring in accordance with the present invention;

FIG. 11 shows an example block diagram that illustrates the components for receiving a fault monitoring probe packet in accordance with the present invention; and

FIG. 12 shows an example of the components used for a system to perform the low overhead continuous monitoring in accordance with the present invention.

DETAILED DESCRIPTION OF THE INVENTION

The network context within which this invention applies is shown in FIG. 1. It shows two ISP customer networks 101 and 113, that are connected to the Internet 109. Connectivity to the Internet 109 is obtained via the access router-1 107 and access router-2 111. Both access routers 107 and 111 serve as gateways to the ISP customer networks 101 113, and interface these networks with the Internet 109. When network traffic flows from the clients 103 and 105 on ISP customer network-1 101 to the clients 115 and 117 on ISP customer network-2 113, the traffic goes through these access routers making access router-1 107 an ingress node and access router-2 111 an egress node. An access router can alternately serve as both an ingress node and an egress node depending on the direction of the flow of the traffic. FIG. 1 also shows clients on the two ISP customer networks 107 and 111, which communicate with each other. Clients 103 and 105 are connected to ISP customer network-1 101, and may communicate with clients 115 and 117 that are connected to the ISP customer network-2 113. A similar context arises in a corporate intranet, wherein the Internet 109 is replaced by the corporate network instead of the Internet. Other applications would operate in a similar fashion. This includes, for instance, implementing this invention where one of the access points is at an entity firewall.

Within this context, one needs to monitor the bandwidth, delay and loss characteristics of a network between two access points. This needs a bandwidth monitoring protocol which determines the amount of bandwidth used between a pair of access points. For example, a bandwidth monitoring protocol may work as follows. Each ingress access point is configured with a specific count number (say N). This count can be a count of packets sent, bytes sent or time elapsed. The count is incremented based on preset criteria. The preset criteria is usually the occurrence of a type of event. The event may be a number of packets being sent, a number of bytes being sent, time elapse, and/or a dynamically changing event. The ingress access point keeps track of the count as it sends packets into the network. After every N events, it generates a new probe packet into the network. The probe packet is given the same destination (e.g. IP) address as the Nth packet into the network, but it is given a source (e.g. IP) address that is of the ingress access router. The probe packet

contains special data that identifies it as a probe packet in the network. Additional data in the probe packet can be used for network monitoring.

At the egress router, the probe packets are identified and removed from the network data stream. They are sent to a local process, which analyzes the information contained in the probe packet. For bandwidth accounting, the receiving process needs to simply increment the count of packets received from the ingress access router. At periodic intervals, the accumulated number of packets received from each ingress access routers is converted into an estimate of the total number of packets being sent from that access router.

Examples of the detailed steps of a bandwidth monitoring protocol are shown in FIGS. 2 and 3. Referring to FIGS. 1 and 2, the steps shown in FIG. 2 are executed by an ingress access router whenever it obtains a packet from its associated ISP network. For example, it will be executed by software running on access router-1 107 whenever that router receives a packet from the clients on ISP customer network-1 101 as shown in FIG. 1.

FIG. 2 shows an example flow diagram for the creation and transmission of probe packets for bandwidth monitoring. When a packet is received from a client by an ingress access router 203, the event count is checked 205. The event count can be based on the number of packets sent, the number of bytes sent, or the duration of time elapsed. If the event count is equal to N, where N is some preconfigured value, the header of the current packet is copied 207 and the original packet is processed and sent 209 through normal network operations. The count is reset to zero 211. A probe packet is then created 213 with the payload appended 215 to the copied packet header. The probe packet has the same packet header as the copied packet except that the source address is changed to the address of the ingress access router. In addition, the source port is changed to a defined number and the protocol number is changed to a defined number that is not already used by other protocols. The time of the creation of the probe packet is appended in the payload of the probe packet along with some other necessary data. The probe packet is then sent out 217 into the Internet (numbered as 109 in FIG. 1) to be reached by the final destination. On the other hand, if the event count is not equal to N, the counter is incremented by one 219 and the original packet is processed through normal network operations 221.

An example of reception of the probe packets by the egress access routers can be seen in the flow diagram shown in FIG. 3. When a packet is received 303 from the Internet 109 by an egress access router, the protocol field, and/or the UDP source and destination ports (which are formed to contain a special pattern described subsequently) are checked to determine if the packet is a probe packet 305. If it is not a probe packet, the normal processing of the packet is performed 309. If it is indeed a probe packet, statistics are collected 307. This includes data such as incrementing a count of the probe packets received from a certain ingress access point in order to calculate the bandwidth between the two access points.

FIG. 4 shows an example of system components for sending probe packets. It includes the packets 401, a packet counter 403, a probe generator 405, a standard packet 407, and a probe packet 409 going to the data path 411.

FIG. 5 shows an example of system components for receiving packets. It includes the packet as received 501, a packet analyzer 503, a statistics database 505, and the processed packet 507 feeding the data path 509.

While there are multiple ways to create probe packets, we present a scheme, which can be implemented relatively easily on most platforms. In this scheme, the probe packets are created with an IP header and a UDP header. However, the protocol field in the IP header is not marked with the UDP protocol ID, but rather it is marked with a special reserved protocol number. The source port field in the UDP header is filled with a reserved special pattern, and the destination port field is filled with a number that is unlikely to be used at any receiving end-host. This will cause this packet to be discarded through normal IP mechanisms at its destination if an egress edge device is not encountered by the probe packet. This reserved protocol number is used by the egress access router to identify probe packets in order to extract them from the stream. It then matches the reserved special pattern in the source and destination port fields to double-check the probe packet. It replaces the protocol field with the UDP protocol number and the destination port is replaced with a local port number where a monitoring process is active. This allows the egress access router to handle the packet as a normal UDP packet.

A performance monitoring protocol is used in order to monitor the performance of the network. In an example of a performance monitoring protocol the UDP payload of the probe packet is used to carry information about network delays. The source access router includes the time-stamp of the creation in the probe packet. It sometimes also contains an optional indication for the destination access router to reflect the packet back to the source. Example flow diagrams of the steps to implement a typical protocol are given following the description of the protocol.

In this performance monitoring protocol, when the packet is received at the egress access router, the local time-stamp of packet reception is computed. The difference in time between the packet reception and transmission is the delay. This difference is rounded to zero if it happens to be negative. The time difference is smoothed out using a running average, and compared to a target delay stored in a configuration file, or in a directory database. If the smoothed delay exceeds a target delay, or in some cases when it approaches it, a trigger is generated for the ingress and egress access routers to activate the heavy weight performance monitoring protocol available on them. This scheme works well when the target delay amounts exceed the amount of clock skew that is expected among the two routers.

In cases wherein the clock skew (a common time difference between access routers) is an issue, the ingress access router contains an option that probe packets be reflected back to it. The ingress access router then has a smoothed round-trip delay. It activates the heavy weight performance monitoring when the round-trip delay exceeds the target round-trip delays.

Note that the continuous monitoring provides a low overhead trigger that is activated when network performance problems are suspected. A more precise value of network performance is obtained by the heavy weight monitoring protocol, and an appropriate alarm generated for notifying a network operator.

For the following description refer back to FIG. 1, wherein access router-1 107 represents an ingress access router for ISP customer network-1 101, and access router-2 111 represents an egress access router for ISP customer network-2 113. Client devices 103 and 105 are part of ISP customer network-1 101 and client devices 113 and 117 are part of ISP customer network-2 113. The description

assumes that data travels from ISP customer network-1 101 to ISP customer network-2 113. In this case, the access router-1 107, receives data from client devices 103 and 105 in the form of (e.g. IP) packets. The combined flows of clients 103 and 105 to access router-1 107 have a rate 'R' which is the rate measured at any given point in time. Rate 'R' may or may not be continuous. Access router-1 107 includes an event counter 'E'. This counter is incremented based on a configurable preset criteria. The preset criteria is usually the occurrence of a type of event to increment event counter 'E'. The event may be a number of packets being sent, a number of bytes being sent, time elapse, and/or a dynamically changing event. The criteria is used as a setting which indicates that counter 'E' should be incremented. In a simple embodiment, it is based on the number of packets or bytes which it receives from clients 103 and 105. It is sometimes based on an amount of time elapsed. Access router-1 107 increments this counter. When the counter reaches a predetermined value N, access router-1 107 copies the current packet received (or the next packet if the event is time driven) and then forwards the original packet as normal.

The (e.g. IP) packet that has been copied is altered and converted into a probe packet in the following manner. The source address is replaced with the source address of access router-1 107. The protocol number field is changed to a defined value not used by other protocols. The payload section of the copied packet is discarded and is replaced by a UDP header and data section. The destination and source port numbers are set to a well-known port number. The probe packet is formed to also contain additional data to be used for network monitoring. The additional data may include things such as a probe identifier, local time-stamp and reflect field. Access router-1 107 then forwards this packet onto the Internet 109.

The probe packet travels over the Internet 109 to be received by access router-2 111. Access router-2 111 recognizes this packet as a probe packet by its protocol number and/or its source port field within the packet header. Access router-2 111 replaces the protocol field with the UDP protocol number and the destination port with a local port number. The local port is where some process local to access router-2 111 receives this probe packet in the form of a UDP packet and processes it.

Access router-2 111 processes the probe packet by extracting the time-stamp included by access router-1 107 in the data section of the probe packet. The 'reflect' packet field is checked to determine if the probe packet should be reflected back to the source access router-1 107. When access router-2 111, receives the probe packet, a local time-stamp is compared to the time-stamp included in the packet. The difference between the receive time-stamp and the sent time-stamp is the delay of transmission from access router-1 107 to access router-2 111. This transmission delay is used in a running average, which is compared to a target transmission delay. If the average delay exceeds or approaches the target delay, access router-2 111 will start a heavy weight performance monitoring protocol between access router-1 107 and access router-2 111. The heavy weight performance monitoring protocol may also be started by access router-1 107 if probe packets are being reflected back to access router-1 107 from access router-2 111 and the round trip delay exceeds a target round trip delay for access router-1 107. The results of the heavy weight monitoring mechanism are sent to the network operator.

As each probe packet is being received by access router-2 111, access router-2 111 keeps track of how many probe

packets it has received from each source access router. Using the number of probe packets received, access router-2 111 can estimate how many packets it expects to receive from the source, access router-1 107. The estimated bandwidth can be sent to the network operator.

FIG. 6 shows an example of a flow diagram for the transmission of the probe packets for delay and performance monitoring. In this case, before the probe packet is sent out, the clock skew is checked 621 whether it is within the specified value. If it is, the probe packet is sent out 625. If it is not, the 'reflect' field of the probe packet is set to 'ON' 623, before the probe packet is sent out 625.

FIG. 7 shows an example of a flow diagram for the reception of the probe packets for the delay and performance monitoring. When a packet is received on an egress access router 703, it is checked to see if it is a probe packet 705. If it is not, the normal processing of the packet is performed 721. If the packet is a probe packet, the reflect field is checked to determine if the packet gets reflected 707. If the reflect field is set, then the probe packet is modified 723 and sent back to the originator 725. If the field is not set the local time is determined 709 and the origination packet time is extracted 711. The packet delay is calculated by subtracting the origination transmission time of the packet from the current local time and taking the maximum value of the difference and zero 713, thereby rounding to zero if the difference is negative. Then, a running average delay is calculated 715. If this average is within the delay specified by the network operator, then no action is taken. If the average delay is not within the specified delay, a heavy weight performance monitoring process is activated 719.

The system components for the sending and receiving of the probe packets for the delay and performance monitoring are shown in FIGS. 8 and 9 respectively. FIG. 8 shows the packet as sent 801, going to a packet counter 803, and then to a performance monitor packet generator 805. The probe packet 809 and original packet 807 are put on the data path 811.

FIG. 9 shows the packets that are received 901, and analyzed 903. Normal (non-probe) packets pass through the packet analyzer 903, forming packet 909, which is fed onto the data path 913. The results of the probe packets analysis go to a statistics database 905. When the statistics indicate that a heavy weight process is needed, the heavy weight performance monitoring process 907 is initiated. The heavy weight performance monitoring process 907 forms a set of performance monitor packets 911. The performance monitor packets 911 are fed onto the data path 913. The format and quantity of the performance monitor packets 911 are dependent upon the protocol used within the heavy weight performance monitoring process in ways known to those skilled in the art. These ways are outside the scope of this invention.

The probe packet also serves the purpose of fault monitoring. It is noted that access router-2 111 has a smoothed average of the number of packets it has received from access router-1 107. When the smoothed expected number is significantly more than the actual number of packets received, a heavy weight fault detection protocol is activated. This may be, for example, by running ping between access router-1 107 and access router-2 111. The results of the heavy weight fault detection mechanism are sent to the network operator.

FIG. 10 shows an example of a flow diagram for fault monitoring when receiving the probe packets on the egress access routers. When a packet is received 1003, it is checked to see if it is a probe packet 1005. If the received packet is

not a probe packet, then the normal processing of the packet is performed 1013. On the other hand, if the received packet is a probe packet, the expected bandwidth is calculated 1007. If the expected bandwidth is greater than the actual bandwidth, a heavy weight fault detection process is performed 1011. If it is not greater than the actual bandwidth, no action is taken and the flow is returned to receive another packet 1003.

The system components for the reception of the probe packets for fault monitoring are shown in FIG. 11. Normal packets pass through the packet analyzer 1103. The analyzer sends normal (non-probe) packets 1109 back onto the data path 1113. Probe packets are analyzed by the packet analyzer which then forwards the probe packet information to the statistics database 1105. When the database results indicate that a heavy weight fault detection process is needed, the heavy weight fault detection process 1107 is initiated. The heavy weight fault detection process 1107 forms a set of fault detection packets 1111 which are fed onto the data path 1113. The format and quantity of the fault detection packets 1111 are dependent upon the protocol used within the heavy weight fault detection process, in ways known to those skilled in the art. These ways are outside the scope of this invention.

The system components for the reception of the probe packets for fault monitoring are shown in FIG. 11. It shows the received packets 1101, feeding a packet analyzer 1103. The analyzer sends the packet 1109 for forwarding on the data path 1113 and also to the statistics database 1105. When the database results indicate that a heavy weight fault detection process is needed, the heavy weight fault detection process 1107 is initiated. The heavy weight fault detection process 1107 forms a performance monitor packet 1111. The performance monitor packet 1111 is fed onto the data path 1113.

FIG. 12 shows an example of the components used for a system to perform the low overhead continuous monitoring described in this application. It shows a way in which an apparatus required for the low overhead continuous monitoring of network performance can be constructed. The apparatus consists of components at each of the two access points of the network. The apparatus 1201 is at the ingress access point and the apparatus 1203 is at the egress access point of the network. The analysis component 1205 is used to analyze probe packets and to determine the final bandwidth performance value. Although the analysis component 1205 is shown to be located at the egress apparatus 1203, it can be located at either the ingress or the egress access point, or at another machine in the network.

At the ingress access point, the apparatus 1201 includes a trigger mechanism 1207 and a probe generation mechanism 1209. These components are in addition to the data forwarding component 1215 which is present in any access point. The trigger mechanism 1207 determines when to generate a probe packet. This decision could be made when specific time-intervals elapse, on a count of packets or bytes processed by the data forwarding component 1215. When the trigger mechanism determines that a probe is to be generated, it activates the probe generation component 1209 which creates a probe packet. The probe packet is created from the contents of the current or the next data packet being processed by the access point.

At the egress point, the apparatus 1203 includes a probe detection mechanism 1211, an analysis mechanism 1205, and the data forwarding component 1217 which is present in any access points. The probe detection mechanism 1211

11

executes sequential processing in step 134. Although only simple, sequential, and timer/counter based attributes have been discussed, other signature attributes can be incorporated into the present invention.

In step 136 the virtual processor 36 determines if the execution of the attack signature has revealed a network intrusion. If the data collector 10 recognizes a network intrusion, in step 138 the reaction module 38 is notified. If no attack has been detected, in step 140 the virtual processor 36 determines if the instruction cache 42 is empty. If the instruction cache is not empty, the virtual processor 36 returns to step 122 and accesses the next attack signature profile. If the instruction cache 42 is empty, the next packet in the queue 48 is obtained in step 141 to extract packet information into the register cache 40.

Referring to FIG. 10, a method for processing a sequential attack signature profile includes the step 142 of splitting the attack signature profile into expressions. As previously discussed, a sequential attack signature profile is composed of multiple component expressions which are sequentially evaluated to determine if each expression matches a data packet associated with a particular application session. In step 146 the virtual processor 36 determines whether a pointer is set to the sequential attack signature profile in the state cache 44. If the pointer is not set to the sequential attack signature profile, in step 148 an entry is made in the state cache 44 so that a pointer is set to the sequential attack signature profile and the entry parameters are initialized. In step 150, the virtual processor 36 references a state cache entry 44 to determine how many of the expressions have already been matched to data packets associated with the currently monitored application session.

In response to the state cache entry, the virtual processor 36 obtains the next sequential expression from an expression list in step 152. For example, an attack signature profile might include expressions A, B, and C. Expression instruction A was executed and found to match a first packet associated with an application session and expression instruction B was executed and found to match a second packet associated with the application session. Upon receiving a third packet associated with the application session and after referencing the state cache entry to obtain the information that expressions A and B have been matched, the virtual processor 36 obtains the third expression to determine if it matches the third packet. It should be noted that expressions A, B, and C need not be found to match three consecutive data packets associated with an application session. Rather, expression A must be found to match a packet which precedes a packet found to match expression B or C, and B must be found to match a data packet which precedes a packet found to match expression C.

In step 154, after executing an expression instruction, the virtual processor 36 determines whether the expression matches the data packet. If the expression does not match, the virtual processor 36 returns a false value in step 156. If the expression matches, a determination is made in step 158 whether the expression was the last sequential expression. In step 160, the virtual processor 36 updates the entry in the state cache 44 to reflect the match of the expression to the data packet if it is determined that the executed expression is not the last sequential expression and in step 162 the virtual processor returns a value of false. If the expression is the last sequential expression, in step 164 the virtual processor 36 returns a value of true to indicate that a network intrusion has been detected.

The processing of a simple attack signature profile is similar to the processing of a single expression of a sequen-

12

tial attack signature. Referring to FIG. 11, the attack signature profile is reduced to an expression in step 166. After executing the expression instruction, the virtual processor 36 determines whether the expression matches a data packet associated with an application session in step 168. If the expression matches the packet, in step 172 the virtual processor 36 returns a value of true and the reaction module 38 is notified of a network intrusion. If the expression does not match, the virtual processor 36 returns a value of false in step 170.

With reference to FIG. 12, a method for processing a timer/counter based attack signature profile includes the step 174 of reducing the profile to an expression. In step 176 the virtual processor 36 utilizes the timer 37 to make a current time stamp for the data packet being evaluated. Entries in the state cache 44 that are older than an attack interval are purged from the state cache 44 in step 178. Purging stale entries involves comparing a time interval between time stamps associated with entries and the current time. If the actual time interval associated with an entry is greater than the attack signature time interval, that entry is purged from the state cache 44.

In step 180 the expression is evaluated to determine in step 182 if the expression matches the packet currently being analyzed. If the expression does not match, the virtual processor 36 returns a value of false in step 184. If the expression matches the packet, the virtual processor returns a value of true and adds the current time stamp to the application session entry in the state cache 44 in step 186. In step 188 the counter 35 is utilized to update the number of events recognized by execution of the timer/counter expression instruction on data packets associated with the current application session. A determination is made in step 190 whether, after the number of event occurrences has been updated, the threshold number of events has been detected within the predetermined time interval. A value of false is returned in step 192 if the threshold has not been reached. If the threshold has been reached, in step 194, the virtual processor 36 returns a true value to indicate that a timer/counter based network intrusion has been detected.

What is claimed is:

1. A method for detecting network intrusion attempts associated with network objects on a communications network including the steps of:

storing a list of attack signature profiles descriptive of attack signatures associated with said network intrusion attempts;

storing corresponding data representative of a correspondence between subsets of said attack signature profiles and said network objects such that each network object has a corresponding stored subset of attack signature profiles and more than one subset of attack signature profiles corresponds to network objects;

monitoring network traffic transmitted over said communications network for data addressed to one of said network objects;

in response to detecting said data addressed to said network object, accessing a subset of attack signature profiles corresponding to said network object based on said correspondence data; and

executing at least one attack signature profile included in said subset corresponding to said network object to determine if said data addressed to said network object is associated with a network intrusion attempt.

2. The method of claim 1 wherein said executing step includes utilizing a processor to execute said at least one

13

15. A method as recited in claims 13, further comprising the step of determining network faults by comparing the expected number of probe packets at a particular egress access point to a number of probe packets actually received at the particular egress access point.

16. A method as recited in claim 1, wherein the network is an IP network.

17. A method as recited in claim 1, wherein the ingress point is located at a campus firewall.

18. A method as recited in claim 1, wherein the ingress and egress points are each located in separate routers.

19. An apparatus to monitor network performance characteristic between a first and a second access point in the network, said apparatus comprising:

a trigger mechanism at the first access point for determining a time to generate a probe packet;

a probe generation mechanism at the first access point for generating a probe packet based on contents of a data packet and probe generating criteria;

a probe detection mechanism at the second access point for detecting the probe packet; and

an analysis mechanism to compare the probe packet detected at the second access point to the probe packet generated at the first access point.

20. An apparatus as described in claim 19, wherein the trigger mechanism and probe generation mechanism are located in a path of forwarding packets.

21. An apparatus as described in claim 19, wherein the probe detection mechanism is located in the path of forwarding packets.

22. An apparatus as described in claim 21, wherein the probe detection mechanism performs the additional function of removing probe packets from a data stream.

23. An apparatus as described in claim 19, wherein the trigger mechanism and probe generation mechanism are located outside of the data forwarding path of the packets, and monitor the flow of packets through the first access point.

24. An apparatus as described in claim 19, wherein the probe detection mechanism is located outside of the data forwarding path of packets, and operates by monitoring the flow of packets through the second access point.

25. A method comprising:

providing bandwidth measurement between an ingress and an egress access router in a network by:

counting a plurality of packets at the ingress access-router;

generating a probe packet based on content of a data packet whenever a packet count reaches a specified value N; and

counting the probe packets received at the egress access-router.

26. A method as recited in claim 25, further comprising the step of discarding the probe packets through normal network mechanisms if an egress edge device is not found at a specified destination address of the probe packet.

27. A method as recited in claim 25, wherein the step of generating includes probe packets copying every Nth packet and changing the source address of the Nth packet to that of an originating access router; changing the protocol number to a reserved protocol number; attaching a UDP header with a reserved pattern; and attaching a data segment as the UDP payload which includes a probe number and a local time-stamp.

28. The method of claim 27, further comprising the step of the egress access-router identifying probe packets by the reserved protocol number and the UDP header pattern.

14

29. A method as recited in claim 25, wherein a delay between two endpoints can be determined through use of a common clock and probe packets.

30. The method as recited in claim 25, further comprising the step of counting a first number of probe packets received, and comparing the first number to a second number of packets that should have been received.

31. The method of claim 25, further comprising the step of determining a delay between two endpoints in the network by reflecting a probe packet back to a source router, and determining the round-trip time between the access routers.

32. The method of claim 25, further comprising the step of determining network faults by keeping a count of a number of probe packets received over a time interval.

33. A method for providing bandwidth accounting between a first and a second ISP access point in a network, the method comprising:

configuring at least one ingress access point to have a first packet count of 'N-in';

said at least one ingress access point keeping track of a second packet count 'N-out' of packets sent into the network; and

generating a probe packet whenever 'N-out'='N-in', wherein said probe packets being given a destination address of an Nth packet sent into the network, and being given a source address of an ingress router associated with the at least one ingress point.

34. A method as recited in claim 33, further comprising: identifying and removing the probe packet from a network data stream and an egress router associated with the at least one ingress point;

sending the probe packet to a local processor;

analyzing the probe packet at the local processor; and incrementing a packet count of packets received from the ingress access router.

35. The method of claim 34, further comprising the step of converting the packet count into an estimate of packets sent by the ingress access router.

36. The method as recited in claim 33, further comprising the step of discarding the probe packets that do not encounter an egress access router.

37. The method as recited in claim 33, further comprising the step of adjusting the probe packet according to a change in network statistics.

38. A method for forming a plurality of probes packets in a network, said method comprising:

marking of protocol field in a header of each probe packet with a reserved protocol number;

filling a source port field in a UDP header for each probe packet with the reserved pattern; and

filling a destination probe field with the probe number.

39. A method as recited in claim 38, further comprising the step of using the reserved protocol number at an egress access router to identify each probe packet.

40. The method as recited in claim 39, further comprising the step of extracting each probe packet based on an identification obtained in the step of using.

41. A method comprising:

monitoring performance of a shared network by:

forming a probe packet for each group of packets based on content of a data packet received by a router on the network;

counting the probe packets; and estimating the network bandwidth based on the packet probe count compared to a preset criteria.

15

42. A method for measuring network characteristics between a first and a second router in a network, the method comprising:

configuring at least one ingress access point on the first router to generate a plurality of probe packets;

generating each of the probe packets based on the contents of a next data packet passing through the ingress access point;

configuring at least one egress access point on the second router to detect the probe packet; and

correlating each of the probe packets received at the egress access point with each of the probe packets sent by the ingress access point to determine the network characteristics between the ingress and egress access points.

43. A method as recited in claim 42, wherein the probe packets are generated after a preset number of data packets have passed through the ingress access point.

44. A method as recited in claim 42, wherein the network characteristics being measured include network bandwidth, and the step of correlating includes comparing a count of probe packets sent by the first router to the count of probe packets received by the second router.

45. A method as recited in claim 42, wherein the egress access router removes each of the probe packets from normal data traffic after each of the probe packets is detected.

46. A method as recited in claim 42, wherein the probe packets are structured so that they are discarded through normal network mechanisms.

47. A method as recited in claim 42, wherein the step of generating includes forming each of the probe packets as follows:

copying a destination field in a particular data packet passing through the ingress access point to a probe destination field in the header of the probe packet;

forming a probe source address in the header of the probe packet to be a router source address of an originating access router; and

setting fields in the header or payload of the probe packet to be a specific value which enables the egress access point to detect the probe packet.

48. A method as recited in claim 47, wherein the protocol number in the header is changed to a special reserved protocol number.

49. A method as recited in claim 47, wherein an UDP payload is put in the probe packet with a reserved pattern and attaching a data segment as the UDP payload which includes such information as the probe number and local time-stamp.

50. An article of manufacture comprising a computer usable medium having computer readable program code means embodied therein for determining network characteristics between a first and a second access point in a network, the computer readable program code means in said article of manufacture comprising computer readable program code means for causing a computer to effect:

configuring the first access point as an ingress access point to generate a plurality of probe packets;

generating each of the probe packets based on contents of a data packet and preset criteria;

configuring the second access point as an egress access point to detect the probe packets; and

correlating each of the probe packets received at the egress access point with one of the probe packets sent

16

by the ingress access point to determine the network characteristics between the two access points.

51. An article of manufacture as recited in claim 50, wherein the network characteristics include a round-trip delay between two endpoints and the step of generating includes marking the probe packet with a time of generation, and the computer readable program code means in said article of manufacture further comprising computer readable program code means for causing a computer to effect:

the egress access point reflecting a probe packet back to the ingress access point; and

the ingress access point comparing the time of generation to a time when the probe was received back.

52. An article of manufacture as recited in claim 50, the computer readable program code means in said article of manufacture further comprising computer readable program code means for causing a computer to effect the step of the egress access point determining an expected number of probe packets to be obtained from a particular ingress access point on a basis of probe packets being received from the particular ingress access point.

53. A computer program product comprising a computer usable medium having computer readable program code means embodied therein for providing bandwidth measurement between an ingress and an egress access router in a network, the computer readable program code means in said computer program product comprising computer readable program code means for causing a computer to effect:

counting a plurality of packets at the ingress access-router;

generating a probe packet whenever a packet count reaches a specified value N; and

counting the probe packets received at the egress access-router.

54. A computer program product as recited in claim 53, the computer readable program code means in said computer program product further comprising computer readable program code means for causing a computer to effect discarding the probe packets through normal network mechanisms if an egress edge device is not found at a specified destination address of the probe packet.

55. A computer program product as recited in claim 53, wherein the step of generating includes probe packets being generated by copying every Nth packet and changing the source address of the Nth packet to that of an originating access router; changing the protocol number to a reserved protocol number; attaching a UDP header with a reserved pattern; and attaching a data segment as the UDP payload which includes a probe number and a local time-stamp.

56. A program storage device readable by machine, tangibly embodying a program of instructions executable by the machine to perform method steps for providing bandwidth accounting between a first and a second ISP access point in a network, said method steps comprising:

configuring at least one ingress access point to have a first packet count of 'N-in';

said at least one ingress access point keeping track of a second packet count 'N-out' of packets sent into the network; and

generating a probe packet when 'N-out'='N-in', wherein said probe packets being given a destination address of an Nth packet sent into the network, and being given a source address of an ingress router associated with the at least one ingress point.

57. A program storage device readable by machine as recited in claim 56, said method steps further comprising:

17

identifying and removing the probe packet from a network data stream and an egress router associated with the at least one ingress point;

sending the probe packet to a local processor; analyzing the probe packet at the local processor; and incrementing a packet count of packets received from the ingress access router.

58. A program storage device readable by machine as recited in claim 57, said method steps further comprising the step of converting the packet count into an estimate of packets sent by the ingress access router.

59. An article of manufacture comprising a computer usable medium having computer readable program code means embodied therein for measuring network characteristics between a first and a second router in a network, the computer readable program code means in said article of manufacture comprising computer readable program code means for causing a computer to effect:

configuring at least one ingress access point on the first router to generate a plurality of probe packets;

generating each of the probe packets based on the contents of a next data packet passing through the ingress access point;

configuring at least one egress access point on the second router to detect the probe packet; and

correlating each of the probe packets received at the egress access point with each of the probe packets sent by the ingress access point to determine the network characteristics between the ingress and egress access points.

60. An article of manufacture as recited in claim 59, wherein the step of generating includes forming each of the probe packets by effecting the steps of:

copying a destination field in a particular data packet passing through the ingress access point to a probe destination field in the header of the probe packet;

forming a probe source address in the header of the probe packet to be a router source address of an originating access router; and

setting fields in the header or payload of the probe packet to be a specific value which enables the egress access point to detect the probe packet.

61. An article of manufacture as recited in claim 59, wherein the network characteristics being measured include network bandwidth, and the step of correlating includes comparing a count of probe packets sent by the first router to the count of probe packets received by the second router.

62. An article of manufacture as recited in claim 59, wherein the network characteristics being measured include network bandwidth, and the step of correlating includes comparing a count of probe packets sent by the first router to the count of probe packets received by the second router.

63. An article of manufacture as recited in claim 59, wherein the egress access router removes each of the probe packets from normal data traffic after each of the probe packets is detected.

18

64. An article of manufacture as recited in claim 59, wherein the probe packets are structured so that they are discarded through normal mechanisms.

65. A computer program product comprising a computer usable medium having computer readable program code means embodied therein for forming a plurality of probes packets in an network, the computer readable program code means in said computer program product comprising computer readable program code means for causing a computer to effect:

marking a protocol field in an header of each probe packet with a reserved protocol number;

filling a source port field in a UDP header for each probe packet with the reserved pattern; and

filling a destination probe field with the probe number.

66. A computer program product as recited in claim 65, the computer readable program code means in said computer program product further comprising the step of using the reserved protocol number at an egress access router to identify each probe packet.

67. A computer program product as recited in claim 66, the computer readable program code means in said computer program product further comprising the step of extracting each probe packet based on an identification obtained in the step of using.

68. A computer program product comprising a computer usable medium having computer readable program code means embodied therein for monitoring performance of a shared network, the computer readable program code means in said computer program product comprising computer readable program code means for causing a computer to effect:

forming a probe packet for each group of packets received by a router on the network;

counting the probe packets; and

estimating the network bandwidth based on the packet probe count compared to a preset criteria.

69. A method as recited in claim 8, wherein the step of generating also includes the step of setting the source address in a probe packet header to the source address of the first access point.

70. A method comprising:

determining network characteristics between a first and a second access point in a network, by:

configuring the first access point as an ingress access point to generate a plurality of probe packets;

generating each of the probe packets based on contents of a data packet and on preset criteria;

configuring the second access point as an egress access point to detect the probe packets; and

correlating each of the probe packets received at the egress access point with one of the probe packets sent by the ingress access point to determine the network characteristics between the two access points.

* * * * *



US006115393A

United States Patent [19]
Engel et al.

[11] Patent Number: 6,115,393
[45] Date of Patent: Sep. 5, 2000

- [54] NETWORK MONITORING 5,142,528 8/1992 Kobayashi et al. 370/79
- 5,142,622 8/1992 Owens 395/200
- [75] Inventors: Ferdinand Engel, Northborough; 5,150,464 9/1992 Sidhu et al. 395/200
- Kendall S. Jones, Newton Center; 5,347,524 9/1994 l'Anson et al. 371/20.1
- Kary Robertson, Bedford, all of Mass.;
- David M. Thompson, Redmond,
- Wash.; Gerard White, Tyngsborough,
- Mass.

FOREIGN PATENT DOCUMENTS

WO 88/06822 9/1988 WIPO .

OTHER PUBLICATIONS

- [73] Assignee: Concord Communications, Inc.,
Marlboro, Mass.
- [21] Appl. No.: 08/505,083
- [22] Filed: Jul. 21, 1995
- Related U.S. Application Data
- [60] Division of application No. 07/761,269, Sep. 17, 1991,
abandoned, which is a continuation-in-part of application
No. 07/684,695, Apr. 12, 1991, abandoned.
- [51] Int. Cl.⁷ H04J 3/16; H04J 3/22
- [52] U.S. Cl. 370/469
- [58] Field of Search 370/94.1, 85.13,
370/85.14, 94.2, 110.1, 79, 241, 252, 254,
465, 464, 466, 467, 469; 395/200, 183.15,
189.01, 189.04, 200.54, 285, 831; 371/20.1

Hewlett-Packard brochure regarding local area network protocol analyzer (HP 4972A), Jun. 1987.
 F. Kaplan et al., "Application of Expert Systems to Transmission Maintenance", IEEE, 1986, pp. 449-453.
 D.M. Chiu et al., "Studying the User and Application Behaviour of a Large Network", Jun. 30, 1988, pp. 1-23.
 R. Sudama et al., "The Design of a Realtime DECnet Performance Monitor", Jul. 15, 1988, pp. 1-23.
 B.L. Hitson, "Knowledge-Based Monitoring and Control: An Approach to Understanding the Behavior of TCP/IP Network Protocols", ACM, 1988, pp. 210-221.
 A.T. Dabhura et al., "Formal Methods for Generating Protocol Conformance Test Sequences", Proceedings of the IEEE, vol. 78, No. 8, Aug. 1990, pp. 1317-1326.
 Hewlett-Packard Datasheet brochure, "Analyzing TCP/IP Networks with the HP 4972A", Sep. 1989, pp. 1-8.

Primary Examiner—Ajit Patel
Attorney, Agent, or Firm—Fish & Richardson P.C.

[56] References Cited

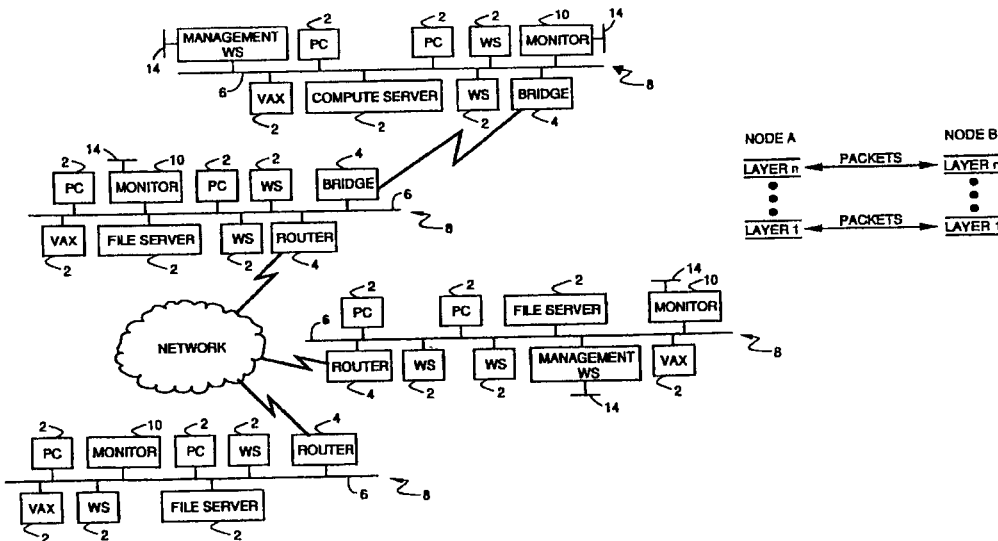
U.S. PATENT DOCUMENTS

- 4,648,061 3/1987 Foster 340/825.06
- 4,817,351 3/1989 Soha .
- 4,887,260 12/1989 Carden et al. .
- 4,930,159 5/1990 Kravitz et al. .
- 5,021,949 6/1991 Morten et al. 364/200
- 5,025,491 6/1991 Tsuchiya et al. .
- 5,038,345 8/1991 Roth 340/825.15
- 5,060,228 10/1991 Tsutsui et al. 370/85.13
- 5,097,469 3/1992 Douglas .
- 5,101,402 3/1992 Chiu et al. .
- 5,136,580 8/1992 Videlock et al. 370/85.13

[57] ABSTRACT

Monitoring is done of communications which occur in a network of nodes, each communication being effected by a transmission of one or more packets among two or more communicating nodes, each communication complying with a predefined communication protocol selected from among protocols available in the network. The contents of packets are detected passively and in real time, communication information associated with multiple protocols is derived from the packet contents.

25 Claims, 38 Drawing Sheets



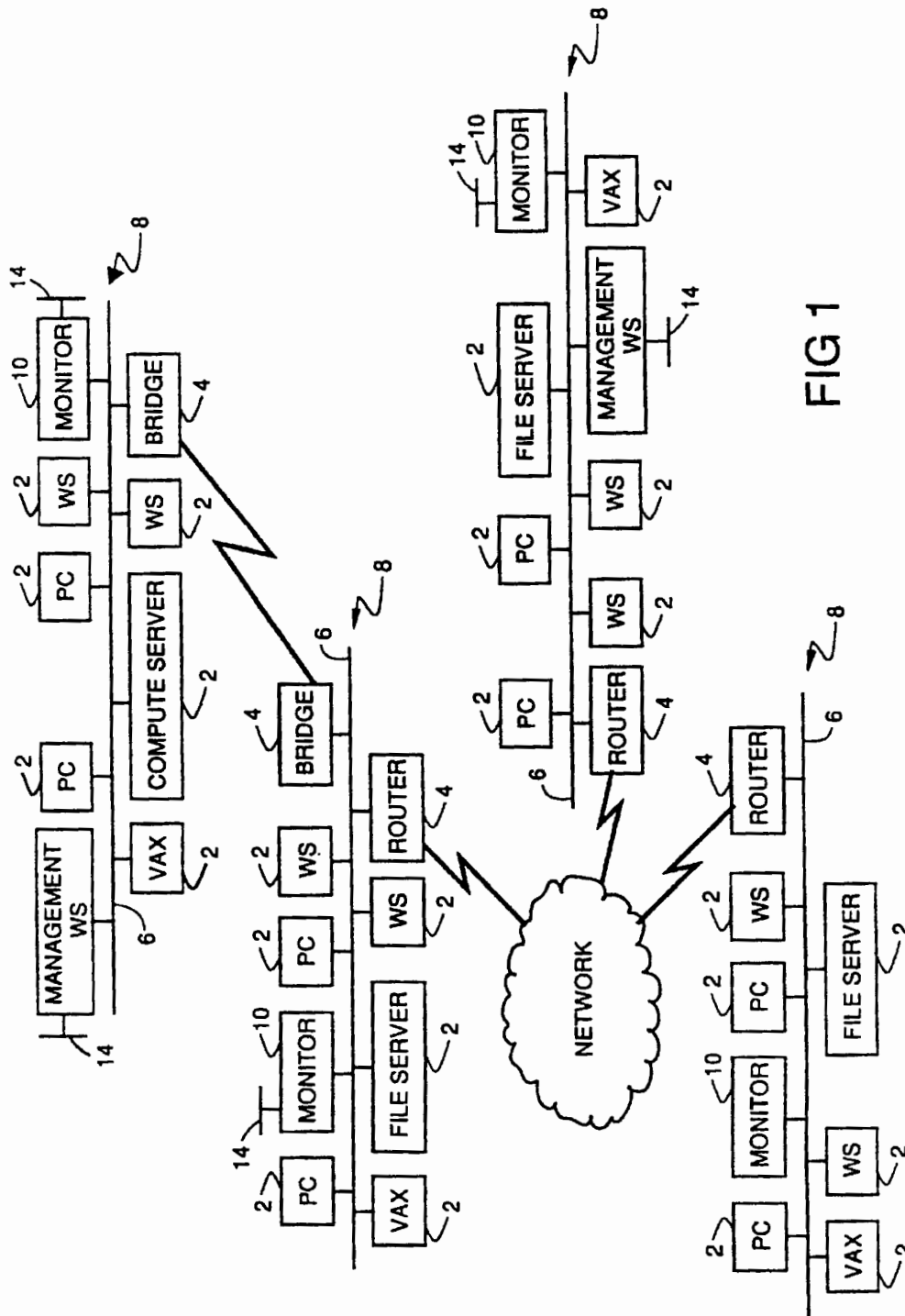


FIG 1

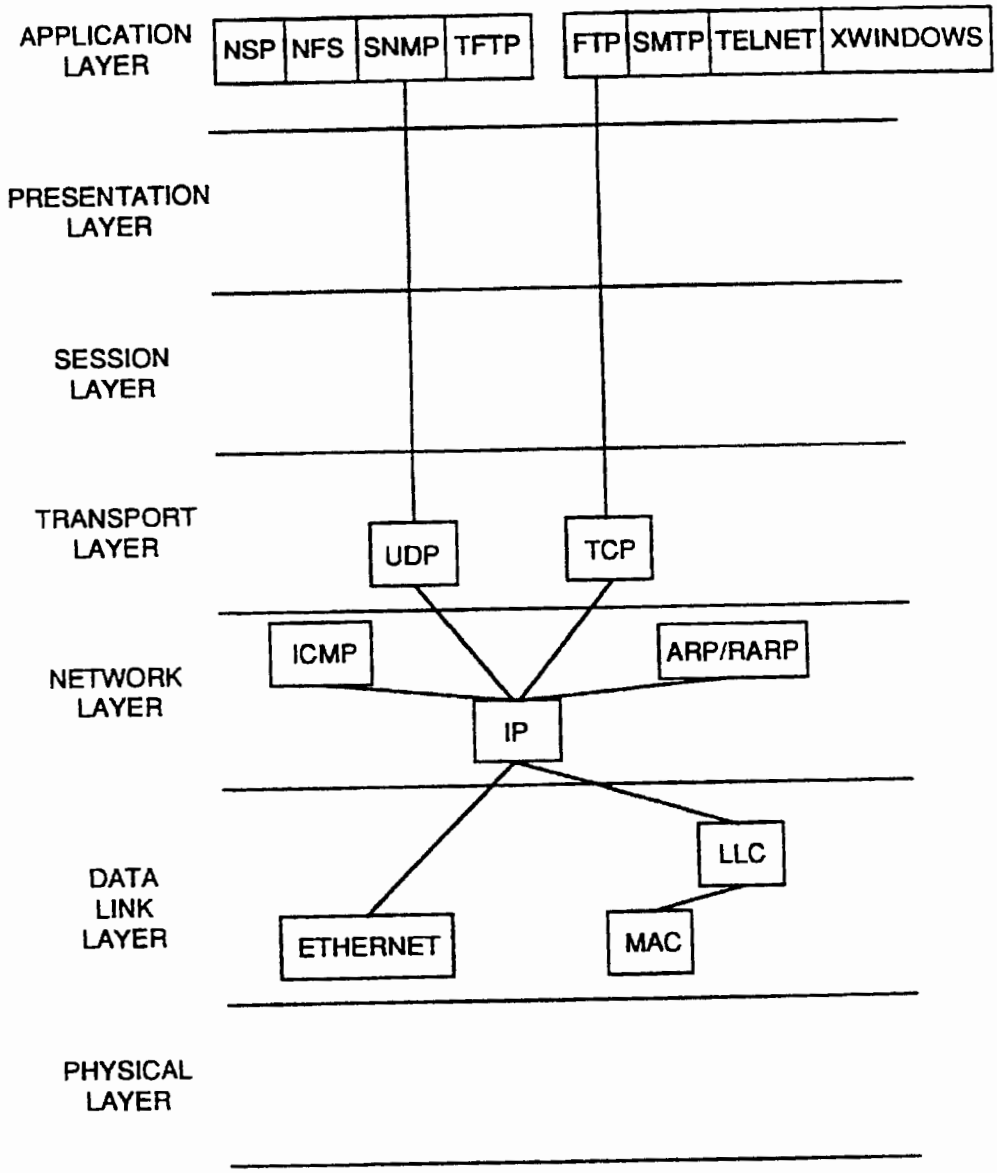


FIG 2

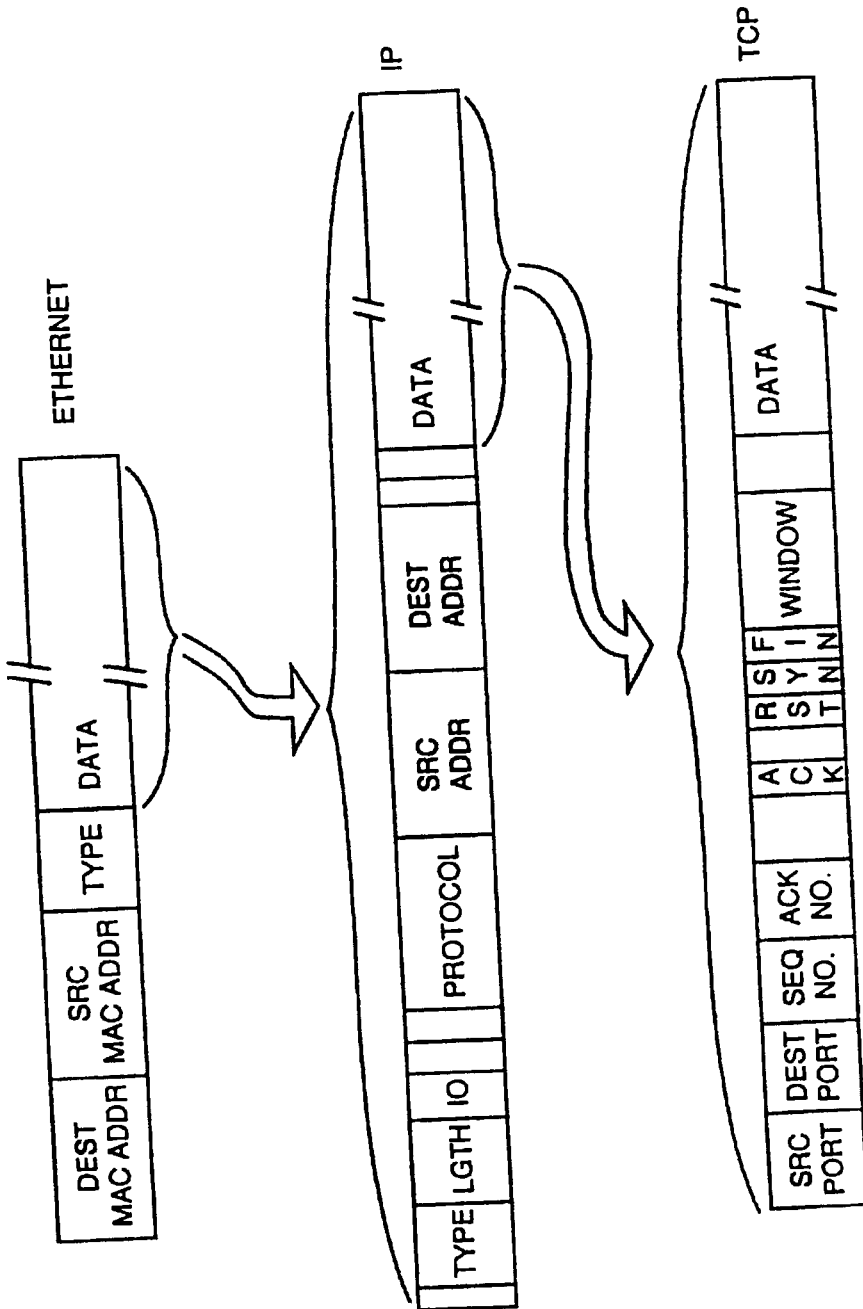


FIG 3

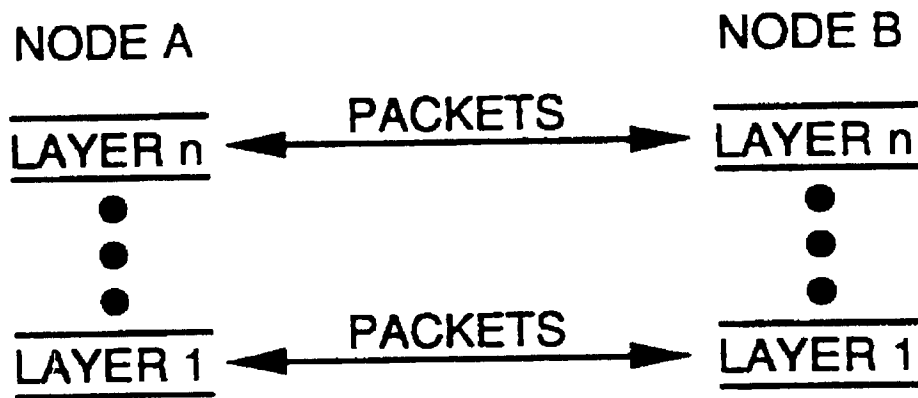


FIG 4

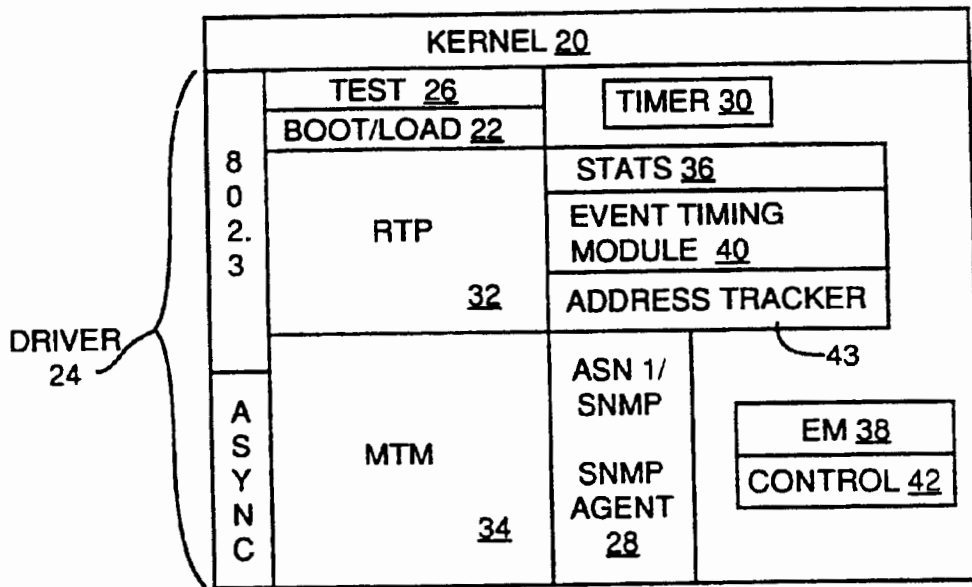


FIG 5

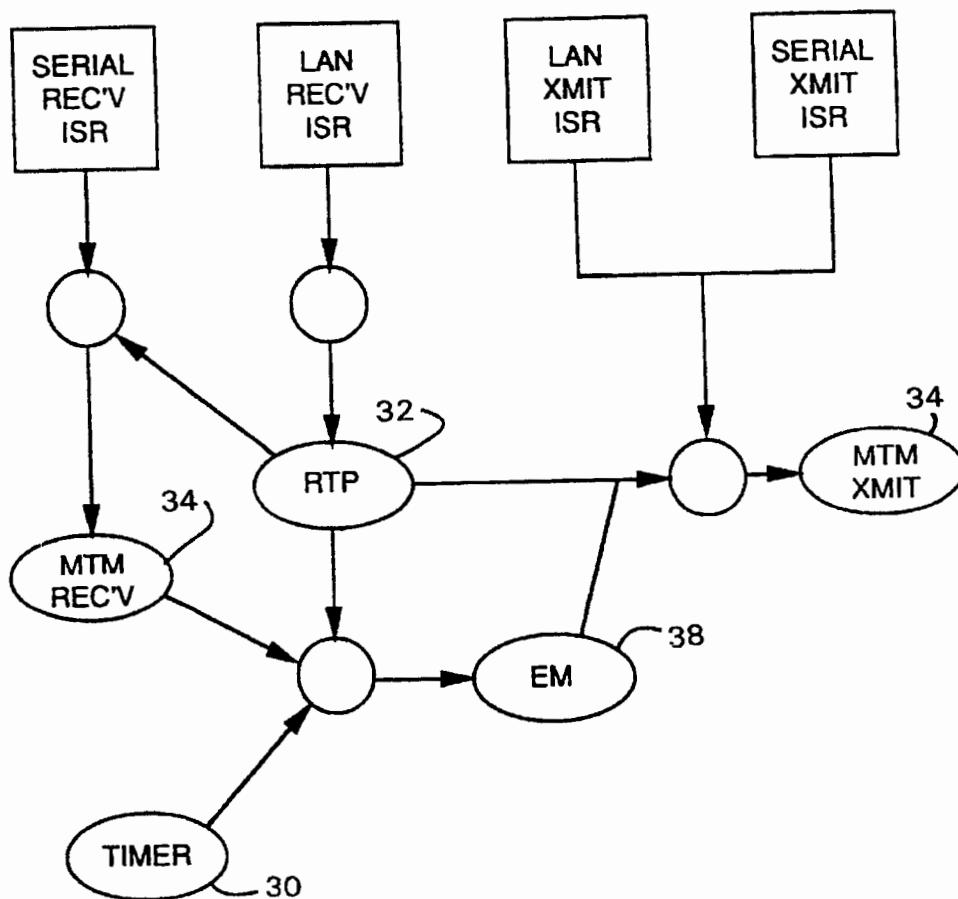


FIG 6

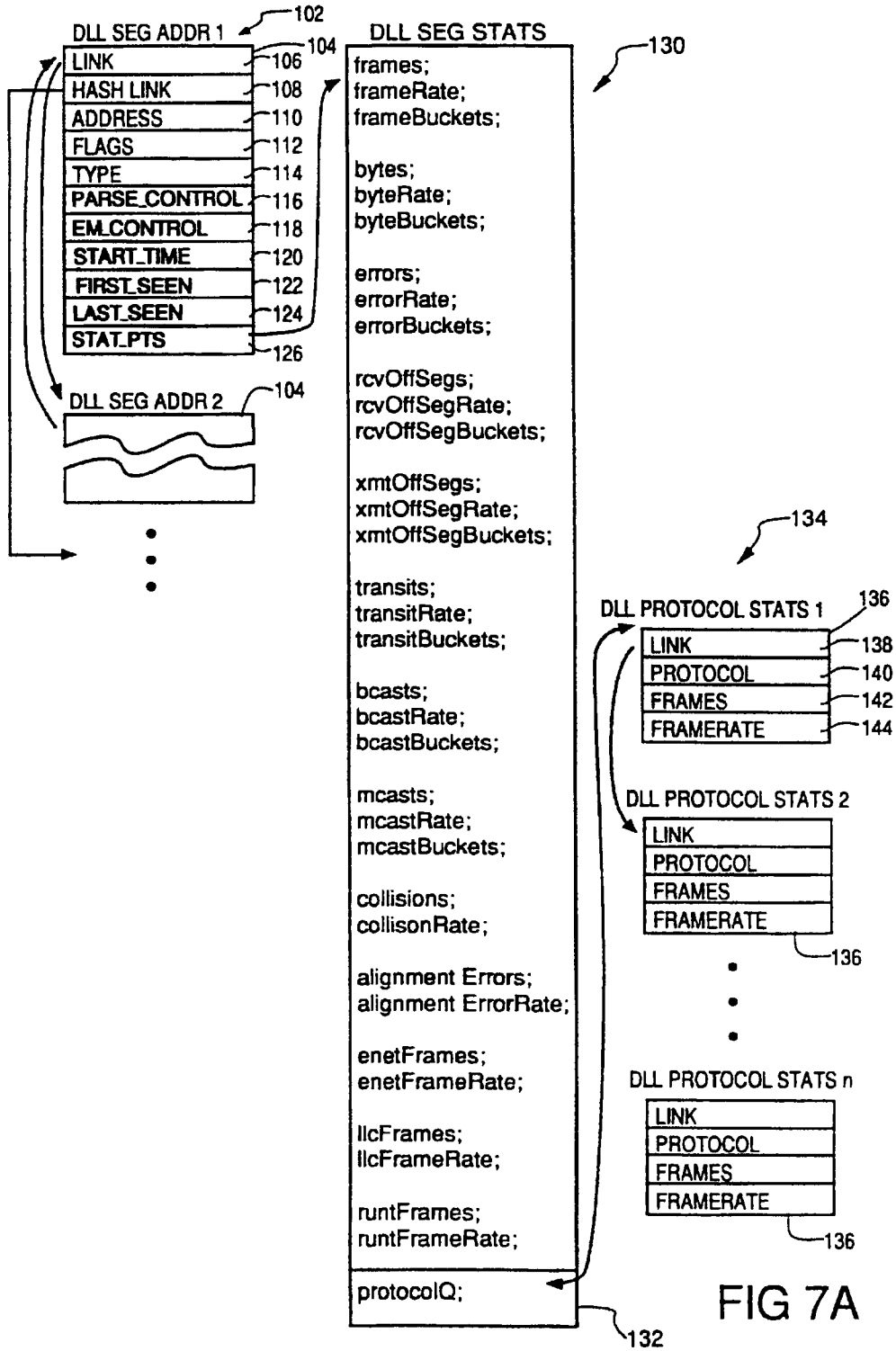


FIG 7A

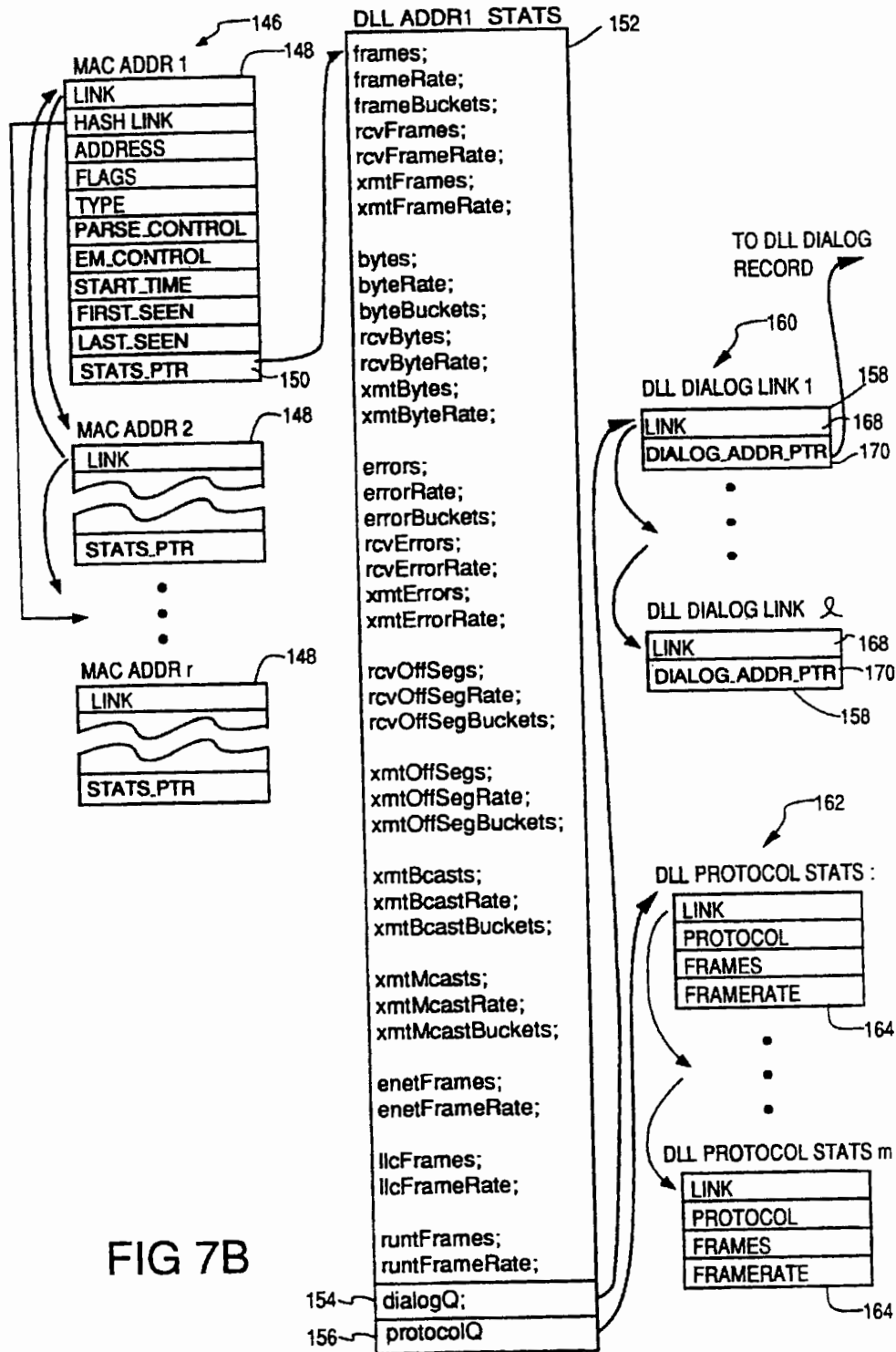


FIG 7B

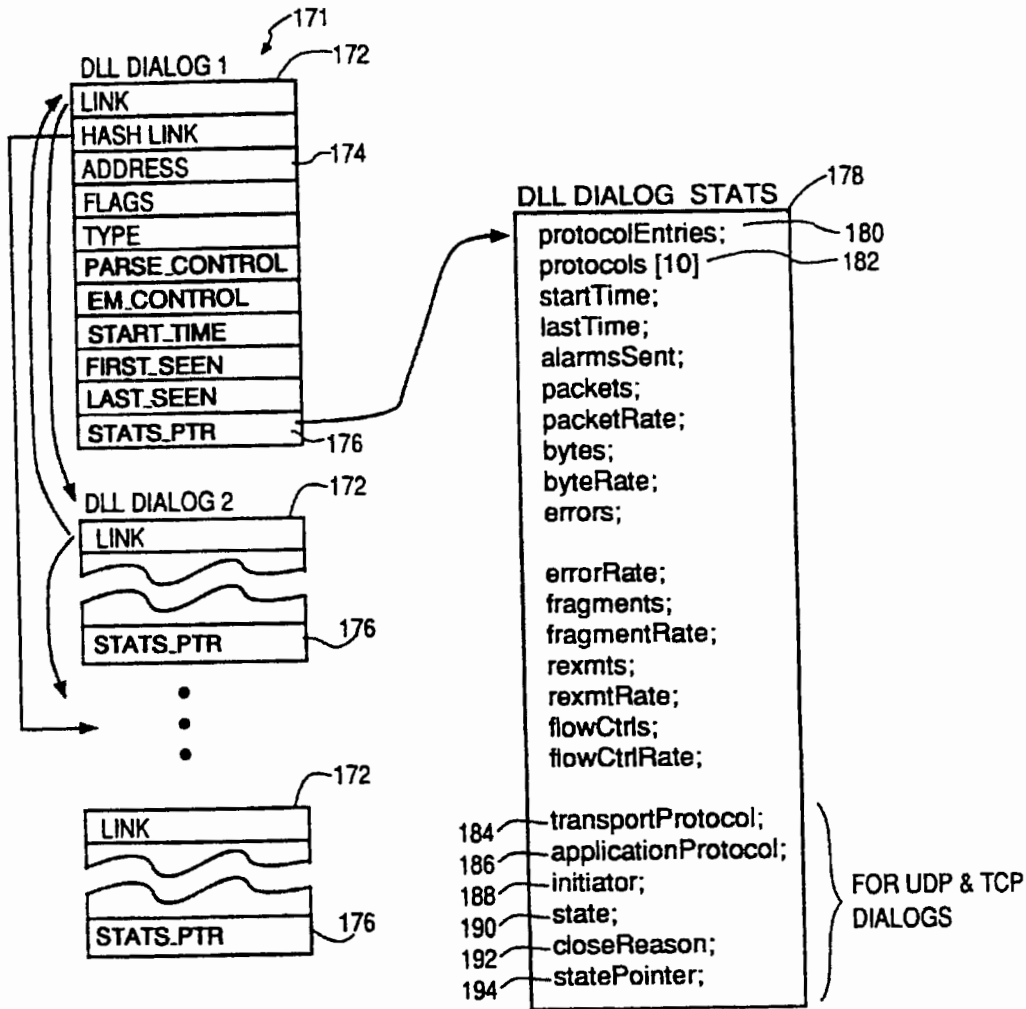


FIG 7C

STATE EVENT	UNKNOWN	CONNECTING	DATA	CLOSING	CLOSED	INACTIVE
UNKNOWN					S= UNKNOWN AFTER CLOSE ++	S= UNKNOWN
CONNECT REQ OR CONNECT CNF (E.G. TCP SYN)	S= CONNECTING	CONNECTION RETRY ++	S= UNKNOWN OUT OF ORDER++ ACTIVE CONN++	S= UNKNOWN OUT OF ORDER++	S=CONNECTING AFTER CLOSE ++	S=CONNECTING
ABORT (E.G. TCP RST)	S= CLOSED START CLOSE TIMER	S= CLOSED FAILED CONN ++ START CLOSE TIMER	S= CLOSED ACTIVE CONN -- START CLOSE TIMER	S= CLOSED ACTIVE CONN -- START CLOSE TIMER	AFTER CLOSE ++	S= CLOSED START CLOSE TIMER
DATA ACK (E.G. TCP ACK)	LOOK FOR DATA STATE	LOOK FOR DATA STATE	LOOK AT HISTORY	LOOK AT HISTORY	S= UNKNOWN AFTER CLOSE ++	S= UNKNOWN LOOK FOR DATA STATE
RELEASE REQ OR RELEASE CNF (E.G. TCP FIN)	S= CLOSING START CLOSE TIMER	S= CLOSING START CLOSE TIMER	S= CLOSING ACTIVE CONN -- START CLOSE TIMER		S= UNKNOWN AFTER CLOSE ++	S= CLOSING
DATA	LOOK FOR DATA STATE	LOOK FOR DATA STATE	LOOK AT HISTORY	OUT OF ORDER++	S= UNKNOWN AFTER CLOSE ++	S= UNKNOWN LOOK FOR DATA STATE
CLOSE TIMER EXPIRES				S= CLOSED		
INACTIVE TIMER EXPIRES	RECYCLE RESOURCES	RECYCLE RESOURCES FAILED CONN++	RECYCLE RESOURCES ACTIVE CONN --	RECYCLE RESOURCES	RECYCLE RESOURCES	RECYCLE RESOURCES

FIG 8

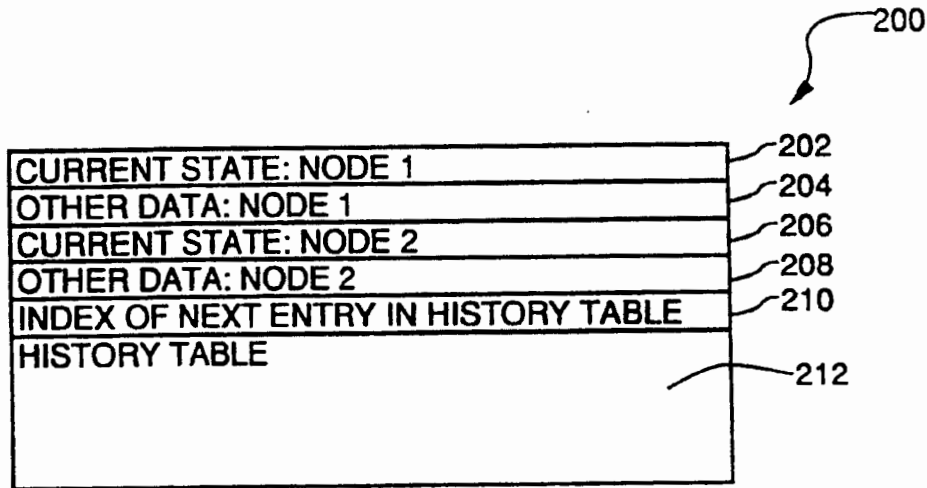


FIG 9A

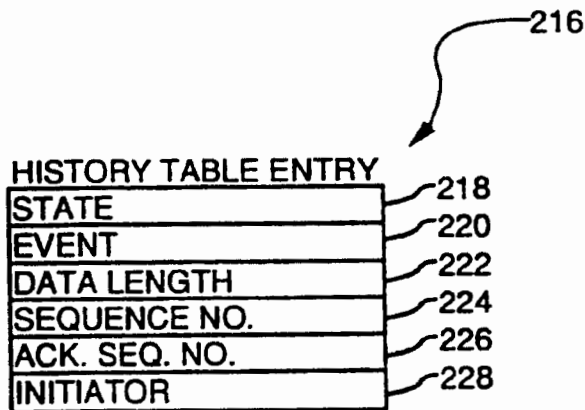
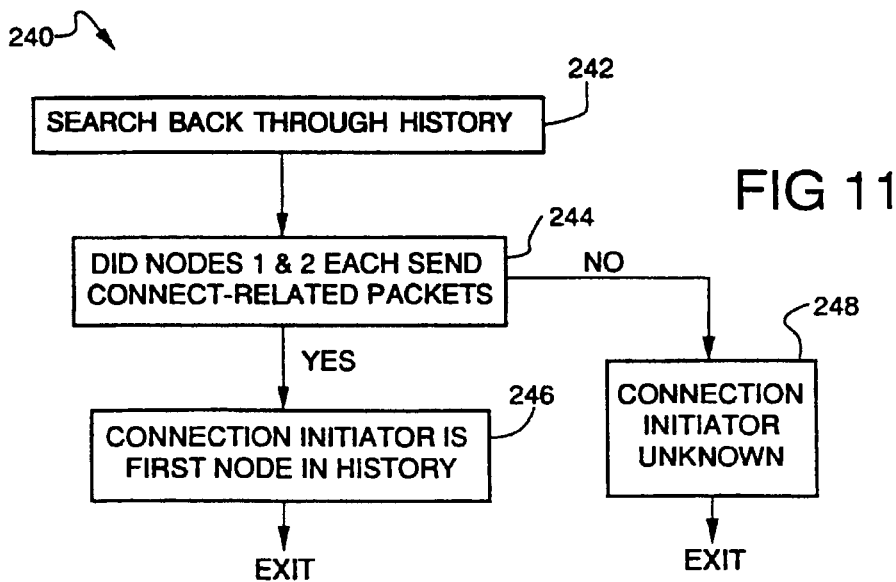
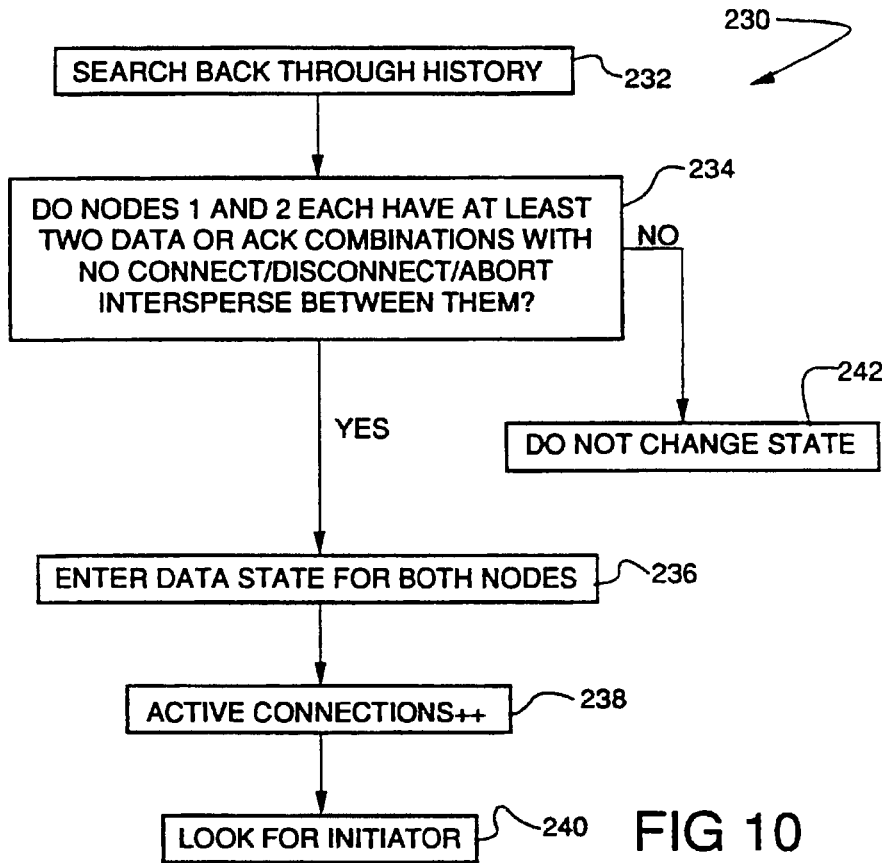


FIG 9B



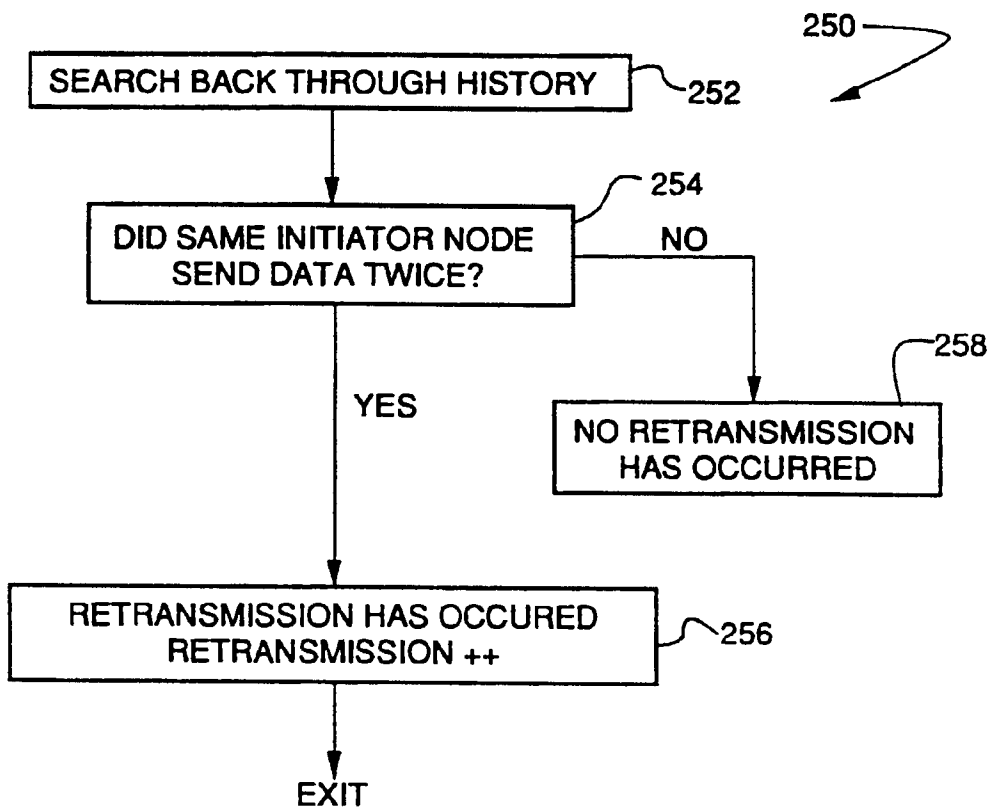


FIG 12

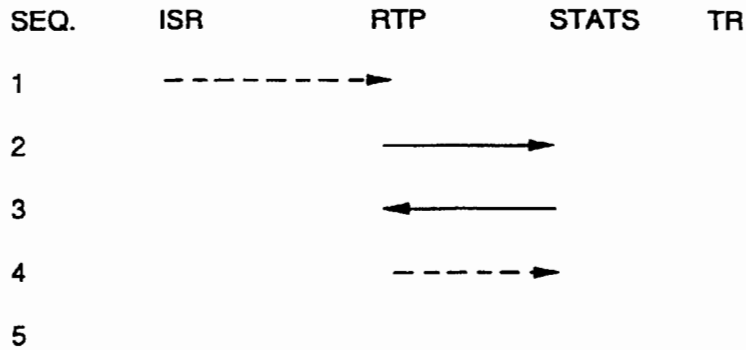


FIG 13

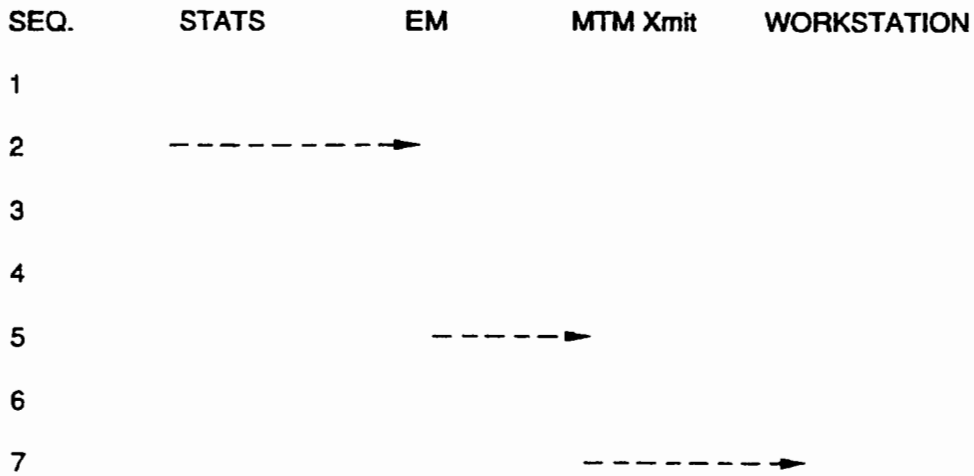


FIG 14

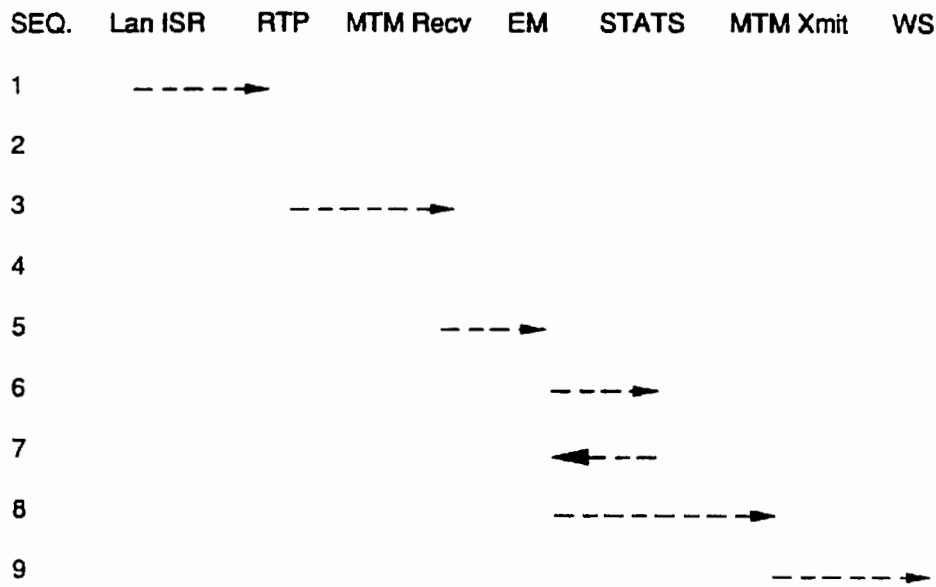


FIG 15

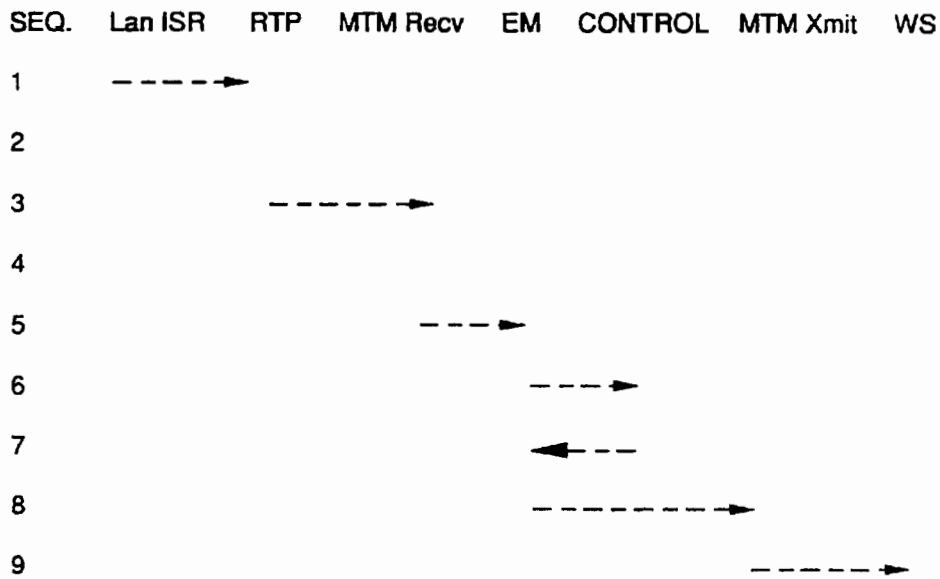


FIG 16

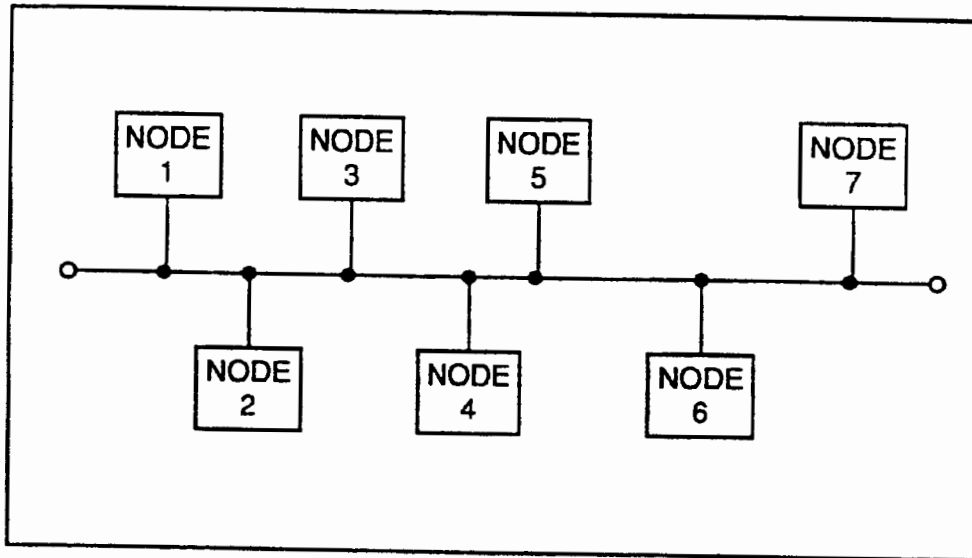


FIG 17

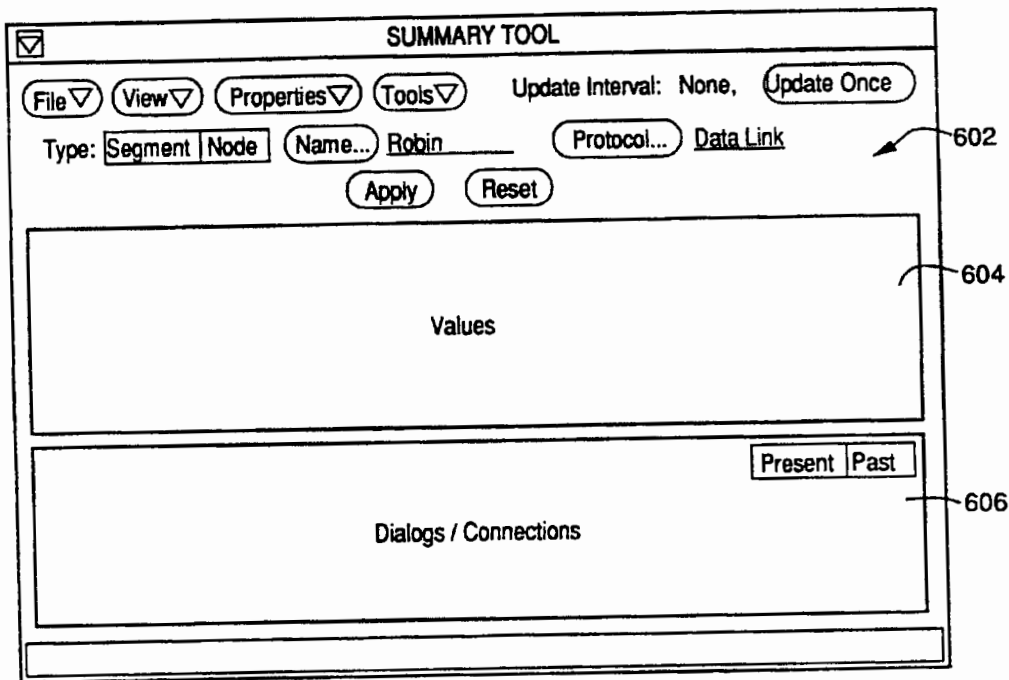


FIG 18

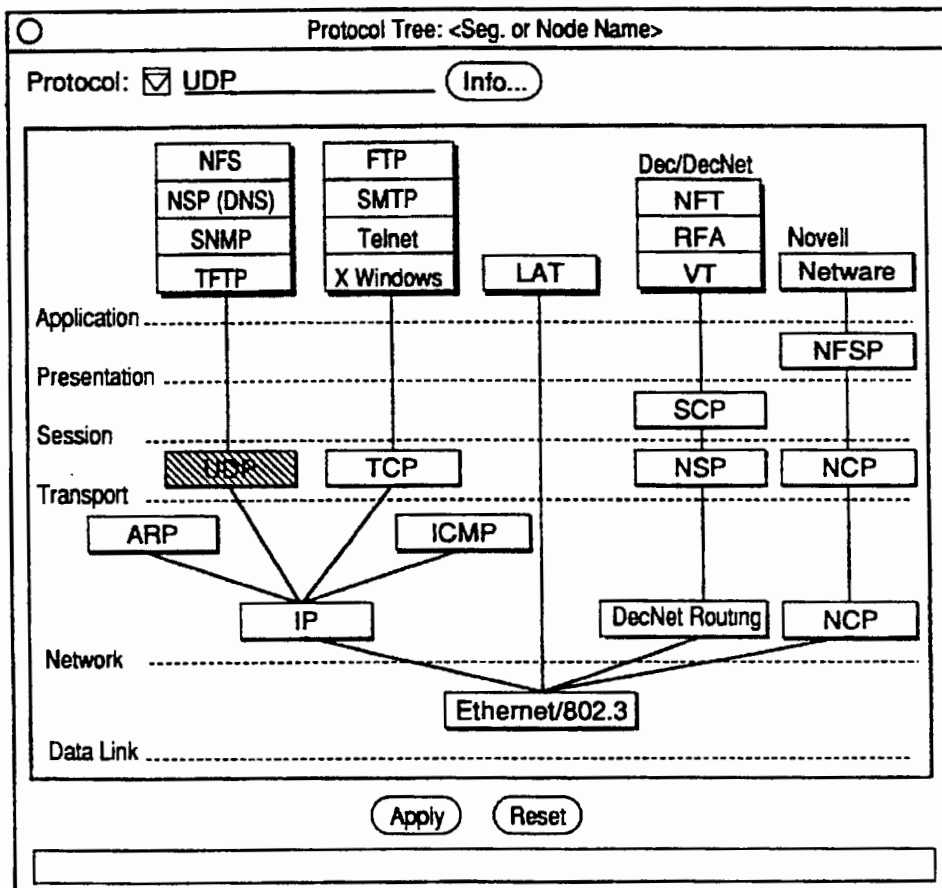


FIG 19

Data Link

	Current	5 Min.	15 Min.	10 Min. Max	60 Min. Max	Accum.Val.
Frame Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Byte Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Errors:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn
Broadcast Frm. Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Multicast Frm. Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn

Off Segment Frames

In:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
Out:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
**Transit:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn

Most Active Protocols (Frm. Rate)

1234567890123456	nnn %
<protocol>	nnn %
<protocol>	nnn %
<protocol>	nnn %
<protocol>	nnn %

Most Active Nodes (Frm. Rate)

1234567890123456	nnn %
<node>	nnn %
<node>	nnn %
<node>	nnn %
<node>	nnn %

Total Segment Bandwidth: nnn %

Total Active Dialogs: nn, nnn

FIG 20A

IP

	Current	5 Min.	15 Min.	10 Min. Max	60 Min. Max	Accum.Val.
Packet Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Byte Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Errors:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn
Broadcast Pkt. Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Multicast Pkt. Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Flow Controls:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn
Fragments:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn

Off Segment Packets

In:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
Out:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
**Transit:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn

Most Active Protocols (Pkt. Rate)

1234567890123456	nnn %
<protocol>	nnn %
<protocol>	nnn %
<protocol>	nnn %
<protocol>	nnn %

Most Active Nodes (Pkt. Rate)

1234567890123456	nnn %
<node>	nnn %
<node>	nnn %
<node>	nnn %
<node>	nnn %

Total Segment Bandwidth: nnn %

Total Active Dialogs: nn, nnn

FIG 20B

UDP

	Current	5 Min.	15 Min.	10 Min. Max	60 Min. Max	Accum.Val.
Packet Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Byte Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Errors:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn
Flow Controls:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn

Off Segment Packets

In:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
Out:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
**Transit:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn

Most Active Protocols (Pkt. Rate)

1234567890123456	nnn %
<protocol>	nnn %
<protocol>	nnn %
<protocol>	nnn %
<protocol>	nnn %

Most Active Nodes (Pkt. Rate)

1234567890123456	nnn %
<node>	nnn %
<node>	nnn %
<node>	nnn %
<node>	nnn %

Total Segment Bandwidth: nnn %

Total Active Dialogs: nn, nnn

FIG 20C

TCP

	Current	5 Min.	15 Min.	10 Min. Max	60 Min. Max	Accum.Val.
Packet Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Byte Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Errors:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn
Flow Controls:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn
Retransmissions:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn

Off Segment Packets

In:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
Out:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
**Transit:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn

Most Active Protocols (Pkt. Rate)

1234567890123456	nnn %
<protocob>	nnn %
<protocob>	nnn %
<protocob>	nnn %
<protocob>	nnn %

Most Active Nodes (Pkt. Rate)

1234567890123456	nnn %
<node>	nnn %
<node>	nnn %
<node>	nnn %
<node>	nnn %

Total Segment Bandwidth: nnn %

Total Active Connections: nn, nnn

FIG 20D

ICMP

	Current	5 Min.	15 Min.	10 Min. Max	60 Min. Max	Accum.Val.
Packet Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Byte Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Errors:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn

Off Segment Packets

In:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
Out:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
**Transit:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn

ICMP Types Seen (Count)

Address Mask:	nnn,nnn	Redirect:	nnn,nnn
Dst. Unreachable:	nnn,nnn	Source Quench:	nnn,nnn
Echo:	nnn,nnn	Time Exceeded:	nnn,nnn
Param. Problem:	nnn,nnn	Time Stamp:	nnn,nnn

Most Active Nodes (Pkt. Rate)

1234567890123456	nnn %
<node>	nnn %
<node>	nnn %
<node>	nnn %
<node>	nnn %

Total Segment Bandwidth: nnn %

FIG 20E

NFS

	Current	5 Min.	15 Min.	10 Min. Max	60 Min. Max	Accum.Val.
Packet Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Byte Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Errors:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn
Flow Controls:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn

Off Segment Packets

In:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
Out:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
**Transit:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn

Most Active Nodes (Pkt. Rate)

1234567890123456	nnn %
<node>	nnn %
<node>	nnn %
<node>	nnn %
<node>	nnn %

Total Segment Bandwidth: nnn %

Total Active Dialogs: nn, nnn

FIG 20F

Arp/Rarp	Current	5 Min.	15 Min.	10 Min. Max	60 Min. Max	Accum.Val.
Packet Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Byte Rate:	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	nn,nnn /s	n,nnn,nnn
Errors:	nnn,nnn	nnn,nnn	nnn,nnn	-	-	n,nnn,nnn
Off Segment Packets						
In:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
Out:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn
**Transit:	nnn %	nnn %	nnn %	nnn %	nnn %	n,nnn,nnn

Most Active Nodes (Pkt. Rate)

1234567890123456	nnn %
<node>	nnn %
<node>	nnn %
<node>	nnn %
<node>	nnn %

Total Segment Bandwidth: nnn %

FIG 20G

Start Time	Last Seen	Dir.	Partner Node	Protocols	Packets Summary			
					Rate	%	Count	Errors
hh:mm:ss	hh:mm:ss		1234	567890123456	nn,nnn /s	nnn %	n,nnn,nnn	nnn,nnn
10:23:04	15:31:47	To	robin	XNS,XEROX-PUP	325 /s	6%	2,641	0
07:21:38	13:25:51	From	hawk	DOD-IP, X25	87 /s	3%	127	1
10/31/90	08:22:30	?	hawk	BBN-SIMNET APPLETALK	13 /s	1%	24,192	4

FIG 21

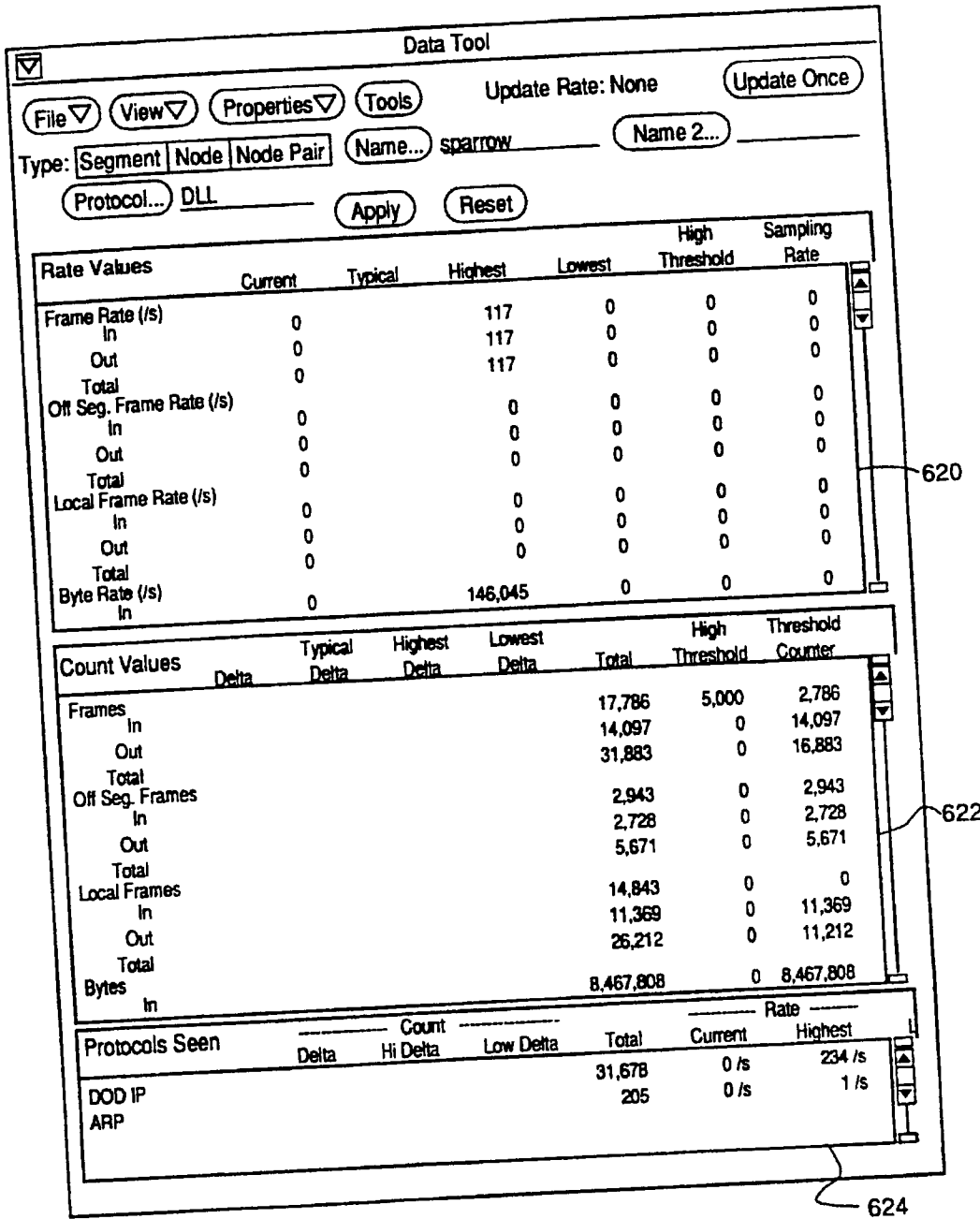


FIG 22

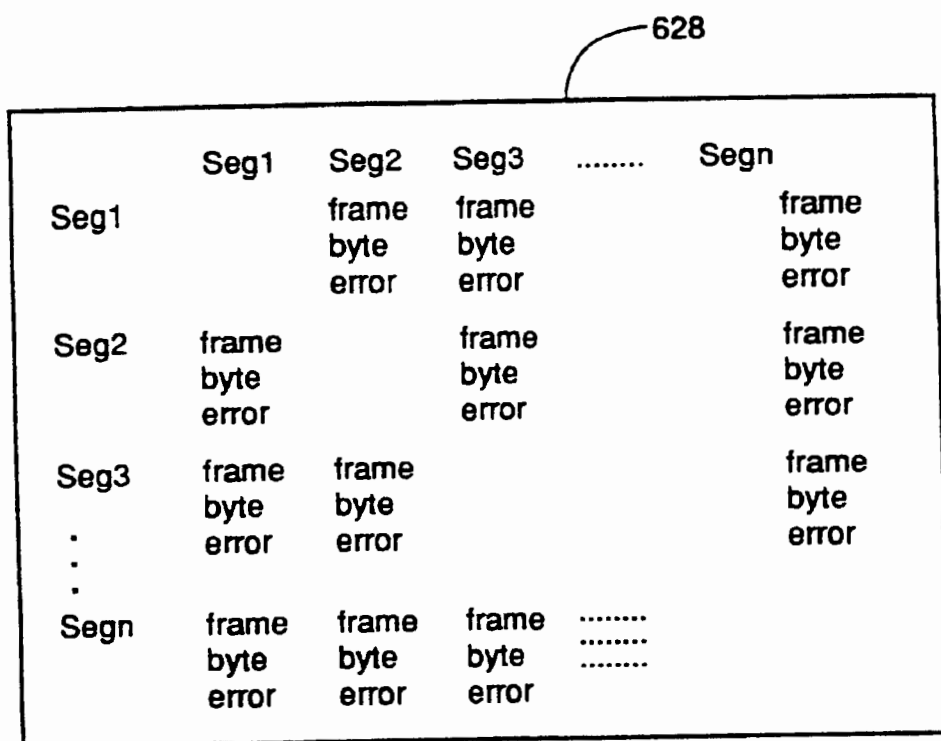


FIG 23

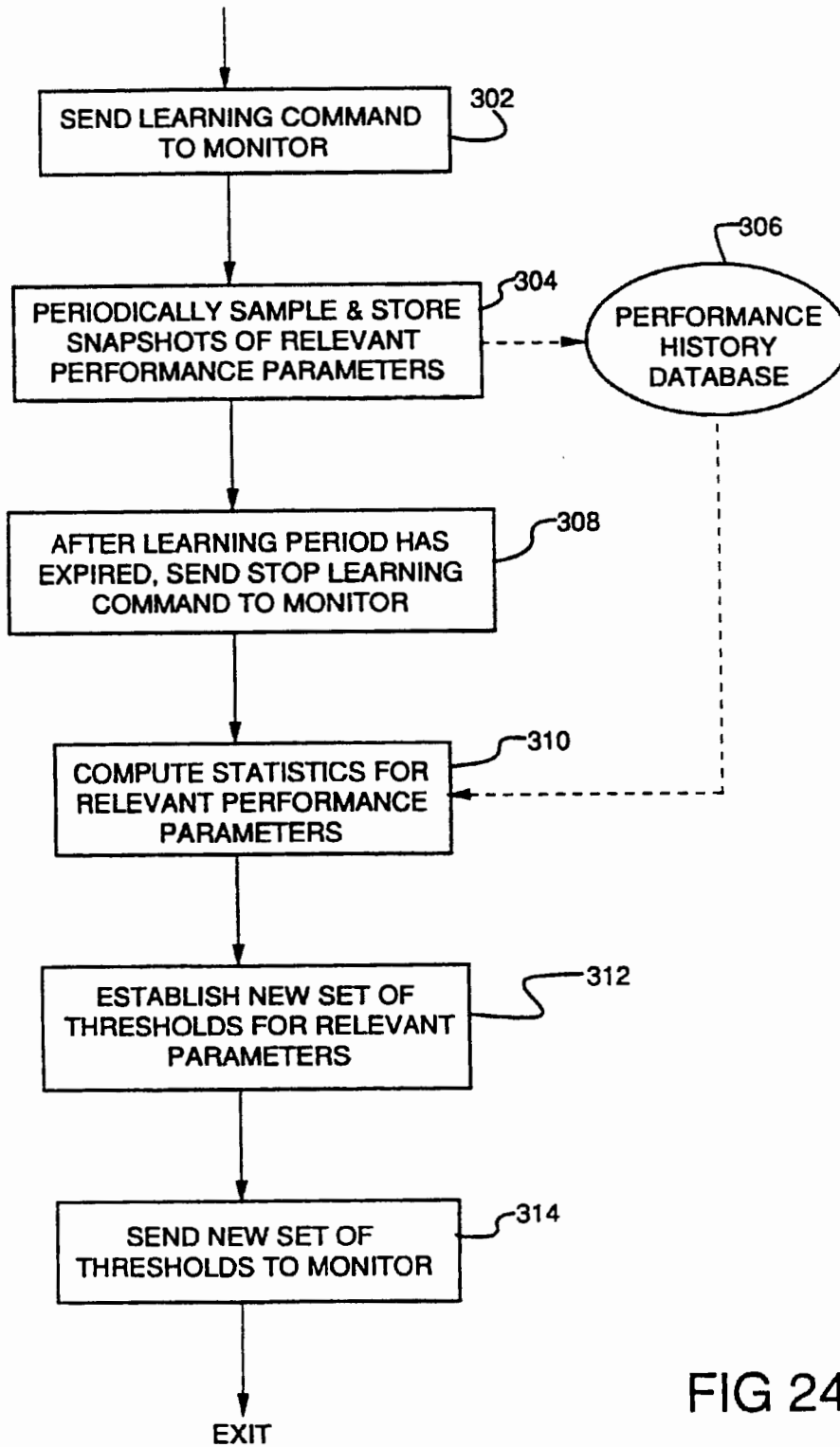


FIG 24

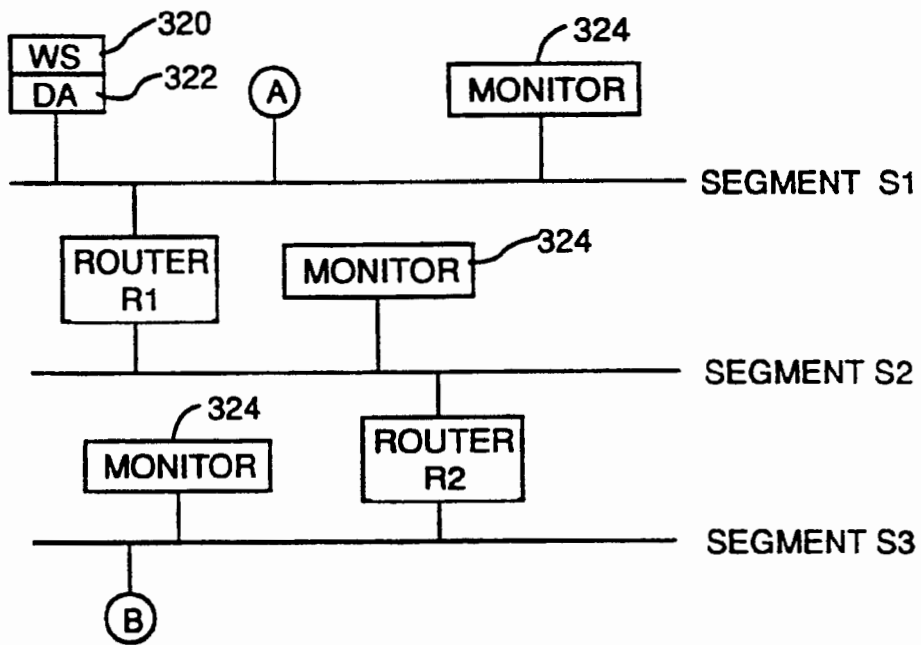


FIG 25

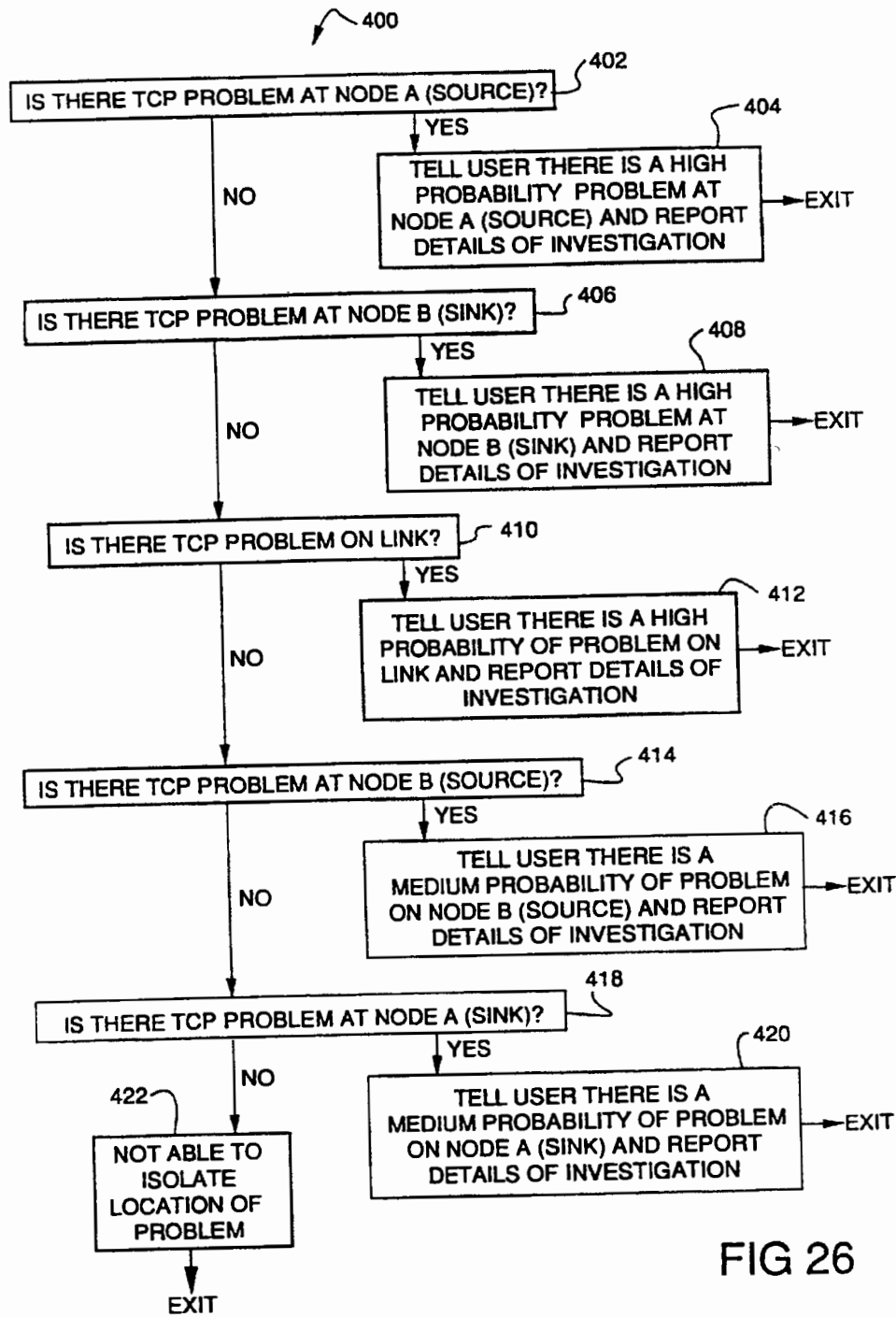


FIG 26

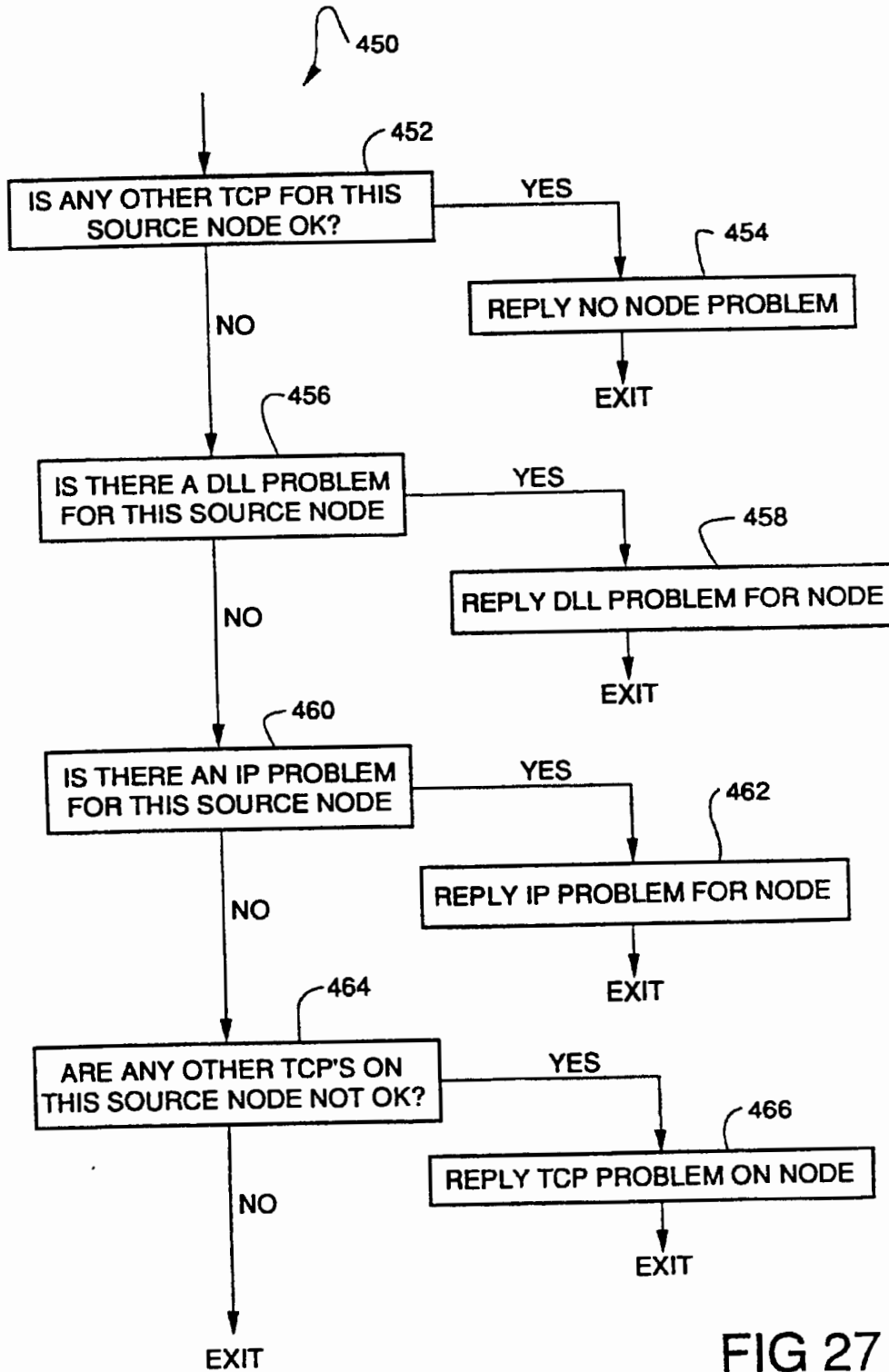


FIG 27

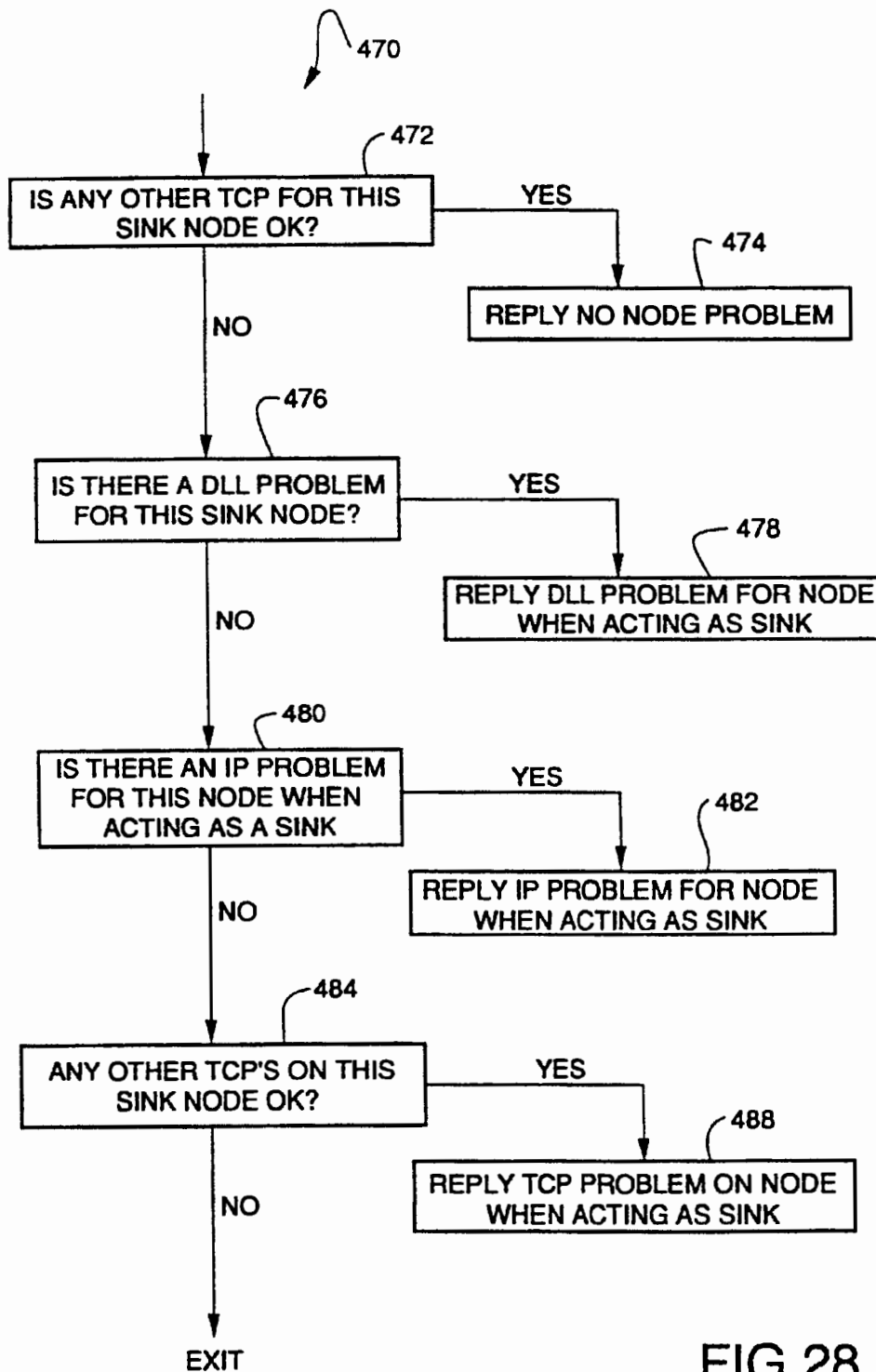


FIG 28

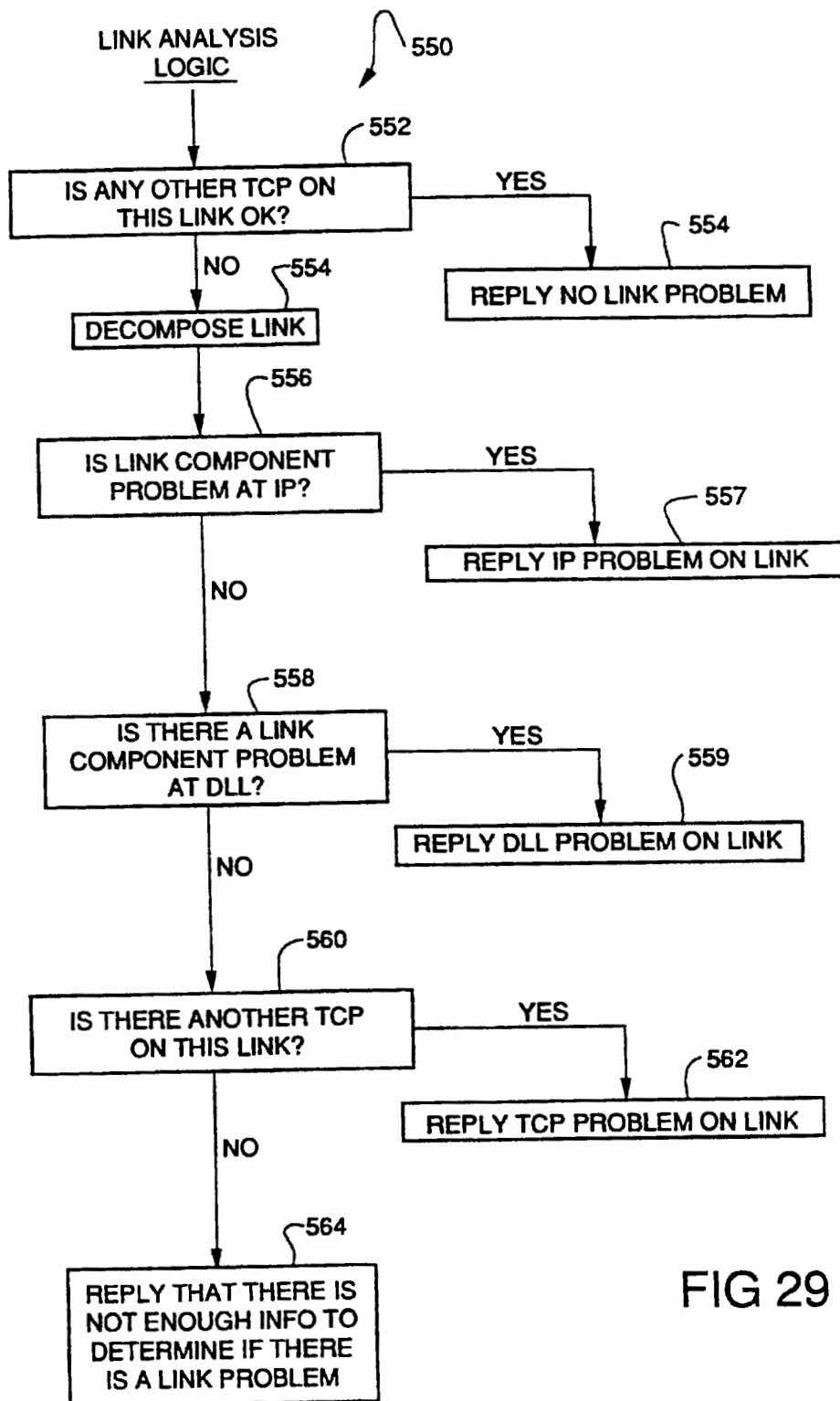
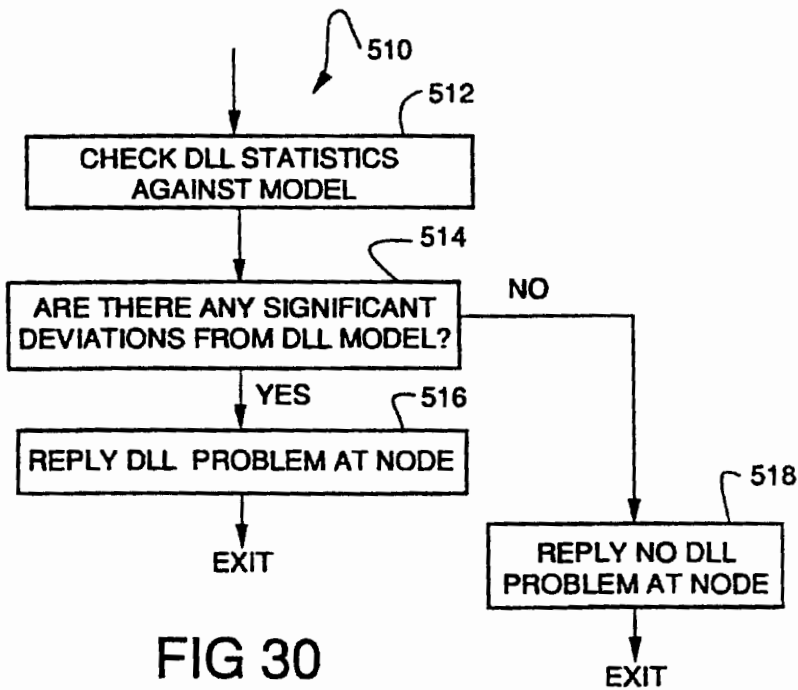
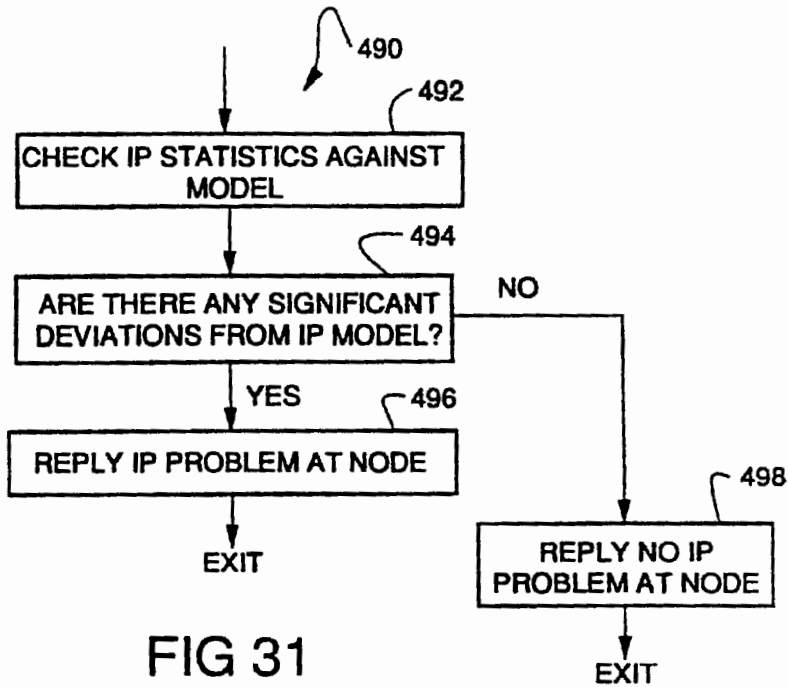


FIG 29



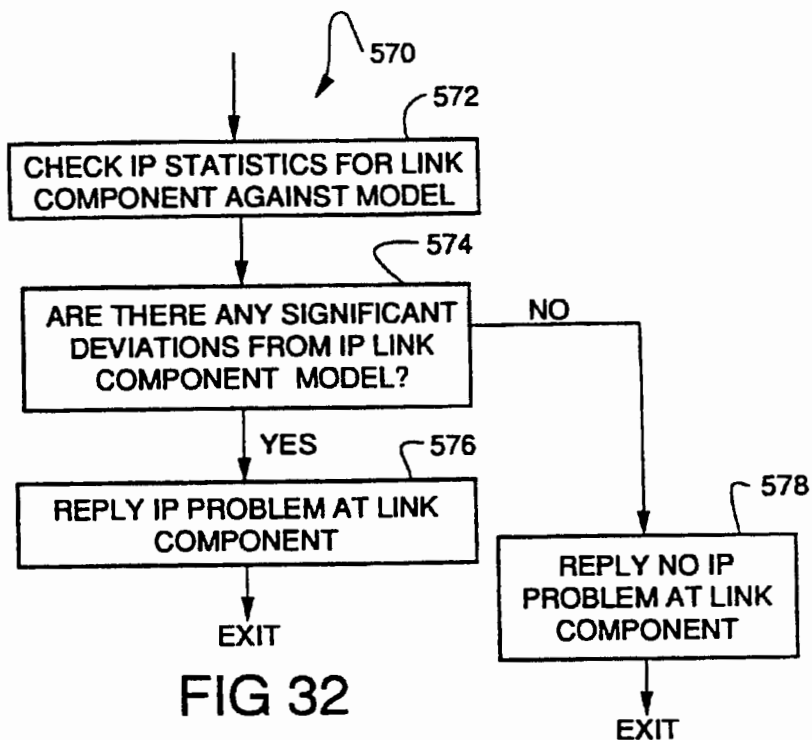


FIG 32

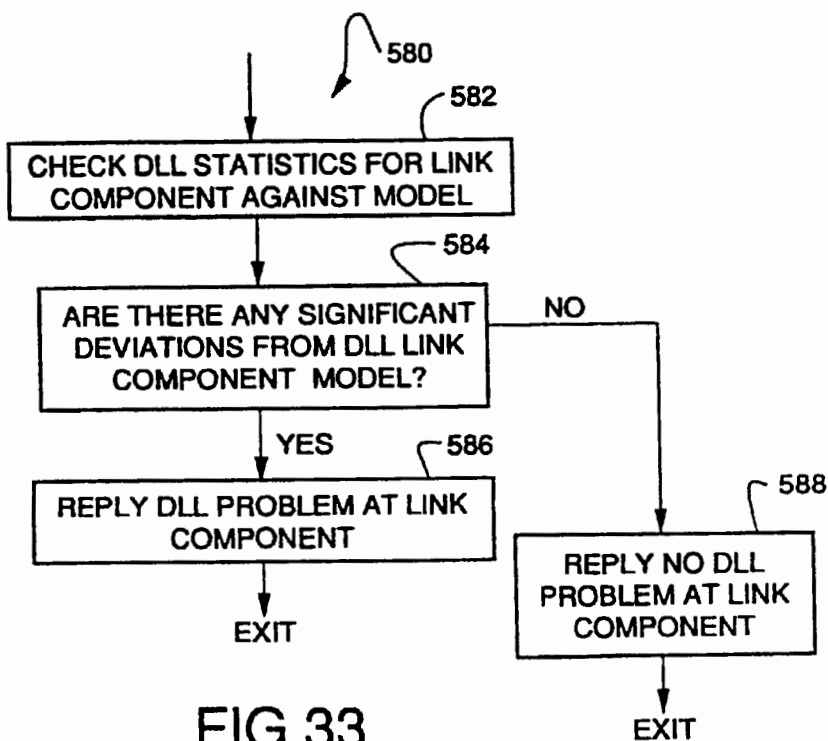


FIG 33

The diagram shows a table structure with four columns and five rows. The columns are labeled 'DIALOG', 'ENTRY TYPE', 'START TIME', and 'AVERAGE TRANSACTION TIME'. The entire table is enclosed in a rectangular border, with a reference numeral '300' pointing to the top-right corner. The right side of the table has four reference numerals '302' pointing to the right edge of each row. The bottom of the table has four reference numerals '304', '306', '308', and '310' pointing to the bottom edge of each column. The table is divided into five rows by horizontal lines. The first two rows are fully shown, the third row is partially shown with break marks on the left and right sides, and the last two rows are also fully shown.

DIALOG	ENTRY TYPE	START TIME	AVERAGE TRANSACTION TIME

FIG 34

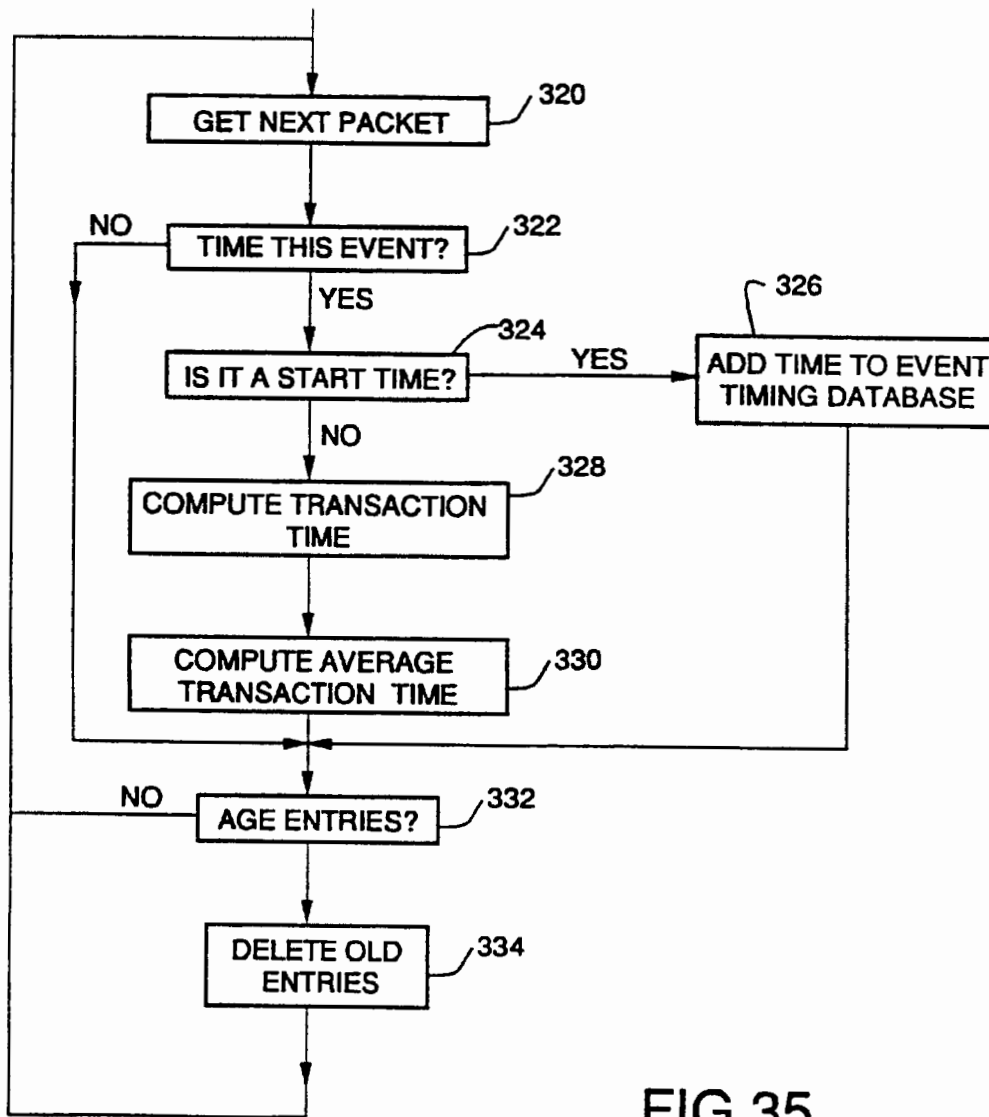


FIG 35

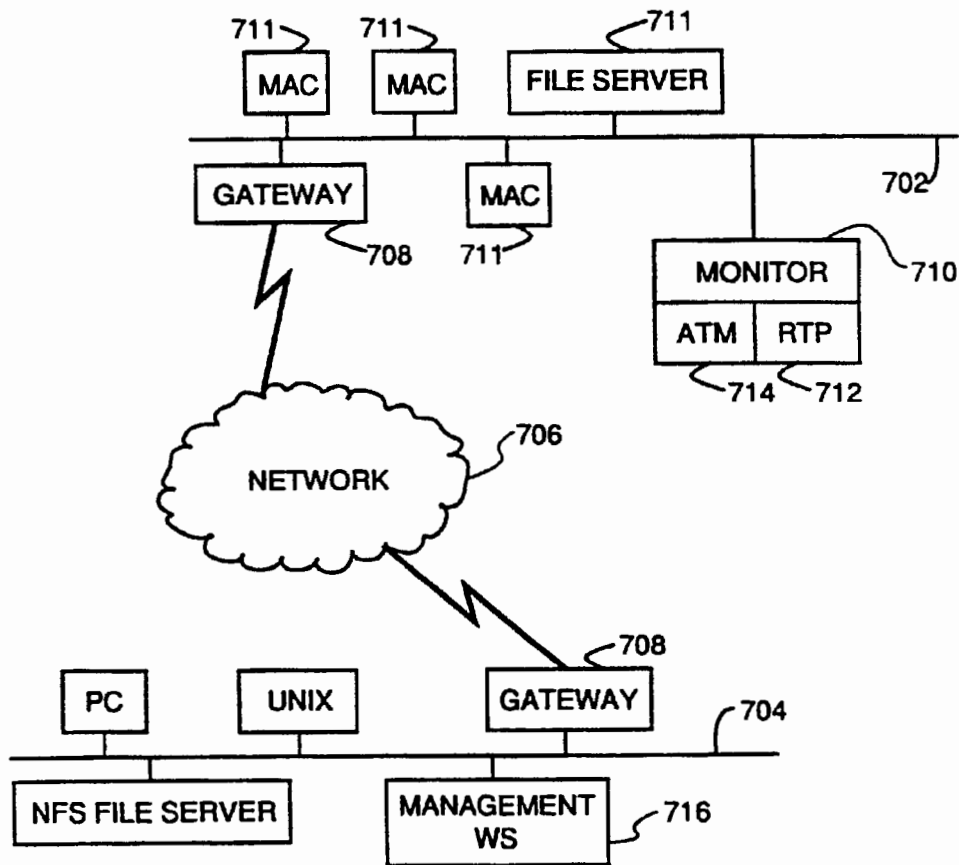


FIG 36

A table with four columns and five rows. The columns are labeled 'NODE NAME', 'NODE ADDRESS', 'IP ADDRESS', and 'TIME'. Callouts 724, 726, 728, and 729 point to the column headers. Callout 720 points to the entire table. Callouts 722 point to the four data rows. A wavy line is drawn across the data rows.

724 NODE NAME	726 NODE ADDRESS	728 IP ADDRESS	729 TIME

FIG 37

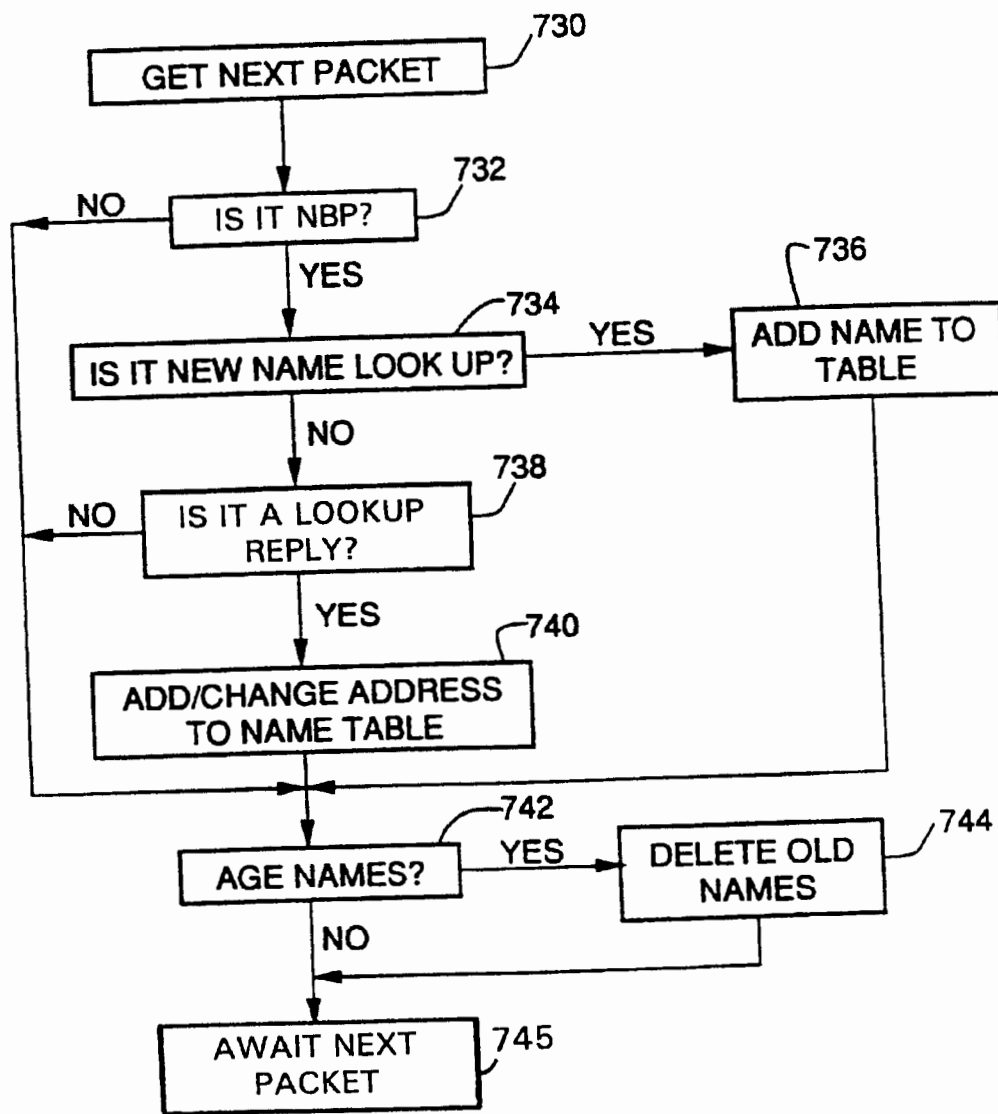


FIG 38

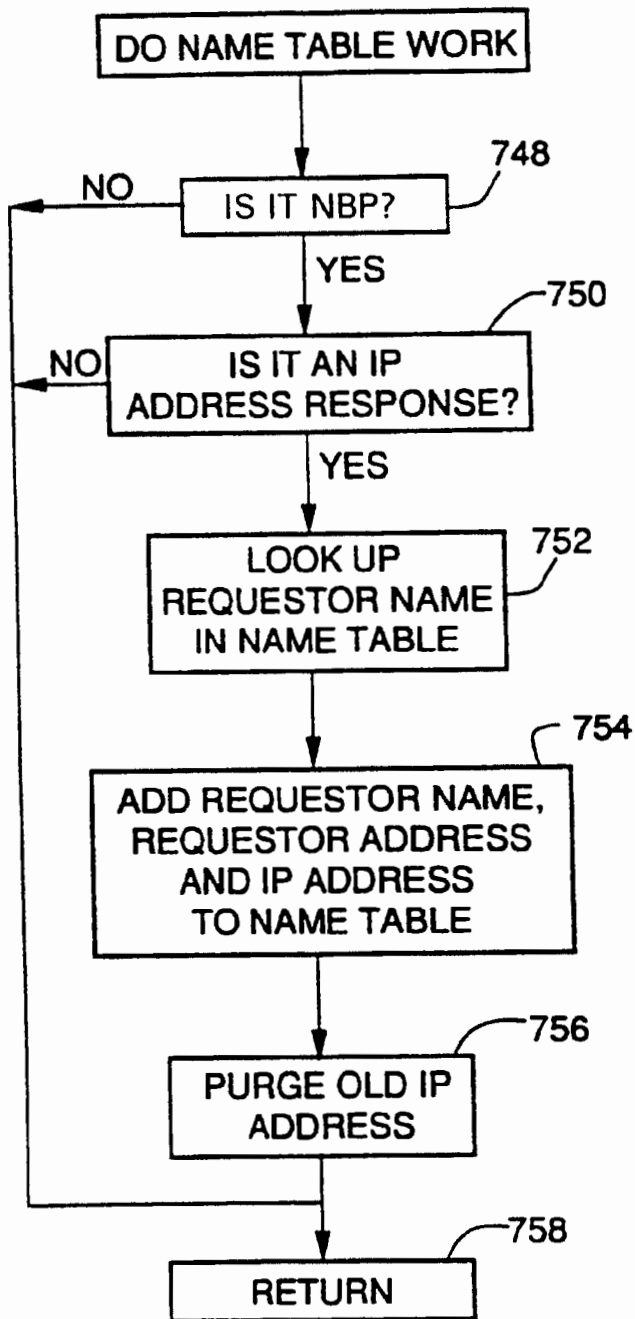


FIG 39

NETWORK MONITORING

CROSS REFERENCE TO RELATED APPLICATION

This is a divisional of application Ser. No. 07/761,269 filed on Sep. 17, 1991, now abandoned, which is a continuation-in-part of U.S. patent application, Ser. No. 07/684,695, filed Apr. 12, 1991, now abandoned.

REFERENCE TO MICROFICHE APPENDIX

A Microfiche Appendix containing fourteen microfiche accompanies this patent application pursuant to 37 CFR §1.96(b) and is designated as Appendix VI. The first thirteen microfiche each contain 49 frames and the fourteenth microfiche contains 18 frames.

BACKGROUND OF THE INVENTION

The invention relates to monitoring and managing communication networks for computers.

Today's computer networks are large complex systems with many components from a large variety of vendors. These networks often span large geographic areas ranging from a campus-like setting to world wide networks. While the network itself can be used by many different types of organizations, the purpose of these networks is to move information between computers. Typical applications are electronic mail, transaction processing, remote database, query, and simple file transfer. Usually, the organization that has installed and is running the network needs the network to be running properly in order to operate its various controls provided by the different equipment to control and manage the network. Network management is the task of planning, engineering, securing and operating a network.

To manage the network properly, the Network Manager has some obvious needs. First, the Network Manager must trouble shoot problems. As the errors develop in a running network, the Network Manager must have some tools that notify him of the errors and allow him to diagnose and repair these errors. Second, the Network Manager needs to configure the network in such a manner that the network loading characteristics provide the best service possible for the network users. To do this the Network Manager must have tools that allow him visibility into access patterns, bottlenecks and general loading. With such data, the Network Manager can reconfigure the network components for better service.

There are many different components that need to be managed in the network. These elements can be, but are not limited to: routers, bridges, PC's, workstations, minicomputers, supercomputers, printers, file servers, switches and pbx's. Each component provides a protocol for reading and writing the management variables in the machine. These variables are usually defined by the component vendor and are usually referred to as a Management Information Base (MIB). There are some standard MIB's, such as the IETF (Internet Engineering Task Force) MIB I and MIB II standard definitions. Through the reading and writing of MIB variables, software in other computers can manage or control the component. The software in the component that provides remote access to the MIB variables is usually called an agent. Thus, an individual charged with the responsibility of managing a large network often will use various tools to manipulate the MIB's of various agents on the network.

Unfortunately, the standards for accessing MIBs are not yet uniformly provided nor are the MIB definitions complete

enough to manage an entire network. The Network Manager must therefore use several different types of computers to access the agents in the network. This poses a problem, since the errors occurring on the network will tend to show up in different computers and the Network Manager must therefore monitor several different screens to determine if the network is running properly. Even when the Network Manager is able to accomplish this task, the tools available are not sufficient for the Network Manager to function properly.

Furthermore, there are many errors and loadings on the network that are not reported by agents. Flow control problems, retransmissions, on-off segment loading, network capacities and utilizations are some of the types of data that are not provided by the agents. Simple needs like charging each user for actual network usage are impossible.

SUMMARY OF THE INVENTION

In general, in one aspect, the invention features monitoring communications which occur in a network of nodes, each communication being effected by a transmission of one or more packets among two or more communicating nodes, each communication complying with a predefined communication protocol selected from among protocols available in the network. The contents of packets are detected passively and in real time, communication information associated with multiple protocols is derived from the packet contents.

Preferred embodiments of the invention include the following features. The communication information derived from the packet contents is associated with multiple layers of at least one of the protocols.

In general, in another aspect, the invention features monitoring communication dialogs which occur in a network of nodes, each dialog being effected by a transmission of one or more packets among two or more communicating nodes, each dialog complying with a predefined communication protocol selected from among protocols available in the network. Information about the states of dialogs occurring in the network and which comply with different selected protocols available in the network is derived from the packet contents.

Preferred embodiments of the invention include the following features. A current state is maintained for each dialog, and the current state is updated in response to the detected contents of transmitted packets. For each dialog, a history of events is maintained based on information derived from the contents of packets, and the history of events is analyzed to derive information about the dialog. The analysis of the history includes counting events and gathering statistics about events. The history is monitored for dialogs which are inactive, and dialogs which have been inactive for a predetermined period of time are purged. For example, the current state is updated to data state in response to observing the transmission of at least two data related packets from each node. Sequence numbers of data related packets stored in the history of events are analyzed and retransmissions are detected based on the sequence numbers. The current state is updated based on each new packet associated with the dialog; if an updated current state cannot be determined, information about prior packets associated with the dialog is consulted as an aid in updating the state. The history of events may be searched to identify the initiator of a dialog.

The full set of packets associated with a dialog up to a point in time completely define a true state of the dialog at that point in time, and the step of updating the current state in response to the detected contents of transmitted packets includes generating a current state (e.g., "unknown") which

5

FIG. 14 is a diagram of the major steps in the processing of a statistics threshold event;

FIG. 15 is a diagram of the major steps in the processing of a database update;

FIG. 16 is a diagram of the major steps in the processing of a monitor control request;

FIG. 17 is a logical map of the network as displayed by the Management Workstation;

FIG. 18 is a basic summary tool display screen;

FIG. 19 is a protocol selection menu that may be invoked through the summary tool display screen;

FIGS. 20a-g are examples of the statistical variables which are displayed for different protocols;

FIG. 21 is an example of information that is displayed in the dialogs panel of the summary tool display screen;

FIG. 22 is a basic data screen presenting a rate values panel, a count values panel and a protocols seen panel;

FIG. 23 is a traffic matrix screen;

FIG. 24 is a flow diagram of the algorithm for adaptively establishing network thresholds based upon actual network performance;

FIG. 25 is a simple multi-segment network;

FIG. 26 is a flow diagram of the operation of the diagnostic analyzer algorithm;

FIG. 27 is a flow diagram of the source node analyzer algorithm;

FIG. 28 is a flow diagram of the sink node analyzer algorithm;

FIG. 29 is a flow diagram of the link analysis logic;

FIG. 30 is a flow diagram of the DLL problem checking routine;

FIG. 31 is a flow diagram of the IP problem checking routine;

FIG. 32 is a flow diagram of the IP link component problem checking routine;

FIG. 33 is a flow diagram of the DLL link component problem checking routine;

FIG. 34 shows the structure of the event timing database;

FIG. 35 is a flow diagram of the operation of the event timing module (ETM) in the Network Monitor;

FIG. 36 is a network which includes an Appletalk® segment;

FIG. 37 is a Name Table that is maintained by the Address Tracking Module (ATM);

FIG. 38 is a flow diagram of the operation of the ATM; and

FIG. 39 is a flow diagram of the operation of the ATM. Also attached hereto before the claims are the following appendices:

Appendix I identifies the SNMP MIB subset that is supported by the Monitor and the Management Workstation (2 pages);

Appendix II defines the extension to the standard MIB that are supported by the Monitor and the Management Workstation (25 pages);

Appendix III is a summary of the protocol variables for which the Monitor gathers statistics and a brief description of the variables, where appropriate (17 pages);

Appendix IV is a list of the Summary Tool Values Display Fields with brief descriptions (2 pages); and

Appendix V is a description of the actual screens for the Values Tool (34 pages).

6

Appendix VI is a microfiche appendix presenting source code for the real time parser, the statistics data structures and the statistics modules.

Structure and Operation

The Network:

A typical network, such as the one shown in FIG. 1, includes at least three major components, namely, network nodes 2, network elements 4 and communication lines 6. Network nodes 2 are the individual computers on the network. They are the very reason the network exists. They include but are not limited to workstations (WS), personal computers (PC), file servers (FS), compute servers (CS) and host computers (e.g., a VAX), to name but a few. The term server is often used as though it was different from a node, but it is, in fact, just a node providing special services.

In general, network elements 4 are anything that participate in the service of providing data movement in a network, i.e., providing the basic communications. They include, but are not limited to, LAN's, routers, bridges, gateways, multiplexors, switches and connectors. Bridges serve as connections between different network segments. They keep track of the nodes which are connected to each of the segments to which they are connected. When they see a packet on one segment that is addressed to a node on another of their segments, they grab the packet from the one segment and transfer it to the proper segment. Gateways generally provide connections between different network segments that are operating under different protocols and serve to convert communications from one protocol to the other. Nodes send packets to routers so that they may be directed over the appropriate segments to the intended destination node.

Finally, network or communication lines 6 are the components of the network which connect nodes 2 and elements 4 together so that communications between nodes 2 may take place. They can be private lines, satellite lines or Public Carrier lines. They are expensive resources and are usually managed as separate entities. Often networks are organized into segments 8 that are connected by network elements 4. A segment 8 is a section of a LAN connected at a physical level (this may include repeaters). Within a segment, no protocols at layers above the physical layer are needed to enable signals from two stations on the same segment to reach each other (i.e., there are no routers, bridges, gateways . . .).

The Network Monitor and the Management Workstation:

In the described embodiment, there are two basic elements to the monitoring system which is to be described, namely, a Network Monitor 10 and a Management Workstation 12. Both elements interact with each other over the local area network (LAN).

Network Monitor 10 (referred to hereinafter simply as Monitor 10) is the data collection module which is attached to the LAN. It is a high performance real time front end processor which collects packets on the network and performs some degree of analysis to search for actual or potential problems and to maintain statistical information for use in later analysis. In general, it performs the following functions. It operates in a promiscuous mode to capture and analyze all packets on the segment and it extracts all items of interest from the frames. It generates alarms to notify the Management Workstation of the occurrence of significant events. It receives commands from the Management Workstation, processes them appropriately and returns responses.

Management Workstation 12 is the operator interface. It collects and presents troubleshooting and performance infor-

mation to the user. It is based on the SunNet Manager (SNM) product and provides a graphical network-map-based interface and sophisticated data presentation and analysis tools. It receives information from Monitor 10, stores it and displays the information in various ways. It also instructs Monitor 10 to perform certain actions. Monitor 10, in turn, sends responses and alarms to Management Workstation 12 over either the primary LAN or a backup serial link 14 using SNMP with the MIB extensions defined later.

These devices can be connected to each other over various types of networks and are not limited to connections over a local area network. As indicated in FIG. 1, there can be multiple Workstations 12 as well as multiple Monitors 10.

Before describing these components in greater detail, background information will first be reviewed regarding communication protocols which specify how communications are conducted over the network and regarding the structure of the packets.

The Protocol Tree:

As shown in FIG. 2, communication over the network is organized as a series of layers or levels, each one built upon the next lower one, and each one specified by one or more protocols (represented by the boxes). Each layer is responsible for handling a different phase of the communication between nodes on the network. The protocols for each layer are defined so that the services offered by any layer are relatively independent of the services offered by the neighbors above and below. Although the identities and number of layers may differ depending on the network (i.e., the protocol set defining communication over the network), in general, most of them share a similar structure and have features in common.

For purposes of the present description, the Open Systems Interconnection (OSI) model will be presented as representative of structured protocol architectures. The OSI model, developed by the International Organization for Standardization, includes seven layers. As indicated in FIG. 2, there is a physical layer, a data link layer (DLL), a network layer, a transport layer, a session layer, a presentation layer and an application layer, in that order. As background for what is to follow, the function of each of these layers will be briefly described.

The physical layer provides the physical medium for the data transmission. It specifies the electrical and mechanical interfaces of the network and deals with bit level detail. The data link layer is responsible for ensuring an error-free physical link between the communicating nodes. It is responsible for creating and recognizing frame boundaries (i.e., the boundaries of the packets of data that are sent over the network.) The network layer determines how packets are routed within the network. The transport layer accepts data from the layer above it (i.e., the session layer), breaks the packets up into smaller units, if required, and passes these to the network layer for transmission over the network. It may insure that the smaller pieces all arrive properly at the other end. The session layer is the user's interface into the network. The user must interface with the session layer in order to negotiate a connection with a process in another machine. The presentation layer provides code conversion and data reformatting for the user's application. Finally, the application layer selects the overall network service for the user's application.

FIG. 2 also shows the protocol tree which is implemented by the described embodiment. A protocol tree shows the protocols that apply to each layer and it identifies by the tree structure which protocols at each layer can run "on top of" the protocols of the next lower layer. Though standard

abbreviations are used to identify the protocols, for the convenience of the reader, the meaning of the abbreviations are as follows:

ARP	Address Resolution Protocol
ETHERNET	Ethernet Data Link Control
FTP	File Transfer Protocol
ICMP	Internet Control Message Protocol
IP	Internet Protocol
LLC	802.2 Logical Link Control
MAC	802.3 CSMA/CD Media Access Control
NFS	Network File System
NSP	Name Server Protocol
RARP	Reverse Address Resolution Protocol
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
UDP	User Datagram Protocol

Two terms are commonly used to describe the protocol tree, namely, a protocol stack and a protocol family (or suite). A protocol stack generally refers to the underlying protocols that are used when sending a message over a network. For example, FTP/TCP/IP/LLC is a protocol stack. A protocol family is a loose association of protocols which tend to be used on the same network (or derive from a common source). Thus, for example, the TCP/IP family includes IP, TCP, UDP, ARP, TELNET and FTP. The Decnet family includes the protocols from Digital Equipment Corporation. And the SNA family includes the protocols from IBM.

The Packet:

The relevant protocol stack defines the structure of each packet that is sent over the network. FIG. 3, which shows an TCP/IP packet, illustrates the typical structure of a packet. In general, each level of the protocol stack takes the data from the next higher level and adds header information to form a protocol data unit (PDU) which it passes to the next lower level. That is, as the data from the application is passed down through the protocol layers in preparation for transmission over the network, each layer adds its own information to the data passed down from above until the complete packet is assembled. Thus, the structure of a packet resembles that of an onion, with each PDU of a given layer wrapped within the PDU of the adjacent lower level.

At the ethernet level, the PDU includes a destination address (DEST MAC ADDR), a source address (SRC MAC ADDR), a type (TYPE) identifying the protocol which is running on top of this layer, and a DATA field for the PDU from the IP layer.

Like the ethernet packet, the PDU for the IP layer includes an IP header plus a DATA field. The IP header includes a type field (TYPE) for indicating the type of service, a length field (LGTH) for specifying the total length of the PDU, an identification field (ID), a protocol field (PROT) for identifying the protocol which is running on top of the IP layer (in this case, TCP), a source address field (SRC ADDR) for specifying the IP address of the sender, a destination address field (DEST ADDR) for specifying the IP address of the destination node, and a DATA field.

The PDU built by the TCP protocol also consists of a header and the data passed down from the next higher layer. In this case the header includes a source port field (SRC PORT) for specifying the port number of the sender, a destination port field (DEST PORT) for specifying the port number of the destination, a sequence number field (SEQ NO.) for specifying the sequence number of the data that is being sent in this packet, and an acknowledgment number

field (ACK NO.) for specifying the number of the acknowledgment being returned. It also includes bits which identify the packet type, namely, an acknowledgment bit (ACK), a reset connection bit (RST), a synchronize bit (SYN), and a no more data from sender bit (FIN). There is also a window size field (WINDOW) for specifying the size of the window being used.

The Concept of a Dialog:

The concept of a dialog is used throughout the following description. As will become apparent, it is a concept which provides a useful way of conceptualizing, organizing and displaying information about the performance of a network—for any protocol and for any layer of the multi-level protocol stack.

As noted above, the basic unit of information in communication is a packet. A packet conveys meaning between the sender and the receiver and is part of a larger framework of packet exchanges. The larger exchange is called a dialog within the context of this document. That is, a dialog is a communication between a sender and a receiver, which is composed of one or more packets being transmitted between the two. There can be multiple senders and receivers which can change roles. In fact, most dialogs involve exchanges in both directions.

Stated another way, a dialog is the exchange of messages and the associated meaning and state that is inherent in any particular exchange at any layer. It refers to the exchange between the peer entities (hardware or software) in any communication. In those situations where there is a layering of protocols, any particular message exchange could be viewed as belonging to multiple dialogs. For example, in FIG. 4 Nodes A and B are exchanging packets and are engaged in multiple dialogs. Layer 1 in Node A has a dialog with Layer 1 in Node B. For this example, one could state that this is the data link layer and the nature of the dialog deals with the message length, number of messages, errors and perhaps the guarantee of the delivery. Simultaneously, Layer n of Node A is having a dialog with Layer n of node B. For the sake of the example, one could state that this is an application layer dialog which deals with virtual terminal connections and response rates. One can also assume that all of the other layers (2 through n-1) are also having simultaneous dialogs.

In some protocols there are explicit primitives that deal with the dialog and they are generally referred to as connections or virtual circuits. However, dialogs exist even in stateless and connectionless protocols. Two more examples will be described to help clarify the concept further, one dealing with a connection oriented protocol and the other dealing with a connectionless protocol.

In a typical connection oriented protocol, Node A sends a connection request (CR) message to Node B. The CR is an explicit request to form a connection. This is the start of a particular dialog, which is no different from the start of the connection. Nodes A and B could have other dialogs active simultaneously with this particular dialog. Each dialog is seen as unique. A connection is a particular type of dialog.

In a typical connectionless protocol, Node A sends Node B a message that is a datagram which has no connection paradigm, in fact, neither do the protocol(s) at higher layers. The application protocol designates this as a request to initiate some action. For example, a file server protocol such as Sun Microsystems' Network File System (NFS) could make a mount request. A dialog comes into existence once the communication between Nodes A and B has begun. It is possible to determine that communication has occurred and to determine the actions being requested. If in fact there

exists more than one communication thread between Nodes A and B, then these would represent separate, different dialogs.

Inside the Network Monitor:

Monitor 10 includes a MIPS R3000 general purpose microprocessor (from MIPS Computer Systems, Inc.) running at 25 MHz. It is capable of providing 20 mips processing power. Monitor 10 also includes a 64Kbyte instruction cache and a 64Kbyte data cache, implemented by SRAM.

The major software modules of Monitor 10 are implemented as a mixture of tasks and subroutine libraries as shown in FIG. 5. It is organized this way so as to minimize the context switching overhead incurred during critical processing sequences. There is NO PREEMPTION of any module in the monitor subsystem. Each module is cognizant of the fact that it should return control to the kernel in order to let other tasks run. Since the monitor subsystem is a closed environment, the software is aware of real time constraints.

Among the major modules which make up Monitor 10 is a real time kernel 20, a boot/load module 22, a driver 24, a test module 26, an SNMP Agent 28, a Timer module 30, a real time parser (RTP) 32, a Message Transport Module (MTM) 34, a statistics database (STATS) 36, an Event Manager (EM) 38, an Event Timing Module (ETM) 40 and a control module 42. Each of these will now be described in greater detail.

Real Time Kernel 20 takes care of the general housekeeping activities in Monitor 10. It is responsible for scheduling, handling intertask communications via queues, managing a potentially large number of timers, manipulating linked lists, and handling simple memory management.

Boot/Load Module 22, which is FProm based, enables Monitor 10 to start itself when the power is turned on in the box. It initializes functions such as diagnostics, and environmental initialization and it initiates down loading of the Network Monitor Software including program and configuration files from the Management Workstation. Boot/load module 22 is also responsible for reloading program and/or configuration data following internal error detection or on command from the Management Workstation. To accomplish down loading, boot/load module 22 uses the Trivial File Transfer Protocol (TFTP). The protocol stack used for loading is TFTP/UDP/IP/ethernet over the LAN and TFTP/UDP/IP/SLIP over the serial line.

Device Driver 24 manages the network controller hardware so that Monitor 10 is able to read and write packets from the network and it manages the serial interface. It does so both for the purposes of monitoring traffic (promiscuous mode) and for the purposes of communicating with the Management Workstation and other devices on the network. The communication occurs through the network controller hardware of the physical network (e.g. Ethernet). The drivers for the LAN controller and serial line interface are used by the boot load module and the MTM. They provide access to the chips and isolate higher layers from the hardware specifics.

Test module 26 performs and reports results of physical layer tests (TDR, connectivity, . . .) under control of the Management Workstation. It provides traffic load information in response to user requests identifying the particular traffic data of interest. The load information is reported either as a percent of available bandwidth or as frame size(s) plus rate.

SNMP Agent 28 translates requests and information into the network management protocol being used to communicate with the Management Workstation, e.g., the Simple Network Management Protocol (SNMP).

11

Control Module 42 coordinates access to monitor control variables and performs actions necessary when these are altered. Among the monitor control variables which it handles are the following:

- set reset monitor—transfer control to reset logic;
- set time of day—modify monitor hardware clock and generate response to Management Workstation;
- get time of day—read monitor hardware clock and generate response to Workstation;
- set trap permit—send trap control ITM to EM and generate response to Workstation;
- get trap permit—generate response to Workstation;

Control module 42 also updates parse control records within STATS when invoked by the RTP (to be described) or during overload conditions so that higher layers of parsing are dropped until the overload situation is resolved. When overload is over it restores full parsing.

Timer 30 is invoked periodically to perform general housekeeping functions. It pulses the watchdog timer at appropriate intervals. It also takes care of internal time stamping and kicking off routines like the EM routine which periodically recalculates certain numbers within the statistical database (i.e., STATS).

Real Time Parser (RTP) 32 sees all frames on the network and it determines which protocols are being used and interprets the frames. The RTP includes a protocol parser and a state machine. The protocol parser parses a received frame in the "classical" manner, layer-by-layer, lowest layer first. The parsing is performed such that the statistical objects in STATS (i.e., the network parameters for which performance data is kept) are maintained. Which layers are to have statistics stored for them is determined by a parse control record that is stored in STATS (to be described later). As each layer is parsed, the RTP invokes the appropriate functions in the statistics module (STATS) to update those statistical objects which must be changed.

The state machine within RTP 32 is responsible for tracking state as appropriate to protocols and connections. It is responsible for maintaining and updating the connection oriented statistical elements in STATS. In order to track connection states and events, the RTP invokes a routine within the state machine. This routine determines the state of a connection based on past observed frames and keeps track of sequence numbers. It is the routine that determines if a connection is in data transfer state and if a retransmission has occurred. The objectives of the state machine are to keep a brief history of events, state transitions, and sequence numbers per connection; to detect data transfer state so that sequence tracking can begin; and to count inconsistencies but still maintain tracking while falling into an appropriate state (e.g. unknown).

RTP 32 also performs overload control by determining the number of frames awaiting processing and invoking control module 42 to update the parse control records so as to reduce the parsing depth when the number becomes too large.

Statistics Module (STATS) 36 is where Monitor 10 keeps information about the statistical objects it is charged with monitoring. A statistical object represents a network parameter for which performance information is gathered. This information is contained in an extended MIB (Management Information Base), which is updated by RTP 32 and EM 38.

STATS updates statistical objects in response to RTP invocation. There are at least four statistical object classes, namely, counters, timers, percentages (%), and meters. Each statistical object is implemented as appropriate to the object class to which it belongs. That is, each statistical object

12

behaves such that when invoked by RTP 32 it updates and then generates an alarm if its value meets a preset threshold. (Meets means that for a high threshold the value is equal to or greater than the threshold and for a low threshold the value is equal to or less than the threshold. Note that a single object may have both high and low thresholds.)

STATS 36 is responsible for the maintenance and initial analysis of the database. This includes coordinating access to the database variables, ensuring appropriate interlocks are applied and generating alarms when thresholds are crossed. Only STATS 36 is aware of the internal structure of the database, the rest of the system is not.

STATS 36 is also responsible for tracking events of interest in the form of various statistical reductions. Examples are counters, rate meters, and rate of change of rate meters. It initiates events based on particular statistics reaching configured limits, i.e., thresholds. The events are passed to the EM which sends a trap (i.e., an alarm) to the Management Workstation. The statistics within STATS 36 are readable from the Management Workstation on request.

STATS performs lookup on all addressing fields. It assigns new data structures to address field values not currently present. It performs any hashing for fast access to the database. More details will be presented later in this document.

Event Manager (EM) 38 extracts statistics from STATS and formats it in ways that allow the Workstation to understand it. It also examines the various statistics to see if their behavior warrants a notification to the Management Workstation. If so, it uses the SNMP Agent software to initiate such notifications.

If the Workstation asks for data, EM 38 gets the data from STATS and sends it to the Workstation. It also performs some level of analysis for statistical, accounting and alarm filtering and decides on further action (e.g. delivery to the Management Workstation). EM 38 is also responsible for controlling the delivery of events to the Management Workstation, e.g., it performs event filtering. The action to be taken on receipt of an event (e.g. threshold exceeded in STATS) is specified by the event action associated with the threshold. The event is used as an index to select the defined action (e.g. report to Workstation, run local routine xxxx, ignore). The action can be modified by commands from the Management Workstation (e.g., turn off an alarm) or by the control module in an overload situation. An update to the event action, however, does not affect events previously processed even if they are still waiting for transmission to the Management Workstation. Discarded events are counted as such by EM 38.

EM 38 also implements a throttle mechanism to limit the rate of delivery of alarms to the console based on configured limits. This prevents the rapid generation of multiple alarms. In essence, Monitor 10 is given a maximum frequency at which alarms may be sent to the Workstation. Although alarms in excess of the maximum frequency are discarded, a count is kept of the number of alarms that were discarded.

EM 38 invokes routines from the statistics module (STATS) to perform periodic updates such as rate calculations and threshold checks. It calculates time averages, e.g., average traffic by source stations, destination stations. EM 38 requests for access to monitor control variables are passed to the control module.

EM 38 checks whether asynchronous traps (i.e., alarms) to the Workstation are permitted before generating any.

EM 38 receives database update requests from the Management Workstation and invokes the statistics module (STATS) to process these.

Message Transport Module (MTM) 34, which is DRAM based, has two distinct but closely related functions. First, it is responsible for the conversion of Workstation commands and responses from the internal format used within Monitor 10 to the format used to communicate over the network. It isolates the rest of the system from the protocol used to communicate within Management Workstation. It translates between the internal representation of data and ASN.1 used for SNMP. It performs initial decoding of Workstation requests and directs the requests to appropriate modules for processing. It implements SNMP/UDP/IP/LLC or ETHERNET protocols for LAN and SNMP/UDP/IP/SLIP protocols for serial line. It receives network management commands from the Management Workstation and delivers these to the appropriate module for action. Alarms and responses destined for the Workstation are also directed via this module.

Second, MTM 34 is responsible for the delivery and reception of data to and from the Management Workstation using the protocol appropriate to the network. Primary and backup communication paths are provided transparently to the rest of the monitor modules (e.g. LAN and dial up link). It is capable of full duplex delivery of messages between the console and monitoring module. The messages carry event, configuration, test and statistics data.

Event Timing Module (ETM) 40 keeps track of the start time and end times of user specified transactions over the network. In essence, this module monitors the responsiveness of the network at any protocol or layer specified by the user.

Address Tracking Module 42 keeps track of the node name to node address bindings on networks which implement dynamic node addressing protocols.

Memory management for Monitor 10 is handled in accordance with following guidelines. The available memory is divided into four blocks during system initialization. One block includes receive frame buffers. They are used for receiving LAN traffic and for receiving secondary link traffic. These are organized as linked lists of fixed sized buffers. A second block includes system control message blocks. They are used for intertask messages within Monitor and are organized as a linked list of free blocks and multiple linked lists of in process intertask messages. A third block includes transmit buffers. They are used for creation and transmission of workstation alarms and responses and are organized as a linked list of fixed sized buffers. A fourth block is the statistics. This is allocated as a fixed size area at system initialization and managed by the statistics module during system operation.

Task Structure of Monitor;

The structure of the Monitor in terms of tasks and intertask messages is shown in FIG. 6. The rectangular blocks represent interrupt service routines, the ovals represent tasks and the circles represent input queues.

Each task in the system has a single input queue which it uses to receive all input. All inter-process communications take place via messages placed onto the input queue of the destination task. Each task waits on a (well known) input queue and processes events or inter-task messages (i.e., ITM's) as they are received. Each task returns to the kernel within an appropriate time period defined for each task (e.g. after processing a fixed number of events).

Interrupt service routines (ISR's) run on receipt of hardware generated interrupts. They invoke task level processing by sending an ITM to the input queue of the appropriate task.

The kernel scheduler acts as the base loop of the system and calls any runnable tasks as subroutines. The determination of whether a task is runnable is made from the input

queue, i.e., if this has an entry the task has work to perform. The scheduler scans the input queues for each task in a round robin fashion and invokes a task with input pending. Each task processes items from its input queue and returns to the scheduler within a defined period. The scheduler then continues the scan cycle of the input queues. This avoids any task locking out others by processing a continuously busy input queue. A task may be given an effectively higher priority by providing it with multiple entries in the scan table.

Database accesses are generally performed using access routines. This hides the internal structure of the database from other modules and also ensures that appropriate interlocks are applied to shared data.

The EM processes a single event from the input queue and then returns to the scheduler.

The MTM Xmit task processes a single event from its input queue and then returns control to the scheduler. The MTM Recv task processes events from the input queue until it is empty or a defined number (e.g. 10) events have been processed and then returns control to the scheduler.

The timer task processes a single event from the input queue and then returns control to the scheduler.

RTP continues to process frames until the input queue is empty or it has processed a defined number (e.g. 10) frames. It then returns to the scheduler.

The following sections contain a more detailed description of some of the above-identified software modules.

The Statistics Module (STATS):

The functions of the statistics module are:

-
- * to define statistics records;
 - * to allocate and initialize statistics records;
 - * to provide routines to lookup statistics records, e.g. lookup_id_addr;
 - * to provide routines to manipulate the statistics within the records, e.g. stats_age, stats_incr and stats_rate;
 - * to provide routines to free statistics records, e.g. stats_allocate and stats_deallocate
-

It provides these services to the Real Time Parser (RTP) module and to the Event Manager (EM) module.

STATS defines the database and it contains subroutines for updating the statistics which it keeps.

STATS contains the type definitions for all statistics records (e.g. DLL, IP, TCP statistics). It provides an initialization routine whose major function is to allocate statistics records at startup from cacheable memory. It provides lookup routines in order to get at the statistics. Each type of statistics record has its own lookup routine (e.g. lookup_ip_address) which returns a pointer to a statistics record of the appropriate type or NULL.

As a received frame is being parsed, statistics within statistics records need to be manipulated (e.g. incremented) to record relevant information about the frame. STATS provides the routines to manipulate those statistics. For example, there is a routine to update counters. After the counter is incremented/decremented and if there is a non-zero threshold associated with the counter, the internal routine compares its value to the threshold. If the threshold has been exceeded, the Event Manager is signaled in order to send a trap to the Workstation. Besides manipulating statistics, these routines, if necessary, signal the Event Manager via an Intertask Message (ITM) to send a trap to the Management Workstation.

The following is an example of some of the statistics records that are kept in STATS.

- o monitor statistics
- o mac statistics for segment
- o llc statistics for segment
- o statistics per ethernet/lsap type for segment
- o ip statistics for segment
- o icmp statistics for segment
- o tcp statistics for segment
- o udp statistics for segment
- o nfs statistics for segment
- o ftp control statistics for segment
- o ftp data statistics for segment
- o telnet statistics for segment
- o smtp statistics for segment
- o arp statistics for segment
- o statistics per mac address
- o statistics per ethernet type/lasp per mac address
- o statistics per ip address (includes icmp)
- o statistics per tcp socket
- o statistics per udp socket
- o statistics per nfs socket
- o statistics per ftp control socket
- o statistics per ftp data socket
- o statistics per telnet socket
- o statistics per smtp socket
- o arp statistics per ip address
- o statistics per mac address pair
- o statistics per ip pair (includes icmp)
- o statistics per tcp connection
- o statistics per udp pair
- o statistics per nfs pair
- o statistics per ftp control connection
- o statistics per ftp data connection
- o statistics per telnet connection
- o statistics per smtp connection
- o connection histories per udp and tcp socket

All statistics are organized similarly across protocol types. The details of the data structures for the DLL level are presented later.

As noted earlier, there are four statistical object classes (i.e., variables), namely, counts, rates, percentages (%), and meters. They are defined and implemented as follows.

A count is a continuously incrementing variable which rolls around to 0 on overflow. It may be reset on command from the user (or from software). A threshold may be applied to the count and will cause an alarm when the threshold count is reached. The threshold count fires each time the counter increments past the threshold value. For example, if the threshold is set to 5, alarms are generated when the count is 5, 10, 15,

A rate is essentially a first derivative of a count variable. The rate is calculated at a period appropriate to the variable. For each rate variable, a minimum, maximum and average value is maintained. Thresholds may be set on high values of the rate. The maximums and minimums may be reset on command. The threshold event is triggered each time the rate calculated is in the threshold region.

As commonly used, the % is calculated at a period appropriate to the variable. For each % variable a minimum, maximum and average value is maintained. A threshold may be set on high values of the %. The threshold event is triggered each time the % calculated is in the threshold region.

Finally, a meter is a variable which may take any discrete value within a defined range. The current value has no correlation to past or future values. A threshold may be set on a maximum and/or minimum value for a meter.

The rate and % fields of network event variables are updated differently than counter or meter fields in that they are calculated at fixed intervals rather than on receipt of data from the network.

Structures for statistics kept on a per address or per address pair basis are allocated at initialization time. There are several sizes for these structures. Structures of the same size are linked together in a free pool. As a new structure is needed, it is obtained from a free queue, initialized, and linked into an active list. Active lists are kept on a per statistics type basis.

As an address or address pair (e.g. mac, ip, tcp . . .) is seen, RTP code calls an appropriate lookup routine. The lookup routine scans active statistics structures to see if a structure has already been allocated for the statistics. Hashing algorithms are used in order to provide for efficient lookup. If no structure has been allocated, the lookup routine examines the appropriate parse control records to determine whether statistics should be kept, and, if so, it allocates a structure of the appropriate size, initializes it and links it into an active list.

Either the address of a structure or a NULL is returned by these routines. If NULL is returned, the RTP does not stop parsing, but it will not be allowed to store the statistics for which the structure was requested.

The RTP updates statistics within the data base as it runs. This is done via macros defined for the RTP. The macros call on internal routines which know how to manipulate the relevant statistic. If the pointer to the statistics structure is NULL, the internal routine will not be invoked.

The EM causes rates to be calculated. The STATS module supplies routines (e.g. stats_rate) which must be called by the EM in order to perform the rate calculations. It also calls subroutines to reformat the data in the database in order to present it to the Workstation (i.e., in response to a get from the Workstation).

The calculation algorithms for the rate and % fields of network event variables are as follows.

The following rates are calculated in units per second, at the indicated (approximate) intervals:

1. 10 second intervals:
e.g. DLL frame, byte, ethernet, 802.3, broadcast, multicast rates
2. 60 second intervals
e.g., all DLL error, ethertype/dsap rates all IP rates.
TCP packets, bytes, errors, retransmitted packets, retransmitted bytes, acks, rsts UDP packet, error, byte rates
FTP file transfer, byte transfer, error rates

For these rates, the new average replaces the previous value directly. Maximum and minimum values are retained until reset by the user.

The following rates are calculated in units per hour at the indicated time intervals:

1. 15 minute interval.
e.g., TCP—connection rate
Telnet connection rate
FTP session rate

The hourly rate is calculated from a sum of the last twelve 5 minute readings, as obtained from the buckets for the pertinent parameter. Each new reading replaces the oldest of the twelve values maintained. Maximum and minimum values are retained until reset by the user.

There are a number of other internal routines in STATS. For example, all statistical data collected by the Monitor is subject to age out. Thus, if no activity is seen for an address (or address pair) in the time period defined for age out, then the data is discarded and the space reclaimed so that it may be recycled. In this manner, the Monitor is able to use the memory for active elements rather than stale data. The user can select the age out times for the different components. The EM periodically kicks off the aging mechanism to perform this recycling of resources. STATS provides the routines which the EM calls, e.g. stats_age.

There are also routines in STATS to allocate and deallocate Statistics, e.g., stats_allocate and stats_deallocate. The allocate routine is called when stations and dialogs are picked up by the Network Monitor. The deallocate routine is called by the aging routines when a structure is to be recycled.

The Data Structures in STATS

The general structure of the database within STATS is illustrated by FIGS. 7a-c, which shows information that is maintained for the Data Link Layer (DLL) and its organization. A set of data structures is kept for each address associated with the layer. In this case there are three relevant addresses, namely a segment address, indicating which segment the node is on, a MAC address for the node on the segment, and an address which identifies the dialog occurring over that layer. The dialog address is the combination of the MAC addresses for the two nodes which make up the dialog. Thus, the overall data structure has three identifiable components: a segment address data structure (see FIG. 7a), a MAC address data structure (see FIG. 7b) and a dialog data structure (see FIG. 7c).

The segment address structure includes a doubly linked list 102 of segment address records 104, each one for a different segment address. Each segment address record 104 contains a forward and backward link (field 106) for forward and backward pointers to neighboring records and a hash link (field 108). In other words, the segment address records are accessed by either walking down the doubly linked list or by using a hashing mechanism to generate a pointer into the doubly linked list to the first record of a smaller hash linked list. Each record also contains the address of the segment (field 110) and a set of fields for other information. Among these are a flags field 112, a type field 114, a parse_control field 116, and an EM_control field 118. Flags field 112 contains a bit which indicates whether the identified address corresponds to the address of another Network Monitor. This field only has meaning in the MAC address record and not in the segment or dialog address record. Type field 114 identifies the MIB group which applies to this address. Parse control field 116 is a bit mask which indicates what subgroups of statistics from the identified MIB group are maintained, if any. Flags field 112, type field 114 and parse control field 116 make up what is referred to as the parse control record for this MAC address. The Network Monitor uses a default value for parse control field 116 upon initialization or whenever a new node is detected. The default value turns off all statistics gathering. The statistics gathering for any particular address may subsequently be turned on by the Workstation through a Network Monitor control command that sets the appropriate bits of the parse control field to one.

EM_control field 118 identifies the subgroups of statistics within the MIB group that have changed since the EM last serviced the database to update rates and other variables. This field is used by the EM to identify those parts of STATS which must be updated or for which recalculations must be performed when the EM next services STAT.

Each segment address record 104 also contains three fields for time related information. There is a start_time field 120 for the time that is used to perform some of the rate calculations for the underlying statistics; a first_seen field 122 for the time at which the Network Monitor first saw the communication; and a last_seen field 124 for the time at which the last communication was seen. The last_seen time is used to age out the data structure if no activity is seen on the segment after a preselected period of time elapses. The first_seen time is a statistic which may be of interest to the network manager and is thus retrievable by the Management Workstation for display.

Finally, each segment address record includes a stats_pointer field 126 for a pointer to a DLL segment statistics data structure 130 which contains all of the statistics that are maintained for the segment address. If the bits in parse_control field 116 are all set to off, indicating that no statistics are to be maintained for the address, then the pointer in stats_pointer field 126 is a null pointer.

The list of events shown in data structure 130 of FIG. 7a illustrates the type of data that is collected for this address when the parse control field bits are set to on. Some of the entries in DLL segment statistics data structure 130 are pointers to buckets for historical data. In the case where buckets are maintained, there are twelve buckets each of which represents a time period of five minutes duration and each of which generally contains two items of information, namely, a count for the corresponding five minute time period and a MAX rate for that time period. MAX rate records any spikes which have occurred during the period and which the user may not have observed because he was not viewing that particular statistic at the time.

At the end of DLL segment statistics data structure 130, there is a protocol_Q pointer 132 to a linked list 134 of protocol statistics records 136 identifying all of the protocols which have been detected running on top of the DLL layer for the segment. Each record 136 includes a link 138 to the next record in the list, the identity of the protocol (field 140), a frames count for the number of frames detected for the identified protocol (field 142); and a frame rate (field 144).

The MAC address data structure is organized in a similar manner to that of the segment data structure (see FIG. 7b). There is a doubly linked list 146 of MAC address records 148, each of which contains the same type of information as is stored in DLL segment address records 104. A pointer 150 at the end of each MAC address record 148 points to a DLL address statistics data structure 152, which like the DLL segment address data structure 130, contains fields for all of the statistics that are gathered for that DLL MAC address. Examples of the particular statistics are shown in FIG. 7b.

At the end of DLL address statistics data structure 152, there are two pointer fields 152 and 154, one for a pointer to a record 158 in a dialog link queue 160, and the other for a pointer to a linked list 162 of protocol statistics records 164. Each dialog link queue entry 158 contains a pointer to the next entry (field 168) in the queue and it contains a dialog_addr pointer 170 which points to an entry in the DLL dialog queue which involves the MAC address. (see FIG. 7c). Protocol statistics records 164 have the same structure and contain the same categories of information as their counterparts hanging off of DLL segment statistics data structure 130.

The above-described design is repeated in the DLL dialog data structures. That is, dialog record 172 includes the same categories of information as its counterpart in the DLL segment address data structure and the MAC address data structure. The address field 174 contains the addresses of

both ends of the dialog concatenated together to form a single address. The first and second addresses within the single address are arbitrarily designated nodes 1 and 2, respectively. In the stats_pointer field 176 there is a pointer to a dialog statistics data structure 178 containing the relevant statistics for the dialog. The entries in the first two fields in this data structure (i.e., fields 180 and 182) are designated protocol entries and protocols. Protocol entries is the number of different protocols which have been seen between the two MAC addresses. The protocols that have been seen are enumerated in the protocols field 182.

DLL dialog statistics data structure 178, illustrated by FIG. 7c, includes several additional fields of information which only appear in these structures for dialogs for which state information can be kept (e.g. TCP connection). The additional fields identify the transport protocol (e.g., TCP) (field 184) and the application which is running on top of that protocol (field 186). They also include the identity of the initiator of the connection (field 188), the state of the connection (field 190) and the reason that the connection was closed, when it is closed (field 192). Finally, they also include a state_pointer (field 194) which points to a history data structure that will be described in greater detail later. Suffice it to say, that the history data structure contains a short history of events and states for each end of the dialog. The state machine uses the information contained in the history data structure to loosely determine what the state of each of the end nodes is throughout the course of the connection. The qualifier "loosely" is used because the state machine does not closely shadow the state of the connection and thus is capable of recovering from loss of state due to lost packets or missed communications.

The above-described structures and organization are used for all layers and all protocols within STATS.

Real Time Parser (RTP)

The RTP runs as an application task. It is scheduled by the Real Time Kernel scheduler when received frames are detected. The RTP parses the frames and causes statistics, state tracking, and tracing operations to be performed.

The functions of the RTP are:

- * obtain frames from the RTP Input Queue;
- * parse the frames;
- * maintain statistics using routines supplied by the STATS module;
- * maintain protocol state information;
- * notify the MTM via an ITM if a frame has been received with the Network Monitor's address as the destination address; and
- * notify the EM via an ITM if a frame has been received with any Network Monitor's address as the source address.

The design of the RTP is straightforward. It is a collection of routines which perform protocol parsing. The RTP interfaces to the Real Time Kernel in order to perform RTP initialization, to be scheduled in order to parse frames, to free frames, to obtain and send an ITM to another task; and to report fatal errors. The RTP is invoked by the scheduler when there is at least one frame to parse. The appropriate parse routines are executed per frame. Each parse routine invokes the next level parse routine or decides that parsing is done. Termination of the parse occurs on an error or when the frame has been completely parsed.

Each parse routine is a separately compilable module. In general, parse routines share very little data. Each knows where to begin parsing in the frame and the length of the data remaining in the frame.

The following is a list of the parse routines that are available within RTP for parsing the different protocols at the various layers.

Data Link Layer Parse—rtp_dll_parse:

This routine handles Ethernet, IEEE 802.3, IEEE 802.2, and SNAP. See RFC 1010, Assigned Numbers for a description of SNAP (Subnetwork Access Protocol).

Address Resolution Protocol Parse—rtp_arp_parse

ARP is parsed as specified in RFC 826.

Internet Protocol Parse—rtp_ip_parse

IP Version 4 is parsed as specified in RFC 791 as amended by RFC 950, RFC 919, and RFC 922.

Internet Control Message Protocol Parse—rtp_icmp_parse

ICMP is parsed as specified in RFC 792.

Unit Data Protocol Parse—rtp_udp_parse

UDP is parsed as specified in RFC 768.

Transmission Control Protocol Parse—rtp_tcp_parse

TCP is parsed as specified in RFC 793.

Simple Mail Transfer Protocol Parse—rtp_smtp_parse

SMTP is parsed as specified in RFC 821.

File Transfer Protocol Parse—rtp_ftp_parse

FTP is parsed as specified in RFC 959.

Telnet Protocol Parse—rtp_telnet_parse

The Telnet protocol is parsed as specified in RFC 854.

Network File System Protocol Parse—rtp_nfs_parse

The NFS protocol is parsed as specified in RFC 1094.

The RTP calls routines supplied by STATS to look up data structures. By calling these lookup routines, global pointers to data structures are set up. Following are examples of the pointers to statistics data structures that are set up when parse routines call Statistics module lookup routines.

```
mac_segment, mac_dst_segment, mac_this_segment,
mac_src, mac_dst, mac_dialog
ip_src_segment, ip_dst_segment, ip_this_segment,
ip_src, ip_dst, ip_dialog
tcp_src_segment, tcp_dst_segment, tcp_this_
segment,
tcp_src, tcp_dst, tcp_src_socket, tcp_dst_socket,
tcp_connection
```

The mac_src and mac_dst routines return pointers to the data structures within STATS for the source MAC address and the destination MAC address, respectively. The lookup_mac_dialog routine returns a pointer to the data structure within STATS for the dialog between the two nodes on the MAC layer. The other STATS routines supply similar pointers for data structures relevant to other protocols.

The RTP routines are aware of the names of the statistics that must be manipulated within the data base (e.g. frames, bytes) but are not aware of the structure of the data. When a statistic is to be manipulated, the RTP routine invokes a macro which manipulates the appropriate statistics in data structures. The macros use the global pointers which were set up during the lookup process described above.

After a frame has been parsed (whether the parse was successful or not), the RTP routine examines the destination mac and ip addresses. If either of the addresses is that of the Network Monitor, RTP obtains a low priority ITM, initializes it, and sends the ITM to the MTM task. One of the fields of the ITM contains the address of the buffer containing the frame.

The RTP must hand some received frames to the EM in order to accomplish the autotopology function (described later). After a frame has been parsed (whether the parse was successful or not), the RTP routine examines the source mac

and ip addresses. If either of the addresses is that of another Network Monitor, RTP obtains a low priority ITM, initializes it and sends the ITM to the EM task. The address data structure (in particular, the flags field of the parse control record) within STATS for the MAC or the IP address indicates whether the source address is that of another Network Monitor. One of the fields of the ITM contains the address of the buffer containing the frame.

The RTP receives traffic frames from the network for analysis. RTP operation may be modified by sending control messages to the Monitor. RTP first parses these messages, then detects that the messages are destined for the Monitor and passes them to the MTM task. Parameters which affect RTP operation may be changed by such control messages.

The general operation of the RTP upon receipt of a traffic frame is as follows:

```

Get next frame from input queue
get address records for these stations
For each level of active parsing
{
  get pointer to start of protocol header
  call layer parse routine
  determine protocol at next level
  set pointer to start of next layer protocol
}end of frame parsing
if this is a monitor command add to MTM input queue
if this frame is from another monitor, pass to EM
check for overload—if yes tell control

```

The State Machine:

In the described embodiment, the state machine determines and keeps state for both addresses of all TCP connections. TCP is a connection oriented transport protocol, and TCP clearly defines the connection in terms of states of the connection. There are other protocols which do not explicitly define the communication in terms of state, e.g. connectionless protocols such as NFS. Nevertheless, even in the connectionless protocols there is implicitly the concept of state because there is an expected order to the events which will occur during the course of the communication. That is, at the very least, one can identify a beginning and an end of the communication, and usually some sequence of events which will occur during the course of the communication. Thus, even though the described embodiment involves a connection oriented protocol, the principles are applicable to many connectionless protocols or for that matter any protocol for which one can identify a beginning and an end to the communication under that protocol.

Whenever a TCP packet is detected, the RTP parses the information for that layer to identify the event associated with that packet. It then passes the identified event along with the dialog identifier to the state machine. For each address of the two parties to the communication, the state machine determines what the current state of the node is. The code within the state machine determines the state of a connection based upon a set of rules that are illustrated by the event/state table shown in FIG. 8.

The interpretation of the event/state table is as follows. The top row of the table identifies the six possible states of a TCP connection. These states are not the states defined in the TCP protocol specification. The left most column identifies the eight events which may occur during the course of a connection. Within the table is an array of boxes, each of which sits at the intersection of a particular event/state combination. Each box specifies the actions taken by the state machine if the identified event occurs while the con-

nection is in the identified state. When the state machine receives a new event, it may perform three types of action. It may change the recorded state for the node. The state to which the node is changed is specified by the S="STATE" entry located at the top of the box. It may increment or decrement the appropriate counters to record the information relevant to that event's occurrence. (In the table, incrementing and decrementing are signified by the ++ and the -- symbols, respectively, located after the identity of the variable being updated.) or the state machine may take other actions such as those specified in the table as start close timer, Look_for_Data_State, or Look_at_History (to be described shortly). The particular actions which the state machine takes are specified in each box. An empty box indicates that no action is taken for that particular event/state combination. Note, however, that the occurrence of an event is also likely to have caused the update of statistics within STATS, if not by the state machine, then by some other part of the RTP. Also note that it may be desirable to have the state machine record other events, in which case the state table would be modified to identify those other actions.

Two events appearing on the table deserve further explanation, namely, close timer expires and inactivity timer expires. The close timer, which is specified by TCP, is started at the end of a connection and it establishes a period during which any old packets for the connection which are received are thrown away (i.e., ignored). The inactivity timer is not specified by TCP but rather is part of the Network Monitor's resource management functions. Since keeping statistics for dialogs (especially old dialogs) consumes resources, it is desirable to recycle resources for a dialog if no activity has been seen for some period of time. The inactivity timer provides the mechanism for accomplishing this. It is restarted each time an event for the connection is received. If the inactivity timer expires (i.e., if no event is received before the timer period ends), the connection is assumed to have gone inactive and all of the resources associated with the dialog are recycled. This involves freeing them up for use by other dialogs.

The other states and events within the table differ from but are consistent with the definitions provided by TCP and should be self evident in view of that protocol specification.

The event/state table can be read as follows. Assume, for example, that node 1 is in DATA state and the RTP receives another packet from node 1 which it determines to be a TCP FIN packet. According to the entry in the table at the intersection of FIN/DATA (i.e., event/state), the state machine sets the state of the connection for node 1 to CLOSING, it decrements the active connections counter and it starts the close timer. When the close timer expires, assuming no other events over that connection have occurred, the state machine sets node 1's state to CLOSED and it starts the inactivity timer. If the RTP sends another SYN packet to reinitiate a new connection before the inactive timer expires, the state machine sets node 1's state to CONNECTING (see the SYN/CLOSED entry) and it increments an after close counter.

When a connection is first seen, the Network Monitor sets the state of both ends of the connection to UNKNOWN state. If some number of data and acknowledgment frames are seen from both connection ends, the states of the connection ends may be promoted to DATA state. The connection history is searched to make this determination as will be described shortly.

Referring to FIGS. 9a-b, within STATS there is a history data structure 200 which the state machine uses to remember the current state of the connection, the state of each of the

nodes participating in the connection and a short history of state related information. History data structure 200 is identified by a state_pointer found at the end of the associated dialog statistics data structure in STATS (see FIG. 7c). Within history data structure 200, the state machine records the current state of node 1 (field 202), the current state of node 2 (field 206) and other data relating to the corresponding node (fields 204 and 208). The other data includes, for example, the window size for the receive and transmit communications, the last detected sequence numbers for the data and acknowledgment frames, and other data transfer information.

History data structure 200 also includes a history table (field 212) for storing a short history of events which have occurred over the connection and it includes an index to the next entry within the history table for storing the information about the next received event (field 210). The history table is implemented as a circular buffer which includes sufficient memory to store, for example, 16 records. Each record, shown in FIG. 9b, stores the state of the node when the event was detected (field 218), the event which was detected (i.e., received) (field 220), the data field length (field 222), the sequence number (field 224), the acknowledgment sequence number (field 226) and the identity of the initiator of the event, i.e., either node 1 or node 2 or 0 if neither (field 228).

Though the Network Monitor operates in a promiscuous mode, it may occasionally fail to detect or it may, due to overload, lose a packet within a communication. If this occurs the state machine may not be able to accurately determine the state of the connection upon receipt of the next event. The problem is evidenced by the fact that the next event is not what was expected. When this occurs, the state machine tries to recover state by relying on state history information stored in the history table in field 212 to deduce what the state is. To deduce the current state from historical information, the state machine uses one of the two previously mentioned routines, namely, Look_for_Data_State and Look_at_History.

Referring to FIG. 10, Look_for_Data_State routine 230 searches back through the history one record at a time until it finds evidence that the current state is DATA state or until it reaches the end of the circular buffer (step 232). Routine 230 detects the existence of DATA state by determining whether node 1 and node 2 each have had at least two data events or two acknowledgment combinations with no intervening connect, disconnect or abort events (step 234). If such a sequence of events is found within the history, routine 230 enters both node 1 and node 2 into DATA state (step 236), it increments the active connections counter (step 238) and then it calls a Look_for_Initiator routine to look for the initiator of the connection (step 240). If such a pattern of events is not found within the history, routine 230 returns without changing the state for the node (step 242).

As shown in FIG. 11, Look_for_Initiator routine 240 also searches back through the history to detect a telltale event pattern which identifies the actual initiator of the connection (step 244). More specifically, routine 240 determines whether nodes 1 and 2 each sent connect-related packets. If they did, routine 240 identifies the initiator as the first node to send a connect-related packet (step 246). If the search is not successful, the identity of the connection initiator remains unknown (step 248).

The Look_at_History routine is called to check back through the history to determine whether data transmissions have been repeated. In the case of retransmissions, the routine calls a Look_for_Retransmission routine 250, the operation of which is shown in FIG. 12. Routine 250

searches back through the history (step 252) and checks whether the same initiator node has sent data twice (step 254). It detects this by comparing the current sequence number of the packet as provided by the RTP with the sequence numbers of data packets that were previously sent as reported in the history table. If a retransmission is spotted, the retransmission counter in the dialog statistics data structure of STATS is incremented (step 256). If the sequence number is not found within the history table, indicating that the received packet does not represent a retransmission, the retransmission counter is not incremented (step 258).

Other statistics such as Window probes and keep alives may also be detected by looking at the received frame, data transfer variables, and, if necessary, the history.

Even if frames are missed by the Network Monitor, because it is not directly "shadowing" the connection, the Network Monitor still keeps useful statistics about the connection. If inconsistencies are detected the Network Monitor counts them and, where appropriate, drops back to UNKNOWN state. Then, the Network Monitor waits for the connection to stabilize or deteriorate so that it can again determine the appropriate state based upon the history table. Principal Transactions of Network Monitor Modules:

The transactions which represent the major portion of the processing load within the Monitor, include monitoring, actions on threshold alarms, processing database get/set requests from the Management Workstation, and processing monitor control requests from the Management Workstation. Each of these mechanisms will now be briefly described.

Monitoring involves the message sequence shown in FIG. 13. In that figure, as in the other figures involving message sequences, the numbers under the heading SEQ. identify the major steps in the sequence. The following steps occur:

1. ISR puts Received traffic frame ITM on RTP input queue
2. request address of pertinent data structure from STATS (get parse control record for this station)
3. pass pointer to RTP
4. update statistical objects by call to statistical update routine in STATS using pointer to pertinent data structure
5. parse completed—release buffers

The major steps which follow a statistics threshold event (i.e., an alarm event) are shown in FIG. 14. The steps are as follows:

1. statistical object update causes threshold alarm
2. STATS generates threshold event ITM to event manager (EM)
3. look up appropriate action for this event
4. perform local event processing
5. generate network alarm ITM to MTM Xmit (if required)
6. format network alarm trap for Workstation from event manager data
7. send alarm to Workstation

The major steps in processing of a database update request (i.e., a get/set request) from the Management Workstation are shown in FIG. 15. The steps are as follows:

1. LAN ISR receives frame from network and passes it to RTP for parsing
2. RTP parses frame as for any other traffic on segment.
3. RTP detects frame is for monitor and sends received Workstation message over LAN ITM to MTM Recv.
4. MTM Recv processes protocol stack.

5. MTM Recv sends database update request ITM to EM.
6. EM calls STATS to do database read or database write with appropriate IMPB
7. STATS performs database access and returns response to EM.
8. EM encodes response to Workstation and sends database update response ITM to MTM Xmit
9. MTM Xmit transmits.

The major steps in processing of a monitor control request from the Management Workstation are shown in FIG. 16. The steps are as follows:

1. Lan ISR receives frame from network and passes received frame ITM to RTP for parsing.
2. RTP parses frame as for any other traffic on segment.
3. RTP detects frame is for monitor and sends received workstation message over LAN ITM to MTM Recv.
4. MTM Recv processes protocol stack and decodes workstation command.
5. MTM Recv sends request ITM to EM.
6. EM calls Control with monitor control IMPB.
7. Control performs requested operation and generates response to EM.
8. EM sends database update response ITM to MTM Xmit.
9. MTM Xmit encodes response to Workstation and transmits.

The Monitor/Workstation Interface:

The interface between the Monitor and the Management Workstation is based on the SNMP definition (RFC 1089 SNMP; RFC 1065 SMI; RFC 1066 SNMP MIB—Note: RFC means Request for Comments). All five SNMP PDU types are supported:

```
get-request
get-next-request
get-response
set-request
trap
```

The SNMP MIB extensions are designed such that where possible a user request for data maps to a single complex MIB object. In this manner, the get-request is simple and concise to create, and the response should contain all the data necessary to build the screen. Thus, if the user requests the IP statistics for a segment this maps to an IP Segment Group.

The data in the Monitor is keyed by addresses (MAC, IP) and port numbers (telnet, FTP). The user may wish to relate his data to physical nodes entered into the network map. The mapping of addresses to physical nodes is controlled by the user (with support from the Management Workstation system where possible) and the Workstation retains this information so that when a user requests data for node 'Joe' the Workstation asks the Monitor for the data for the appropriate address(es). The node to address mapping need not be one to one.

Loading and dumping of monitors uses TFTP (Trivial File Transfer Protocol). This operates over UDP as does SNMP. The Monitor to Workstation interface follows the SNMP philosophy of operating primarily in a polled mode. The Workstation acts as the master and polls the Monitor slaves for data on a regular (configurable) basis.

The information communicated by the SNMP is represented according to that subset of ASN.1 (ISO 8824 Specification of ASN.1) defined in the Internet standard Structure of Management Information (SMI—RFC 1065). The subset

of the standard Management Information Base (MIB) (RFC 1066 SNMP MIB) which is supported by the Workstation is defined in Appendix III. The added value provided by the Workstation is encoded as enterprise specific extensions to the MIB as defined in Appendix IV. The format for these extensions follows the SMI recommendations for object identifiers so that the Workstation extensions fall in the subtree 1.3.6.1.4.1.x.1. where x is an enterprise specific node identifier assigned by the IAB.

Appendix V is a summary of the network variables for which data is collected by the Monitor for the extended MIB and which can be retrieved by the Workstation. The summary includes short descriptions of the meaning and significance of the variables, where appropriate.

The Management Workstation:

The Management Workstation is a SUN Sparcstation (also referred to as a Sun) available from Sun Microsystems, Inc. It is running the Sun flavor of Unix and uses the Open Look Graphical User Interface (GUI) and the SunNet Manager as the base system. The options required are those to run SunNet Manager with some additional disk storage requirement.

The network is represented by a logical map illustrating the network components and the relationships between them, as shown in FIG. 17. A hierarchical network map is supported with navigation through the layers of the hierarchy, as provided by SNM. The Management Workstation determines the topology of the network and informs the user of the network objects and their connectivity so that he can create a network map. To assist with the map creation process, the Management Workstation attempts to determine the stations connected to each LAN segment to which a Monitor is attached. Automatic determination of segment topology by detecting stations is performed using the autotopology algorithms as described in copending U.S. patent application Ser. No. 07/641,156, entitled "Automatic Topology Monitor for Multi-Segment Local Area Network" filed on Jan. 14, 1991 now U.S. Pat. No. 5,546,540 (Attorney Docket No. 13283-NE.APP), incorporated herein by reference.

In normal operation, each station in the network is monitored by a single Monitor that is located on its local segment. The initial determination of the Monitor responsible for a station is based on the results of the autotopology mechanism. The user may override this initial default if required.

The user is informed of new stations appearing on any segment in the network via the alarm mechanism. As for other alarms, the user may select whether stations appearing on and disappearing from the network segment generate alarms and may modify the times used in the aging algorithms. When a new node alarm occurs, the user must add the new alarm to the map using the SNM tools. In this manner, the SNM system becomes aware of the nodes.

The sequence of events following the detection of a new node is:

1. the location of the node is determined automatically for the user.
2. the Monitor generates an alarm for the user indicating the new node and providing some or all of the following information:
 - mac address of node
 - ip address of node
 - segment that the node is believed to be located on
 - Monitor to be responsible for the node
3. the user must select the segment and add the node manually using the SNM editor
4. The update to the SNM database will be detected and the file reread. The Workstation database is recon-

structed and the parse control records for the Monitors updated if required.

5. The Monitor responsible for the new node has its parse control record updated via SNMP set request(s).

An internal record of new nodes is required for the autotopology. When a new node is reported by a Network Monitor, the Management Workstation needs to have the previous location information in order to know which Network Monitors to involve in autotopology. For example, two nodes with the same IP address may exist in separate segments of the network. The history makes possible the correlation of the addresses and it makes possible duplicate address detection.

Before a new Monitor can communicate with the Management Workstation via SNMP it needs to be added to the SNM system files. As the SNM files are cached in the database, the file must be updated and the SNM system forced to reread it.

Thus, on the detection of a new Monitor the following events need to occur in order to add the Monitor to the Workstation:

1. The Monitor issues, a trap to the Management Workstation software and requests code to be loaded from the Sun Microsystems boot/load server.
2. The code load fails as the Monitor is not known to the unix networking software at this time.
3. The Workstation confirms that the new Monitor does not exceed the configured system limits (e.g. 5 Monitors per Workstation) and terminates the initialization sequence if limits are exceeded. An alarm is issued to the user indicating the presence of the new Monitor and whether it can be supported.
4. The user adds the Monitor to the SNMPHOSTS file of the SNM system, to the etc/hosts file of the Unix networking system and to the SNM map.
5. When the files have been updated the user resets the Monitor using the set tool (described later).
6. The Monitor again issues a trap to the Management Workstation software and requests code to be loaded from the Sun boot/load server.
7. The code load takes place and the Monitor issues a trap requesting data from the Management Workstation.
8. The Monitor data is issued using SNMP set requests. Note that on receiving the set request, the SNMP proxy rereads in the (updated) SNMPHOSTS file which now includes the new Monitor. Also note that the SNMP hosts file need only contain the Monitors, not the entire list of nodes in the system.
9. On completion of the set request(s) the Monitor run command is issued by the Workstation to bring the Monitor on line.

The user is responsible for entering data into the SNM database manually. During operation, the Workstation monitors the file write date for the SNM database. When this is different from the last date read, the SNM database is reread and the Workstation database reconstructed. In this manner, user updates to the SNM database are incorporated into the Workstation database as quickly as possible without need for the user to take any action.

When the Workstation is loaded, the database is created from the data in the SNM file system (which the user has possibly updated). This data is checked for consistency and for conformance to the limits imposed by the Workstation at this time and a warning is generated to the user if any problems are seen. If the data errors are minor the system

continues operation; if they are fatal the user is asked to correct them and Workstation operation terminates.

The monitoring functions of the Management Workstation are provided as an extension to the SNM system. They consist of additional display tools (i.e., summary tool, values tool, and set tool) which the user invokes to access the Monitor options and a Workstation event log in which all alarms are recorded.

As a result of the monitoring process, the Monitor makes a large number of statistics available to the operator. These are available for examination via the Workstation tools that are provided. In addition, the Monitor statistics (or a selected subset thereof) can be made visible to any SNMP manager by providing it with knowledge of the extended MIB. A description of the statistics maintained are described elsewhere.

Network event statistics are maintained on a per network, per segment and per node basis. Within a node, statistics are maintained on a per address (as appropriate to the protocol layer—IP address, port number, . . .) and per connection basis. Per network statistics are always derived by the Workstation from the per segment variables maintained by the Monitors. Subsets of the basic statistics are maintained on a node to node and segment to segment basis.

If the user requests displays of segment to segment traffic, the Workstation calculates this data as follows. The inter segment traffic is derived from the node to node statistics for the intersecting set of nodes. Thus, if segment A has nodes 1, 2, and 3 and segment B has nodes 20, 21, and 22, then summing the node to node traffic for

```
1-->20,21,22
2-->20,21,22
3-->20,21,22
```

produces the required result. On-LAN/off-LAN traffic for segments is calculated by a simply summing node to node traffic for all stations on the LAN and then subtracting this from total segment counts.

Alarms are reported to the user in the following ways:

1. Alarms received are logged in a Workstation log.
2. The node which the alarm relates to is highlighted on the map.
3. The node status change is propagated up through the (map) hierarchy to support the case where the node is not visible on the screen. This is as provided by SNM.

Summary Tool

After the user has selected an object from the map and invokes the display tools, the summary tool generates the user's initial screen at the Management Workstation. It presents a set of statistical data selected to give an overview of the operational status of the object (e.g., a selected node or segment). The Workstation polls the Monitor for the data required by the Summary Tool display screens.

The Summary Tool displays a basic summary tool screen such as is shown in FIG. 18. The summary tool screen has three panels, namely, a control panel 602, a values panel 604, and a dialogs panel 606. The control panel includes the indicated mouse activated buttons. The functions of each of the buttons is as follows. The file button invokes a traditional file menu. The view button invokes a view menu which allows the user to modify or tailor the visual properties of the tool. The properties button invokes a properties menu containing choices for viewing and sometimes modifying the properties of objects. The tools button invokes a tools menu which provides access to the other Workstation tools, e.g. Values Tool.

The Update Interval field allows the user to specify the frequency at which the displayed statistics are updated by

polling the Monitor. The Update Once button enables the user to retrieve a single screen update. When the Update Once button is invoked not only is the screen updated but the update interval is automatically set to "none".

The type field enables the user to specify the type of network objects on which to operate, i.e., segment or node.

The name button invokes a pop up menu containing an alphabetical list of all network objects of the type selected and apply and reset buttons. The required name can then be selected from the (scrolling) list and it will be entered in the name field of the summary tool when the apply button is invoked. Alternatively, the user may enter the name directly in the summary tool name field.

The protocol button invokes a pop up menu which provides an exclusive set of protocol layers which the user may select. Selection of a layer copies the layer name into the displayed field of the summary tool when the apply operation is invoked. An example of a protocol selection menu is shown in FIG. 19. It displays the available protocols in the form of a protocol tree with multiple protocol families. The protocol selection is two dimensional. That is, the user first selects the protocol family and then the particular layer within that family.

As indicated by the protocol trees shown in FIG. 19, the capabilities of the Monitor can be readily extended to handle other protocol families. The particular ones which are implemented depend upon the needs of the particular network environment in which the Monitor will operate.

The user invokes the apply button to indicate that the selection process is complete and the type, name, protocol, etc. should be applied. This then updates the screen using the new parameter set that the user selected. The reset button is used to undo the selections and restore them to their values at the last apply operation.

The set of statistics for the selected parameter set is displayed in values panel 604. The members of the sets differ depending upon, for example, what protocol was selected. FIGS. 20a-g present examples of the types of statistical variables which are displayed for the DLL, IP, UDP, TCP, ICMP, NFS, and ARP/RARP protocols, respectively. The meaning of the values display fields are described in Appendix I, attached hereto.

Dialogs panel 606 contains a display of the connection statistics for all protocols for a selected node. Within the Management Workstation, connection lists are maintained per node, per supported protocol. When connections are displayed, they are sorted on "Last Seen" with the most current displayed first. A single list returned from the Monitor contains all current connection. For TCP, however, each connection also contains a state and TCP connections are displayed as Past and Present based upon the returned state of the connection. For certain dialogs, such as TCP and NFS over UDP, there is an associated direction to the dialog, i.e., from the initiator (source) to the receiver (sink). For these dialogs, the direction is identified in a DIR. field. A sample of information that is displayed in dialogs panel 606 is presented in FIG. 21 for current connections.

Values Tool

The values tool provides the user with the ability to look at the statistical database for a network object in detail. When the user invokes this tool, he may select a basic data screen containing a rate values panel 620, a count values panel 622 and a protocols seen panel 626, as shown in FIG. 22, or he may select a traffic matrix screen 628, as illustrated in FIG. 23.

In rate values and count values panels 620 and 622, value tools presents the monitored rate and count statistics,

respectively, for a selected protocol. The parameters which are displayed for the different protocols (i.e., different groups) are listed in Appendix II. In general, a data element that is being displayed for a node shows up in three rows, namely, a total for the data element, the number into the data element, and the number out of the data element. Any exceptions to this are identified in Appendix II. Data elements that are displayed for segments, are presented as totals only, with no distinction between Rx and Tx.

When invoked the Values Tool displays a primary screen to the user. The primary screen contains what is considered to be the most significant information for the selected object. The user can view other information for the object (i.e., the statistics for the other parameters) by scrolling down.

The displayed information for the count values and rate values panels 620 and 622 includes the following. An alarm field reports whether an alarm is currently active for this item. It displays as "+" if active alarm is present. A Current Value/Rate field reports the current rate or the value of the counter used to generate threshold alarms for this item. This is reset following each threshold trigger and thus gives an idea of how close to an alarm threshold the variable is. A Typical Value field reports what this item could be expected to read in a "normal" operating situation. This field is filled in for those items where this is predictable and useful. It is maintained in the Workstation database and is modifiable by the user using the set tool. An Accumulated Count field reports the current accumulated value of the item or the current rate. A Max Value field reports the highest value recently seen for the item. This value is reset at intervals defined by a user adjustable parameter (default 30 minutes). This is not a rolling cycle but rather represents the highest value since it was reset which may be from 1 to 30 minutes ago (for a rest period of 30 minutes). It is used only for rates. A Min Value field reports the lowest value recently seen for the item. This operates in the same manner as Max Value field and is used only for rates.

A Percent (%) field reports only for the following variables:

```

off seg counts:
  100 (in count / total off seg count)
  100 (out count / total off seg count)
  100 (transit count / total off seg count)
  100 (local count / total off seg count)
off seg rates
  100 (transit rate / total off seg rate), etc.
protocols
  100 (frame rate this protocol / total frame
rate)

```

On the right half of the basic display, there the following additional fields: a High Threshold field and a Sample period for rates field.

Set Tool

The set tool provides the user with the ability to modify the parameters controlling the operation of the Monitors and the Management Workstation. These parameters affect both user interface displays and the actual operation of the Monitors. The parameters which can be operated on by the set tool can be divided into the following categories: alarm thresholds, monitoring control, segment Monitor administration, and typical values.

The monitoring control variables specify the actions of the segment Monitors and each Monitor can have a distinct set of control variables (e.g., the parse control records that are described elsewhere). The user is able to define those

nodes, segments, dialogs and protocols in which he is interested so as to make the best use of memory space available for data storage. This mechanism allows for load sharing, where multiple Monitors on the same segment can divide up the total number of network objects which are to be monitored so that no duplication of effort between them takes place.

The monitor administration variables allow the user to modify the operation of the segment Monitor in a more direct manner than the monitoring control variables. Using the set tool, the user can perform those operations such as reset, time changes etc. which are normally the prerogative of a system administrator.

Note that the above descriptions of the tools available through the Management Workstation are not meant to imply that other choices may not be made regarding the particular information which is displayed and the manner in which it is displayed.

Adaptively Setting Network Monitor Thresholds:

The Workstation sets the thresholds in the Network Monitor based upon the performance of the system as observed over an extended period of time. That is, the Workstation periodically samples the output of the Network Monitors and assembles a model of a normally functioning network. Then, the Workstation sets the thresholds in the Network Monitors based upon that model. If the observation period is chosen to be long enough and since the model represents the "average" of the network performance over the observation period, temporary undesired deviations from normal behavior are smoothed out over time and model tends to accurately reflect normal network behavior.

Referring the FIG. 24, the details of the training procedure for adaptively setting the Network Monitor thresholds are as follows. To begin training, the Workstation sends a start learning command to the Network Monitors from which performance data is desired (step 302). The start learning command disables the thresholds within the Network Monitor and causes the Network Monitor to periodically send data for a predefined set of network parameters to the Management Workstation. (Disabling the thresholds, however, is not necessary. One could have the learning mode operational in parallel with monitoring using existing thresholds.) The set of parameters may be any or all of the previously mentioned parameters for which thresholds are or may be defined.

Throughout the learning period, the Network Monitor sends "snapshots" of the network's performance to the Workstation which, in turn, stores the data in a performance history database 306 (step 304). The network manager sets the length of the learning period. Typically, it should be long enough to include the full range of load conditions that the network experiences so that a representative performance history is generated. It should also be long enough so that short periods of overload or faulty behavior do not distort the resulting averages.

After the learning period has expired, the network manager, through the Management Workstation, sends a stop learning command to the Monitor (step 308). The Monitor ceases automatically sending further performance data updates to the Workstation and the Workstation processes the data in its performance history database (step 310). The processing may involve simply computing averages for the parameters of interest or it may involve more sophisticated statistical analysis of the data, such as computing means, standard deviations, maximum and minimum values, or using curve fitting to compute rates and other pertinent parameter values.

After the Workstation has statistically analyzed the performance data, it computes a new set of thresholds for the relevant performance parameters (step 312). To do this, it uses formulas which are appropriate to the particular parameter for which a threshold is being computed. That is, if the parameter is one for which one would expect to see wide variations in its value during network monitoring, then the threshold should be set high enough so that the normal expected variations do not trigger alarms. On the other hand, if the parameter is of a type for which only small variations are expected and larger variations indicate a problem, then the threshold should be set to a value that is close to the average observed value. Examples of formulae which may be used to compute thresholds are:

-
- * Highest value seen during learning period;
 - * Highest value seen during learning period + 10%;
 - * Highest value seen during learning period + 50%;
 - * Highest value seen during learning period + user-defined percent;
 - * Any value of the parameter other than zero;
 - * Average value seen during learning period + 50%; and
 - * Average value seen during learning period + user-defined percent
-

As should be evident from these examples, there is a broad range of possibilities regarding how to compute a particular threshold. The choice, however, should reflect the parameter's importance in signaling serious network problems and its normal expected behavior (as may be evidenced from the performance history acquired for the parameter during the learning mode).

After the thresholds are computed, the Workstation loads them into the Monitor and instructs the Monitor to revert to normal monitoring using the new thresholds (step 314).

This procedure provides a mechanism enabling the network manager to adaptively reset thresholds in response to changing conditions on the network, shifting usage patterns and evolving network topology. As the network changes over time, the network manager merely invokes the adaptive threshold setting feature and updates the thresholds to reflect those changes.

The Diagnostic Analyzer Module:

The Management Workstation includes a diagnostic analyzer module which automatically detects and diagnoses the existence and cause of certain types of network problems. The functions of the diagnostic module may actually be distributed among the Workstation and the Network Monitors which are active on the network. In principle, the diagnostic analyzer module includes the following elements for performing its fault detection and analysis functions.

The Management Workstation contains a reference model of a normally operating network. The reference model is generated by observing the performance of the network over an extended period of time and computing averages of the performance statistics that were observed during the observation period. The reference model provides a reference against which future network performance can be compared so as to diagnose and analyze potential problems. The Network Monitor (in particular, the STATS module) includes alarm thresholds on a selected set of the parameters which it monitors. Some of those thresholds are set on parameters which tend to be indicative of the onset or the presence of particular network problems.

During monitoring, when a Monitor threshold is exceeded, thereby indicating a potential problem (e.g. in a

TCP connection), the Network Monitor alerts the Workstation by sending an alarm. The Workstation notifies the user and presents the user with the option of either ignoring the alarm or invoking a diagnostic algorithm to analyze the problem. If the user invokes the diagnostic algorithm, the Workstation compares the current performance statistics to its reference model to analyze the problem and report its results. (Of course, this may also be handled automatically so as to not require user intervention.) The Workstation obtains the data on current performance of the network by retrieving the relevant performance statistics from all of the segment Network Monitors that may have information useful to diagnosing the problem.

The details of a specific example involving poor TCP connection performance will now be described. This example refers to a typical network on which the diagnostic analyzer resides, such as the network illustrated in FIG. 25. It includes three segments labelled S1, S2, and S3, a router R1 connecting S1 to S2, a router R2 connecting S2 to S3, and at least two nodes, node A on S1 which communicates with node B on S3. On each segment there is also a Network Monitor 324 to observe the performance of its segment in the manner described earlier. A Management Workstation 320 is also located on S1 and it includes a diagnostic analyzer module 322. For this example, the symptom of the network problem is degraded performance of a TCP connection between Nodes A and B.

A TCP connection problem may manifest itself in a number of ways, including, for example, excessively high numbers for any of the following:

- errors
- packets with bad sequence numbers
- packets retransmitted
- bytes retransmitted
- out of order packets
- out of order bytes
- packets after window closed
- bytes after window closed
- average and maximum round trip times

or by an unusually low value for the current window size. By setting the appropriate thresholds, the Monitor is programmed to recognize any one or more of these symptoms. If any one of the thresholds is exceeded, the Monitor sends an alarm to the Workstation. The Workstation is programmed to recognize the particular alarm as related to an event which can be further analyzed by its diagnostic analyzer module 322. Thus, the Workstation presents the user with the option of invoking its diagnostic capabilities (or automatically invokes the diagnostic capabilities).

In general terms, when the diagnostic analyzer is invoked, it looks at the performance data that the segment Monitors produce for the two nodes, for the dialogs between them and for the links that interconnect them and compares that data to the reference model for the network. If a significant divergence from the reference model is identified, the diagnostic analyzer informs the Workstation (and the user) about the nature of the divergence and the likely cause of the problem. In conducting the comparison to "normal" network performance, the network circuit involved in communications between nodes A and B is decomposed into its individual components and diagnostic analysis is performed on each link individually in the effort to isolate the problem further.

The overall structure of the diagnostic algorithm 400 is shown in FIG. 26. When invoked for analyzing a possible TCP problem between nodes A and B, diagnostic analyzer

322 checks for a TCP problem at node A when it is acting as a source node (step 402). To perform this check, diagnostic algorithm 400 invokes a source node analyzer algorithm 450 shown in FIG. 27. If a problem is identified, the Workstation reports that there is a high probability that node A is causing a TCP problem when operating as a source node and it reports the results of the investigation performed by algorithm 450 (step 404).

If node A does not appear to be experiencing a TCP problem when acting as a source node, diagnostic analyzer 322 checks for evidence of a TCP problem at node B when it is acting as a sink node (step 406). To perform this check, diagnostic algorithm 400 invokes a sink node analyzer algorithm 470 shown in FIG. 28. If a problem is identified, the Workstation reports that there is a high probability that node B is causing a TCP problem when operating as a sink node and it reports the results of the investigation performed by algorithm 470 (step 408).

Note that source and sink nodes are concepts which apply to those dialogs for which a direction of the communication can be defined. For example, the source node may be the one which initiated the dialog for the purpose of sending data to the other node, i.e., the sink node.

If node B does not appear to be experiencing a TCP problem when acting as a sink node, diagnostic analyzer 322 checks for evidence of a TCP problem on the link between Node A and Node B (step 410). To perform this check, diagnostic algorithm 400 invokes a link analysis algorithm 550 shown in FIG. 29. If a problem is identified, the Workstation reports that there is a high probability that a TCP problem exists on the link and it reports the results of the investigation performed by link analysis algorithm 550 (step 412).

If the link does not appear to be experiencing a TCP problem, diagnostic analyzer 322 checks for evidence of a TCP problem at node B when it is acting as a source node (step 414). To perform this check, diagnostic algorithm 400 invokes the previously mentioned source algorithm 450 for Node B. If a problem is identified, the Workstation reports that there is a medium probability that node B is causing a TCP problem when operating as a source node and it reports the results of the investigation performed by algorithm 450 (step 416).

If node B does not appear to be experiencing a TCP problem when acting as a source node, diagnostic analyzer 322 checks for a TCP problem at node A when it is acting as a sink node (step 418). To perform this check, diagnostic algorithm 400 invokes sink node analyzer algorithm 470 for Node A. If a problem is identified, the Network Monitor reports that there is a medium probability that node A is causing a TCP problem when operating as a sink node and it reports the results of the investigation performed by algorithm 470 (step 420).

Finally, if node A does not appear to be experiencing a TCP problem when acting as a sink node, diagnostic analyzer 322 reports that it was not able to isolate the cause of a TCP problem (step 422).

The algorithms which are called from within the above-described diagnostic algorithm will now be described.

Referring to FIG. 27, source node analyzer algorithm 450 checks whether a particular node is causing a TCP problem when operating as a source node. The strategy is as follows. To determine whether a TCP problem exists at this node which is the source node for the TCP connection, look at other connections for which this node is a source. If other TCP connections are okay, then there is probably not a problem with this node. This is an easy check with a high

probability of being correct. If no other good connections exist, then look at the lower layers for possible reasons. Start at DLL and work up as problems at lower layers are more fundamental, i.e., they cause problems at higher layers whereas the reverse is not true.

In accordance with this approach, algorithm 450 first determines whether the node is acting as a source node in any other TCP connection and, if so, whether the other connection is okay (step 452). If the node is performing satisfactorily as a source node in another TCP connection, algorithm 450 reports that there is no problem at the source node and returns to diagnostic algorithm 400 (step 454). If algorithm 450 cannot identify any other TCP connections involving this node that are okay, it moves up through the protocol stack checking each level for a problem. In this case, it then checks for DLL problems at the node when it is acting as a source node by calling an DLL problem checking routine 510 (see FIG. 30) (step 456). If a DLL problem is found, that fact is reported (step 458). If no DLL problems are found, algorithm 450 checks for an IP problem at the node when it is acting as a source by calling an IP problem checking routine 490 (see FIG. 31) (step 460). If an IP problem is found, that fact is reported (step 462). If no IP problems are found, algorithm 450 checks whether any other TCP connection in which the node participates as a source is not okay (step 464). If another TCP connection involving the node exists and it is not okay, algorithm 450 reports a TCP problem at the node (step 466). If no other TCP connections where the node is acting as a source node can be found, algorithm 450 exits.

Referring to FIG. 28, sink node analyzer algorithm 470 checks whether a particular node is causing a TCP problem when operating as a sink node. It first determines whether the node is acting as a sink node in any other TCP connection and, if so, whether the other connection is okay (step 472). If the node is performing satisfactorily as a sink node in another TCP connection, algorithm 470 reports that there is no problem at the source node and returns to diagnostic algorithm 400 (step 474). If algorithm 470 cannot identify any other TCP connections involving this node that are okay, it then checks for DLL problems at the node when it is acting as a sink node by calling DLL problem checking routine 510 (step 476). If a DLL problem is found, that fact is reported (step 478). If no DLL problems are found, algorithm 470 checks for an IP problem at the node when it is acting as a sink by calling IP problem checking routine 490 (step 480). If an IP problem is found, that fact is reported (step 482). If no IP problems are found, algorithm 470 checks whether any other TCP connection in which the node participates as a sink is not okay (step 484). If another TCP connection involving the node as a sink exists and it is not okay, algorithm 470 reports a TCP problem at the node (step 486). If no other TCP connections where the node is acting as a sink node can be found, algorithm 470 exits.

Referring to FIG. 31, IP problem checking routine 490 checks for IP problems at a node. It does this by comparing the IP performance statistics for the node to the reference model (steps 492 and 494). If it detects any significant deviations from the reference model, it reports that there is an IP problem at the node (step 496). If no significant deviations are noted, it reports that there is no IP problem at the node (step 498).

As revealed by examining FIG. 30, DLL problem checking routine 510 operates in a similar manner to IP problem checking routine 490, with the exception that it examines a different set of parameters (i.e., DLL parameters) for significant deviations.

Referring the FIG. 29, link analysis logic 550 first determines whether any other TCP connection for the link is operating properly (step 552). If a properly operating TCP connection exists on the link, indicating that there is no link problem, link analysis logic 550 reports that the link is okay (step 554). If a properly operating TCP connection cannot be found, the link is decomposed into its constituent components and an IP link component problem checking routine 570 (see FIG. 32) is invoked for each of the link components (step 556). IP link component problem routine 570 evaluates the link component by checking the IP layer statistics for the relevant link component.

The decomposition of the link into its components arranges them in order of their distance from the source node and the analysis of the components proceeds in that order. Thus, for example, the link components which make up the link between nodes A and B include in order: segment S1, router R1, segment S2, router R2, and segment S3. The IP data for these various components are analyzed in the following order:

IP data for segment S1
 IP data for address R1
 IP data for source node to R1
 IP data for S1 to S2
 IP data for S2
 IP data for address R2
 IP data for S3
 IP data for S2 to S3
 IP data for S1 to S3

As shown in FIG. 32, IP link component problem checking routine 570 compares IP statistics for the link component to the reference model (step 572) to determine whether network performance deviates significantly from that specified by the model (step 574). If significant deviations are detected, routine 570 reports that there is an IP problem at the link component (step 576). Otherwise, it reports that it found no IP problem (step 578).

Referring back to FIG. 29, after completing the IP problem analysis for all of the link components, logic 550 then invokes a DLL link component problem checking routine 580 (see FIG. 33) for each link component to check its DLL statistics (step 558).

DLL link problem routine 580 is similar to IP link problem routine 570. As shown in FIG. 33, DLL link problem checking routine 580 compares DLL statistics for the link to the reference model (step 582) to determine whether network performance at the DLL deviates significantly from that specified by the model (step 584). If significant deviations are detected, routine 580 reports that there is a DLL problem at the link component (step 586). Otherwise, it reports that no DLL problems were found (step 588).

Referring back to FIG. 29, after completing the DLL problem analysis for all of the link components, logic 550 checks whether there is any other TCP on the link (step 560). If another TCP exists on the link (which implies that the other TCP is also not operating properly), logic 550 reports that there is a TCP problem on the link (step 562). Otherwise, logic 550 reports that there was not enough information from the existing packet traffic to determine whether there was a link problem (step 564).

If the analysis of the link components does not isolate the source of the problem and if there were components for which sufficient information was not available (due possibly to lack of traffic over through that component), the user may send test messages to those components to generate the information needed to evaluate its performance.

The reference model against which comparisons are made to detect and isolate malfunctions may be generated by examining the behavior of the network over an extended period of operation or over multiple periods of operation. During those periods of operation, average values and maximum excursions (or standard deviations) for observed statistics are computed. These values provide an initial estimate of a model of a properly functioning system. As more experience with the network is obtained and as more historical data on the various statistics is accumulated the thresholds for detecting actual malfunctions or imminent malfunctions and the reference model can be revised to reflect the new experience.

What constitutes a significant deviation from the reference model depends upon the particular parameter involved. Some parameters will not deviate from the expected norm and thus any deviation would be considered to be significant, for example, consider ICMP messages of type "destination unreachable," IP errors, TCP errors. Other parameters will normally vary within a wide range of acceptable values, and only if they move outside of that range should the deviation be considered significant. The acceptable ranges of variation can be determined by watching network performance over a sustained period of operation.

The parameters which tend to provide useful information for identifying and isolating problems at the node level for the different protocols and layers include the following.

TCP

- error rate
- header byte rate
- packets retransmitted
- bytes retransmitted
- packets after window closed
- bytes after window closed

UDP

- error rate
- header byte rate

IP

- error rate
- header byte rate
- fragmentation rate
- all ICMP messages of type destination unreachable, parameter problem, redirection

DLL

- error rate
- runts

For diagnosing network segment problems, the above-identified parameters are also useful with the addition of the alignment rate and the collision rate at the DLL. All or some subset of these parameters may be included among the set of parameters which are examined during the diagnostic procedure to detect and isolate network problems.

The above-described technique can be applied to a wide range of problems on the network, including among others, the following:

- TCP Connection fails to establish
- UDP Connection performs poorly
- UDP not working at all
- IP poor performance/high error rate
- IP not working at all
- DLL poor performance/high error rate
- DLL not working at all

For each of these problems, the diagnostic approach would be similar to that described above, using, of course, different parameters to identify the potential problem and isolate its cause.

The Event Timing Module

Referring again to FIG. 5, the RTP is programmed to detect the occurrence of certain transactions for which timing information is desired. The transactions typically occur within a dialog at a particular layer of the protocol stack and they involve a first event (i.e., an initiating event) and a subsequent partner event or response. The events are protocol messages that arrive at the Network Monitor, are parsed by the RTP and then passed to Event Timing Module (ETM) for processing. A transaction of interest might be, for example, a read of a file on a server. In that case, the initiating event is the read request and the partner event is the read response. The time of interest is the time required to receive a response to the read request (i.e., the transaction time). The transaction time provides a useful measure of network performance and if measured at various times throughout the day under different load conditions gives a measure of how different loads affect network response times. The layer of the communication protocol at which the relevant dialog takes place will of course depend upon the nature of the event.

In general, when the RTP detects an event, it transfers control to the ETM which records an arrival time for the event. If the event is an initiating event, the ETM stores the arrival time in an event timing database 300 (see FIG. 34) for future use. If the event is a partner event, the ETM computes a difference between that arrival time and an earlier stored time for the initiating event to determine the complete transaction time.

Event timing database 300 is an array of records 302. Each record 302 includes a dialog field 304 for identifying the dialog over which the transactions of interest are occurring and it includes an entry type field 306 for identifying the event type of interest. Each record 302 also includes a start time field 308 for storing the arrival time of the initiating event and an average delay time field 310 for storing the computed average delay for the transactions. A more detailed description of the operation of the ETM follows.

Referring to FIG. 35, when the RTP detects the arrival of a packet of the type for which timing information is being kept, it passes control to the ETM along with relevant information from the packet, such as the dialog identifier and the event type (step 320). The ETM then determines whether it is to keep timing information for that particular event by checking the event timing database (step 322). Since each event type can have multiple occurrences (i.e., there can be multiple dialogs at a given layer), the dialog identifier is used to distinguish between events of the same type for different dialogs and to identify those for which information has been requested. All of the dialog/events of interest are identified in the event timing database. If the current dialog and event appear in the event timing database, indicating that the event should be timed, the ETM determines whether the event is a starting event or an ending event so that it may be processed properly (step 324). For certain events, the absence of a start time in the entry field of the appropriate record 302 in event timing database 300 is one indicator that the event represents a start time; otherwise, it is an end time event. For other events, the ETM determines if the start time is to be set by the event type as specified in the packet being parsed. For example, if the event is a file read a start time is stored. If the event is the read completion it represents an end time. In general, each protocol event will have its own intrinsic meaning for how to determine start and end times.

Note that the arrival time is only an estimate of the actual arrival time due to possible queuing and other processing delays. Nevertheless, the delays are generally so small in comparison to the transaction times being measured that they are of little consequence.

In step 324, if the event represents a start time, the ETM gets the current time from the kernel and stores it in start time field 308 of the appropriate record in event timing database 300 (step 326). If the event represents an end time event, the ETM obtains the current time from the kernel and computes a difference between that time and the corresponding start time found in event timing database 300 (step 328). This represents the total time for the transaction of interest. It is combined with the stored average transaction time to compute a new running average transaction time for that event (step 330).

Any one of many different methods can be used to compute the running average transaction time. For example, the following formula can be used:

$$\text{New Avg.} = [(5 * \text{stored Avg.}) + \text{Transaction Time}] / 6$$
 After six transactions have been timed, the computed new average becomes a running average for the transaction times. The ETM stores this computed average in the appropriate record of event timing database 300, replacing the previous average transaction time stored in that record, and it clears start time entry field 308 for that record in preparation for timing the next transaction.

After processing the event in steps 322, 326, and 330, the ETM checks the age of all of the start time entries in the event timing database 300 to determine if any of them are too "old" (step 332). If the difference between the current time and any of the start times exceeds a preselected threshold, indicating that a partner event has not occurred within a reasonable period of time, the ETM deletes the old start time entry for that dialog/event (step 334). This insures that a missed packet for a partner event does not result in an erroneously large transaction time which throws off the running average for that event.

If the average transaction time increases beyond a preselected threshold set for timing events, an alarm is sent to the Workstation.

Two examples will now be described to illustrate the operation of the ETM for specific event types. In the first example, Node A of FIG. 25 is communicating with Node B using the NFS protocol. Node A is the client while Node B is the server. The Network Monitor resides on the same segment as node A, but this is not a requirement. When Node A issues a read request to Node B, the Network Monitor sees the request and the RTP within the Network Monitor transfers control to the ETM. Since it is a read, the ETM stores a start time in the Event Timing Database. Thus, the start time is the time at which the read was initiated.

After some delay, caused by the transmission delays of getting the read message to node B, node B performs the read and sends a response back to node A. After some further transmission delays in returning the read response, the Network Monitor receives the second packet for the event. At the time, the ETM recognizes that the event is an end time event and updates the average transaction time entry in the appropriate record with a new computed running average. The ETM then compares the average transaction time with the threshold for this event and if it has been exceeded, issues an alarm to the Workstation.

In the second example, node A is communicating with Node B using the Telnet protocol. Telnet is a virtual terminal protocol. The events of interest take place long after the initial connection has been established. Node A is typing at

a standard ASCII (VT100 class) terminal which is logically (through the network) connected to Node B. Node B has an application which is receiving the characters being typed on Node A and, at appropriate times, indicated by the logic of the applications, sends characters back to the terminal located on Node A. Thus, every time node A sends characters to B, the Network Monitor sees the transmission.

In this case, there are several transaction times which could provide useful network performance information. They include, for example, the amount of time it takes to echo characters typed at the keyboard through the network and back to the display screen, the delay between typing an end of line command and seeing the completion of the application event come back or the network delays incurred in sending a packet and receiving acknowledgment for when it was received.

In this example, the particular time being measured is the time it takes for the network to send a packet and receive an acknowledgement that the packet has arrived. Since Telnet runs on top of TCP, which in turn runs on top of IP, the Network Monitor monitors the TCP acknowledge end-to-end time delays.

Note that this is a design choice of the implementation and that all events visible to the Network Monitor by virtue of the fact that information is in the packet could be measured.

When Node A transmits a data packet to Node B, the Network Monitor receives the packet. The RTP recognizes the packet as being part of a timed transaction and passes control to the ETM. The ETM recognizes it as a start time event, stores the start time in the event timing database and returns control to the RTP after checking for aging.

When Node B receives the data packet from Node A, it sends back an acknowledgment packet. When the Network Monitor sees that packet, it delivers the event to the ETM, which recognizes it as an end time event. The ETM calculates the delay time for the complete transaction and uses that to update the average transaction time. The ETM then compares the new average transaction time with the threshold for this event. If it has been exceeded, the ETM issues an alarm to the Workstation.

Note that this example is measuring something very different than the previous example. The first example measures the time it takes to traverse the network, perform an action and return that result to the requesting node. It measures performance as seen by the user and it includes delay times from the network as well as delay times from the File Server.

The second example is measuring network delays without looking at the service delays. That is, the ETM is measuring the amount of time it takes to send a packet to a node and receive the acknowledgement of the receipt of the message. In this example, the ETM is measuring transmission delays as well as processing delays associated with network traffic, but not anything having to do with non-network processing.

As can be seen from the above examples, the ETM can measure a broad range of events. Each of these events can be measured passively and without the cooperation of the nodes that are actually participating in the transmission. The Address Tracker Module (ATM)

Address tracker module (ATM) 43, one of the software modules in the Network Monitor (see FIG. 5), operates on networks on which the node addresses for particular node to node connections are assigned dynamically. An Appletalk@ Network, developed by Apple Computer Company, is an example of a network which uses dynamic node addressing. In such networks, the dynamic change in the address of a particular service causes difficulty troubleshooting the net-

work because the network manager may not know where the various nodes are and what they are called. In addition, foreign network addresses (e.g., the IP addresses used by that node for communication over an IP network to which it is connected) can not be relied upon to point to a particular node. ATM 43 solves this problem by passively monitoring the network traffic and collecting a table showing the node address to node name mappings.

In the following description, the network on which the Monitor is located is assumed to be an Appletalk® Network. Thus, as background for the following discussion, the manner in which the dynamic node addressing mechanism operates on that network will first be described.

When a node is activated on the Appletalk® Network, it establishes its own node address in accordance with protocol referred to as the Local Link Access Protocol (LLAP). That is, the node guesses its own node address and then verifies that no other node on the network is using that address. The node verifies the uniqueness of its guess by sending an LLAP Enquiry control packet informing all other nodes on the network that it is going to assign itself a particular address unless another node responds that the address has already been assigned. If no other node claims that address as its own by sending an LLAP acknowledgment control packet, the first node uses the address which it has selected. If another node claims the address as its own, the first node tries another address. This continues until, the node finds an unused address.

When the first node wants to communicate with a second node, it must determine the dynamically assigned node address of the second node. It does this in accordance with another protocol referred to as the Name Binding Protocol (NBP). The Name Binding Protocol is used to map or bind human understandable node names with machine understandable node addresses. The NBP allows nodes to dynamically translate a string of characters (i.e., a node name) into a node address. The node needing to communicate with another node broadcasts an NBP Lookup packet containing the name for which a node address is being requested. The node having the name being requested responds with its address and returns a Lookup Reply packet containing its address to the original requesting node. The first node then uses that address for its current communications with the second node.

Referring to FIG. 36, the network includes an Appletalk® Network segment 702 and a TCP/IP segment 704, each of which are connected to a larger network 706 through their respective gateways 708. A Monitor 710, including a Real Time Parser (RTP) 712 and an Address Tracking Module (ATM) 714, is located on Appletalk network segment 702 along with other nodes 711. A Management Workstation 716 is located on segment 704. It is assumed that Monitor 710 has the features and capabilities previously described; therefore, those features not specifically related to the dynamic node addressing capability will not be repeated here but rather the reader is referred to the earlier discussion. Suffice it to say that Monitor 710 is, of course, adapted to operate on Appletalk Network segment 702, to parse and analyze the packets which are transmitted over that segment according to the Appletalk® family of protocols and to communicate the information which it extracts from the network to Management Workstation 716 located on segment 704.

Within Monitor 710, ATM 714 maintains a name table data structure 720 such as is shown in FIG. 37. Name Table 720 includes records 722, each of which has a node name field 724, a node address field 726, an IP address field 728,

and a time field 729. ATM 714 uses Name Table 720 to keep track of the mappings of node names to node address and to IP address. The relevance of each of the fields of records 722 in Name Table 720 are explained in the following description of how ATM 714 operates.

In general, Monitor 710 operates as previously described. That is, it passively monitors all packet traffic over segment 702 and sends all packets to RTP 712 for parsing. When RTP 712 recognizes an Appletalk packet, it transfers control to ATM 714 which analyzes the packet for the presence of address mapping information.

The operation of ATM 714 is shown in greater detail in the flow diagram of FIG. 38. When ATM 714 receives control from RTP 712, it takes the packet (step 730 and strips off the lower layers of the protocol until it determines whether there is a Name Binding Protocol message inside the packet (step 732). If it is a NBP message, ATM 714 then determines whether it is new name Lookup message (step 734). If it is a new name Lookup message, ATM 714 extracts the name from the message (i.e., the name for which a node address is being requested) and adds the name to the node name field 724 of a record 722 in Name Table 720 (step 736).

If the message is an NBP message but it is not a new name Lookup message, ATM 714 determines whether it is a Lookup Reply (step 738). If it is a Lookup Reply, signifying that it contains a node name/node address binding, ATM 714 extracts the name and the assigned node address from the message and adds this information to Name Table 720 (step 740). ATM 714 does this by searching the name fields of records 722 in Name Table 720 until it locates the name. Then, it updates the node address field of the identified record to contain the node address which was extracted from the received NBP packet. ATM 714 also updates time field 729 to record the time at which the message was processed.

After ATM 714 has updated the address field of the appropriate record, it determines whether any records 722 in Name Table 720 should be aged out (step 742). ATM 714 compares the current time to the times recorded in the time fields. If the elapsed time is greater than a preselected time period (e.g. 48 hours), ATM 714 clears the record of all information (step 744). After that, it awaits the next packet from RTP 712 (step 745).

As ATM 714 is processing a packet and it determines either that it does not contain an NBP message (step 732) or it does not contain a Lookup Reply message (step 738), ATM 714 branches to step 742 to perform the age out check before going on to the next packet from RTP 712.

The Appletalk to IP gateways provide services that allow an Appletalk Node to dynamically connect to an IP address for communicating with IP nodes. This service extends the dynamic node address mechanism to the IP world for all Appletalk nodes. While the flexibility provided is helpful to the users, the network manager is faced with the problem of not knowing which Appletalk Nodes are currently using a particular IP address and thus, they can not easily track down problems created by the particular node.

ATM 714 can use passive monitoring of the IP address assignment mechanisms to provide the network manager a Name-to-IP address mapping.

If ATM 714 is also keeping IP address information, it implements the additional steps shown in FIG. 39 after completing the node name to node address mapping steps. ATM 714 again checks whether it is an NBP message (step 748). If it is an NBP message, ATM 714 checks whether it is a response to an IP address request (step 750). IP address requests are typically implied by an NBP Lookup request for an IP gateway. The gateway responds by supplying the

gateway address as well as an IP address that is assigned to the requesting node. If the NBP message is an IP address response, ATM 714 looks up the requesting node in Name Table 720 (step 752) and stores the IP address assignment in the IP address field of the appropriate record 722 (step 754).

After storing the IP address assignment information, ATM 714 locates all other records 722 in Name Table 720 which contain that IP address. Since the IP address has been assigned to a new node name, those old entries are no longer valid and must be eliminated. Therefore, ATM 714 purges the IP address fields of those records (step 756). After doing this cleanup step, ATM 714 returns control to RTP 712.

Other embodiments are within the following claims. For example, the Network Monitor can be adapted to identify node types by analyzing the type of packet traffic to or from the node. If the node being monitored is receiving mount requests, the Monitor would report that the node is behaving like node a file server. If the node is issuing routing requests, the Monitor would report that the node is behaving like a router. In either case, the network manager can check a table of what nodes are permitted to provide what functions to determine whether the node is authorized to function as either a file server or a router, and if not, can take appropriate action to correct the problem.

Appendix VI contains microfiche of the source code for an embodiment of the invention. The source code is presented in 32 files which may be grouped as follows:

Group 1:	R_DLL.C; R_ICMP.C; R_IP.C; R_NFS.C; R_PMAP.C; R_RPC.C; R_TCP.C; R_UDP.C; RTP.C
Group 2:	S_DLL.H; S.H; S_ICMP.H; S_IP.H; S_NFS.H; S_PMAP.H; S_RPC.H; S_TCP.H; S_UDP.H
Group 3:	S_DLL_A.C; S_DLL_P.C; S_ICMP_A.C; S_ICMP_P.C; S_IP_A.C; S_IP_P.C; S_NFS_A.C; S_NFS_P.C; S_PMAP_P.C; S_TCP_A.C; S_TCP_P.C; S_UDP_A.C; S_UDP_P.C; STATS_P.C

The nine files in Group 1 relate to the real time parser code and contain source code for modules that do the parsing and state machine processing. Macros for maintaining statistics also appear among these source code files. The nine files in Group 2 contain the source code that defines the structures which hold the information that is determined during parsing. Among these files, the S.H file contains the base macros for maintaining the statistics. Finally, the remaining 14 files in Group 3 contain code that locates, allocates and deallocates the structures defined by the files found in Group 2. The source code in this group also contains the code that calculates the rates and ages out the data structures that have not been used within a timeout period.

The source code is written in C language. It may be compiled by a MIPS R3000 C compiler (Ver. 2.2) and run on a MIPS 3230 workstation which has a RISC-os operating system (MIPS products are available from MIPS Computer Systems, Inc. of Sunnyvale, Calif.).

We claim:

1. A method for monitoring a plurality of communication dialogs occurring in a network of nodes, each dialog of said plurality of dialogs being effected by a transmission of one or more packets among two or more communicating nodes, each dialog of said plurality of dialogs complying with a different predefined communication protocol selected from among a plurality of protocols available in said network, said plurality of dialogs representing multiple different protocols from among said plurality of available protocols, said method comprising:

passively, in real time, and on an ongoing basis, detecting the contents of packets;

from the detected contents of said packets, identifying all of the dialogs of said plurality of communication dialogs occurring in said network;

deriving from said detected contents of said packets, information about the identified dialogs; and

for each of the identified dialogs, storing the derived information about that identified dialog.

2. The method of claim 1 wherein the derived information is information about the states of the identified dialogs.

3. The method of claim 2 wherein said step of deriving information about the states of dialogs comprises: maintaining a current state for each dialog; and

updating the current state in response to the detected contents of packets detected at a later time.

4. The method of claim 2 wherein said step of deriving information about the states of dialogs comprises:

maintaining, for each dialog, a history of events based on information derived from the contents of packets; and analyzing the history of events to derive information about the dialog.

5. The method of claim 4 wherein said step of analyzing the history includes counting events.

6. The method of claim 4 wherein said step of analyzing the history includes gathering statistics about events.

7. The method of claim 4 further comprising:

monitoring the history of events for dialogs which are inactive; and

purging from the history of events dialogs which have been inactive for a predetermined period of time.

8. The method of claim 3 wherein said step of deriving information about the states of dialogs comprises updating said current state in response to observing the transmission of at least two data related packets between nodes.

9. The method of claim 4 wherein said step of analyzing the history of events comprises:

analyzing sequence numbers of data related packets stored in said history of events; and

detecting retransmissions based on said sequence numbers.

10. The method of claim 3 further comprising:

updating the current state based on each new packet associated with said dialog; and

if an updated current state cannot be determined, consulting information about prior packets associated with said dialog as an aid in updating said state.

11. The method of claim 4 further comprising searching said history of events to identify the initiator of a dialog.

12. The method of claim 4, further comprising searching the history of events for packets which have been retransmitted.

13. The method of claim 3 wherein the full set of packets associated with a dialog up to a point in time completely define a true state of the dialog at that point in time, said step of updating the current state in response to the detected contents of transmitted packets comprises generating a current state which may not conform to the true state.

14. The method of claim 4 wherein the step of updating the current state comprises updating the current state to indicate that it is unknown.

15. The method of claim 13 further comprising updating the current state to the true state based on information about prior packets transmitted in the dialog.

16. The method of claim 14 further comprising updating the current state to the true state based on information about prior packets transmitted in the dialog.

45

17. The method of claim 2 wherein said step of deriving information about the states of dialogs occurring in said network comprises parsing said packets in accordance with more than one but fewer than all layers of a corresponding communication protocol.

18. The method of claim 1 is wherein each said communication protocol includes multiple layers, and each dialog complies with one of said layers.

19. The method of claim 2 wherein said multiple different protocols include a connectionless-type protocol in which the state of a dialog is implicit in transmitted packets, and said step of deriving information about the states of dialogs includes inferring the states of said dialogs from said packets.

20. The method of claim 3 further comprising: parsing said packets in accordance with a corresponding communication protocol; and temporarily suspending parsing of some layers of said protocol when parsing is not rapid enough to match the rate of packets to be parsed.

21. The method of claim 1 further comprising storing the derived communication information in a database.

22. The method of claim 21 further comprising, in response to an inquiry from a remotely located entity, retrieving from the database data regarding communications about which communication information had previously been detected and stored and sending the retrieved data to said remotely located entity.

23. The method of claim 1 wherein said multiple different protocols include protocols for different layers of a protocol stack as well as protocols for different protocol stacks.

46

24. Apparatus for monitoring a plurality of communication dialogs which occur in a network of nodes, each dialog of said plurality of dialogs being effected by a transmission of one or more packets among two or more communicating nodes, each dialog of said plurality of dialogs complying with a different predefined communication protocol selected from among a plurality of protocols available in said network, said plurality of dialogs representing multiple different protocols from among said plurality of available protocols, said apparatus comprising:

a monitor connected to the network medium, said monitor programmed to perform the tasks of passively, in real time, and on an ongoing basis: (1) monitoring transmitted packets; (2) from the monitored packets, identifying all of the dialogs of said plurality of communication dialogs; (3) from the monitored packets, deriving communication information associated with said plurality of communication dialogs; and (4) for each of the identified dialogs, storing the communication information about that identified dialog that is derived from said monitored packets; and

a workstation for receiving said information about dialogs from said monitor and providing an interface through which said communication information about the identified dialogs is displayed to a user.

25. The apparatus of claim 24 wherein said workstation further comprises means for enabling a user to observe events of active dialogs.

* * * * *

United States Patent [19]

Daniel, III et al.

[11] Patent Number: **4,972,453**

[45] Date of Patent: **Nov. 20, 1990**

- [54] **AUTONOMOUS EXPERT SYSTEM FOR DIRECTLY MAINTAINING REMOTE TELEPHONE SWITCHING SYSTEMS**
- [75] Inventors: **William F. Daniel, III**, Louisville; **Karen C. Loeb**, Englewood; **Charles S. Roush**, Boulder, all of Colo.
- [73] Assignee: **AT&T Bell Laboratories**, Murray Hill, N.J.
- [21] Appl. No.: **317,232**
- [22] Filed: **Feb. 28, 1989**
- [51] Int. Cl.⁵ **H04M 3/28**
- [52] U.S. Cl. **379/10; 379/15; 371/15.1; 371/18**
- [58] Field of Search **379/10, 15, 14, 32; 371/15.1, 18; 364/513**

Macleish, Thiedke, Vennergrund, IEEE Communications Magazine, Sep. 1986 vol. 24, No. 9 pp. 26-33.

Primary Examiner—Stafford D. Schreyer
Attorney, Agent, or Firm—John C. Moran

[57] **ABSTRACT**

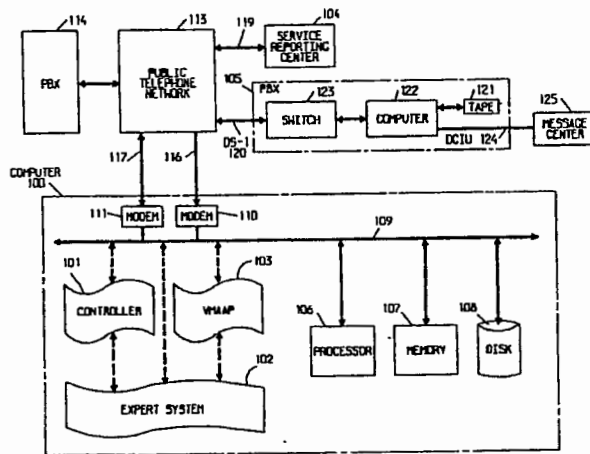
An autonomous expert system for directly maintaining remote computer systems by directly accessing the remote computer systems, diagnosing, and clearing fault conditions on those computer systems. The expert system performs those functions by first accessing a fault report from a centralized service reporting center, establishing a data connection to the computer system reporting the fault, invoking diagnostic routines on the computer system to gather data about the reported fault, analyzing the data, and, if appropriate, clearing the reported fault from the computer system. If the fault cannot be cleared, the expert system recommends maintenance procedures and replacement parts for a technician who the expert system dispatches to the remote computer. The recommendations are based on field experience stored in rules and databases maintained by the expert system. When the remote computer system is controlling a customr switching system (PBX), the expert system only invokes testing procedures in the computer system which do not disrupt stable telephone calls. The expert system access the PBX via the public telephone network.

- [56] **References Cited**
- U.S. PATENT DOCUMENTS**
- 4,456,994 6/1984 Segarra 371/15.1 X
 - 4,517,468 5/1985 Kemper et al. 290/52
 - 4,697,243 9/1987 Moore et al. 364/513
 - 4,701,845 10/1987 Andreasen et al. 371/18 X

OTHER PUBLICATIONS

"Expert Systems for AT&T Switched Network Maintenance", *AT&T Technical Journal*, vol. 67, Issue 1, pp. 93-103, Paul H. Callahan.
"Operations Systems Technology for New AT&T Network and Service Capabilities", *AT&T Technical Journal*, vol. 66, Issue 3, pp. 64-72.
Expert Systems in Central Office Switch Maintenance,

15 Claims, 24 Drawing Sheets



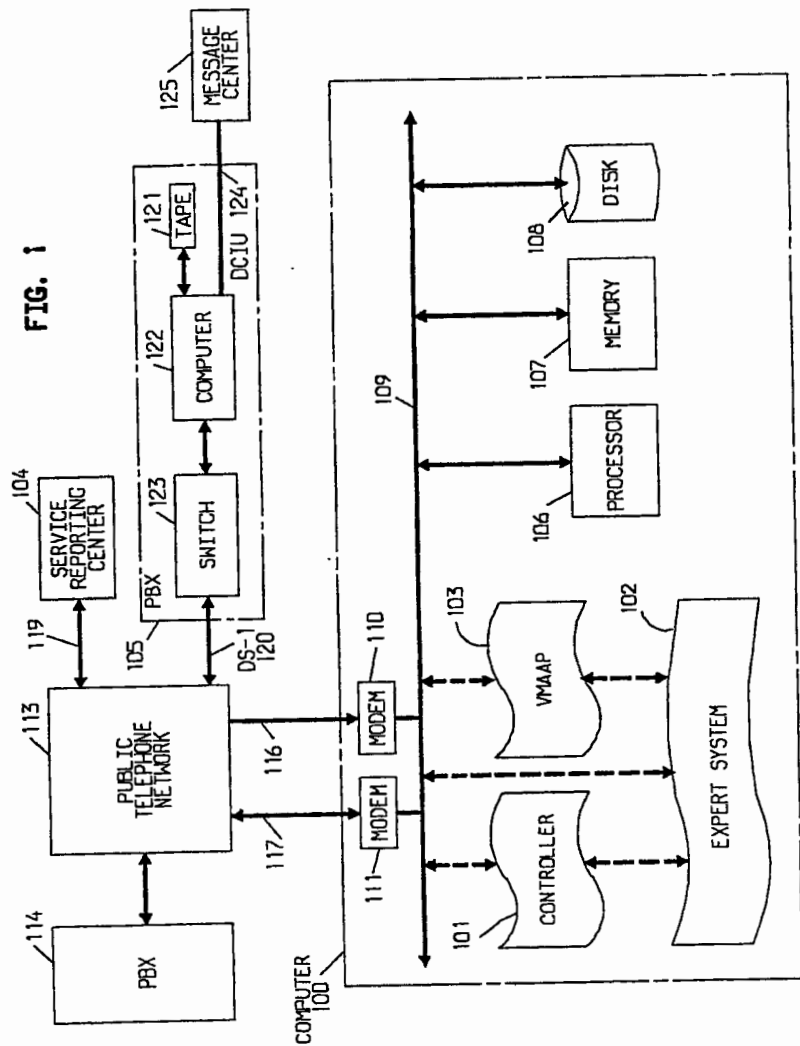


FIG. 2

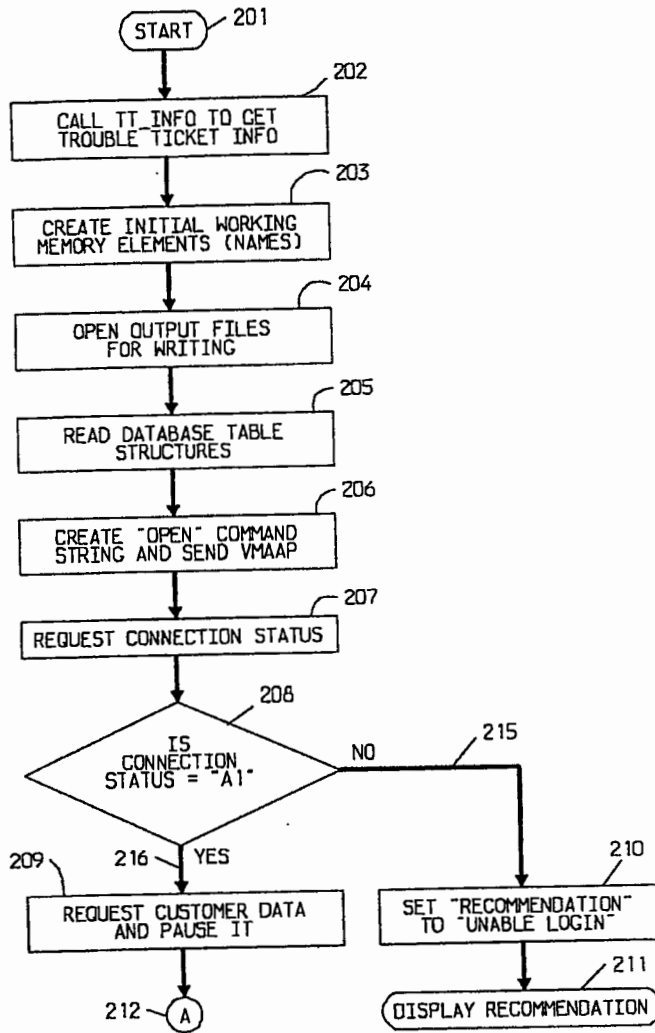


FIG. 3

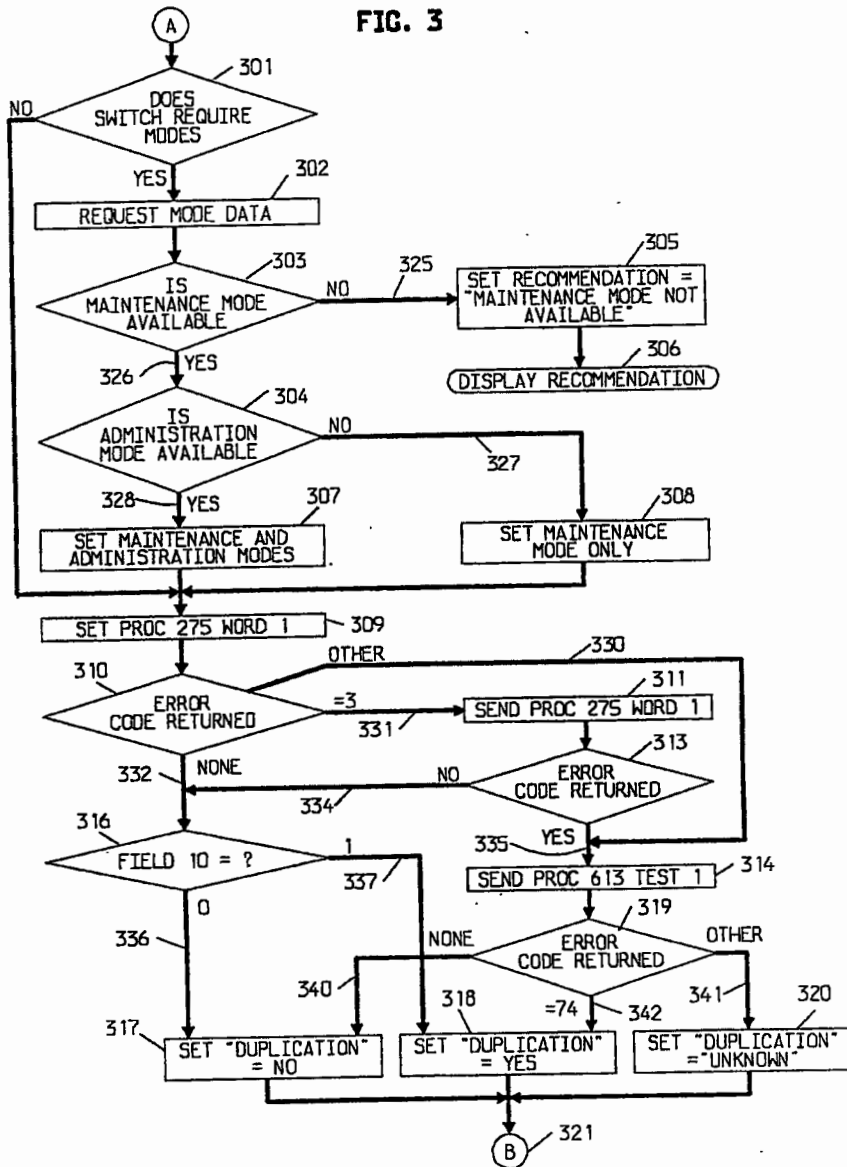


FIG. 4

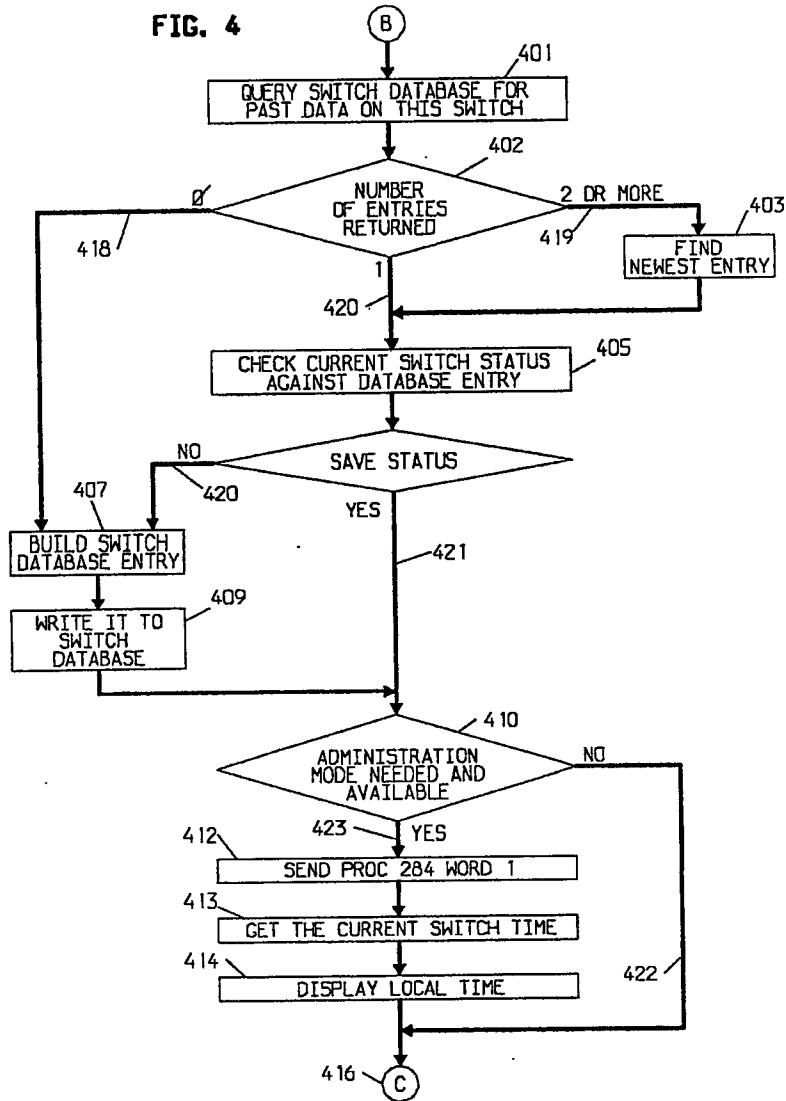


FIG. 5

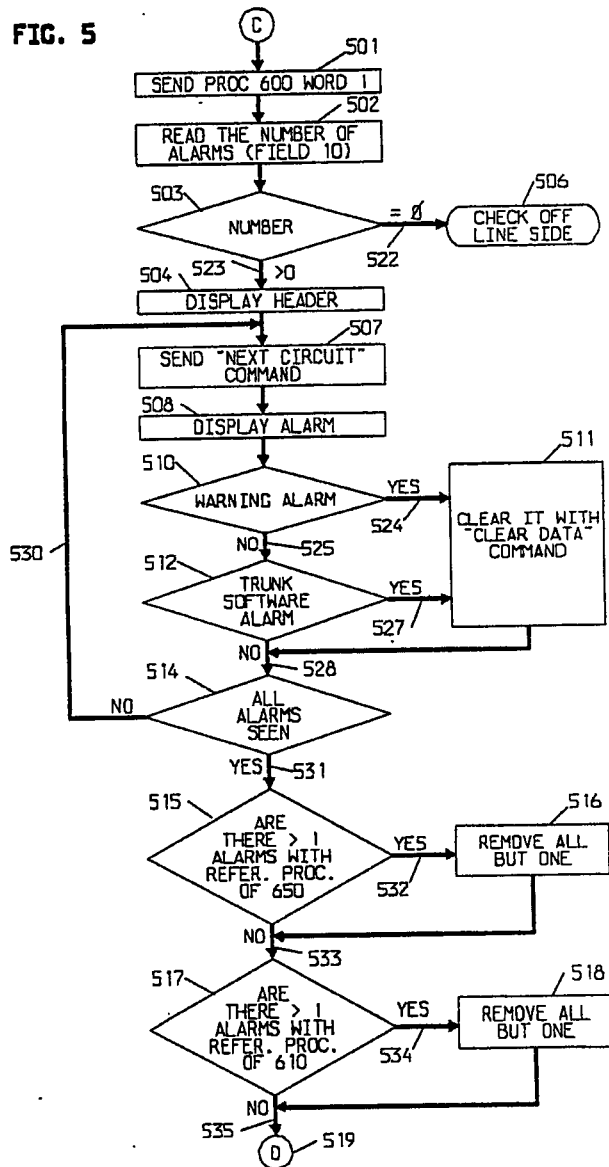


FIG. 6

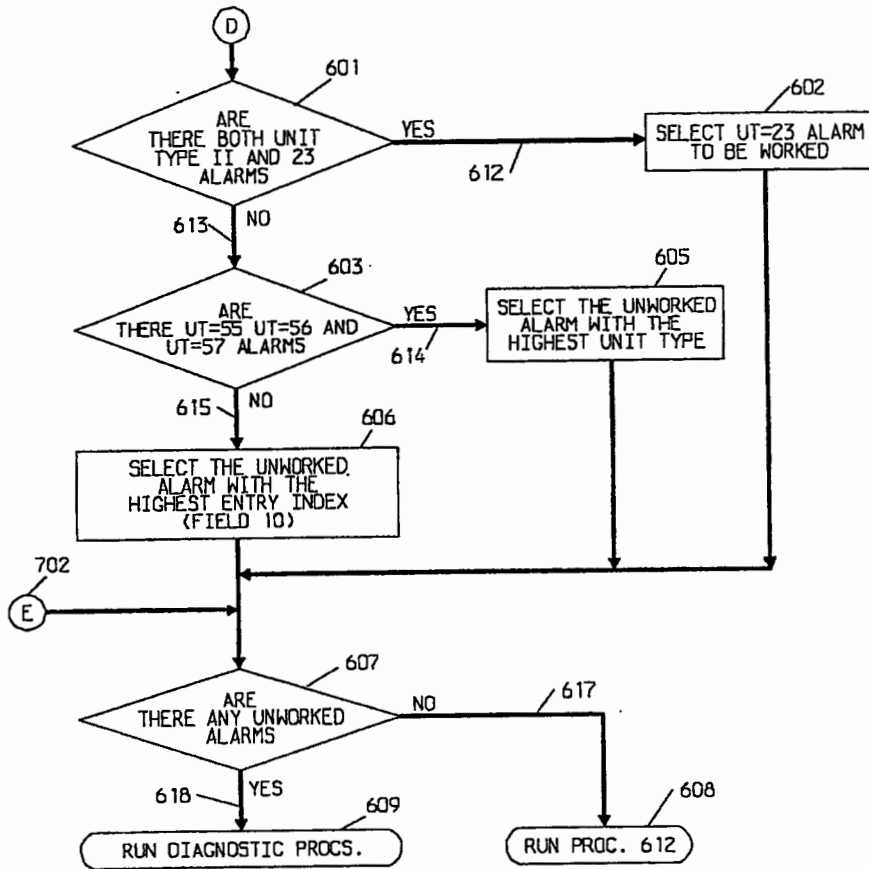


FIG. 7

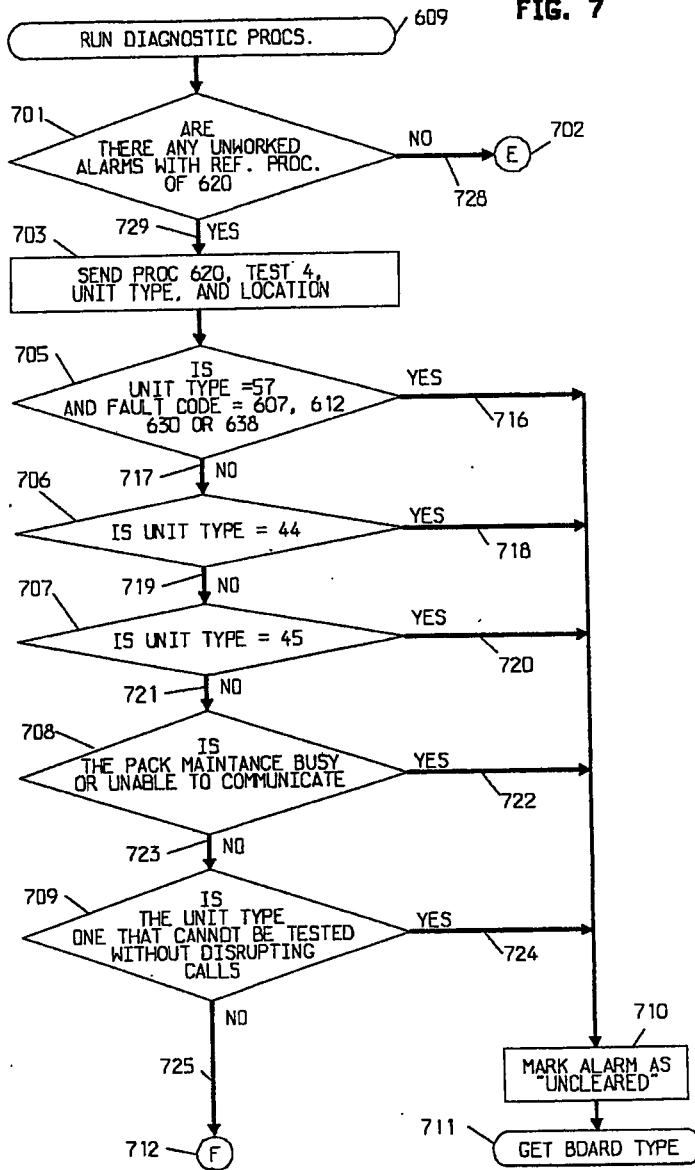


FIG. 8

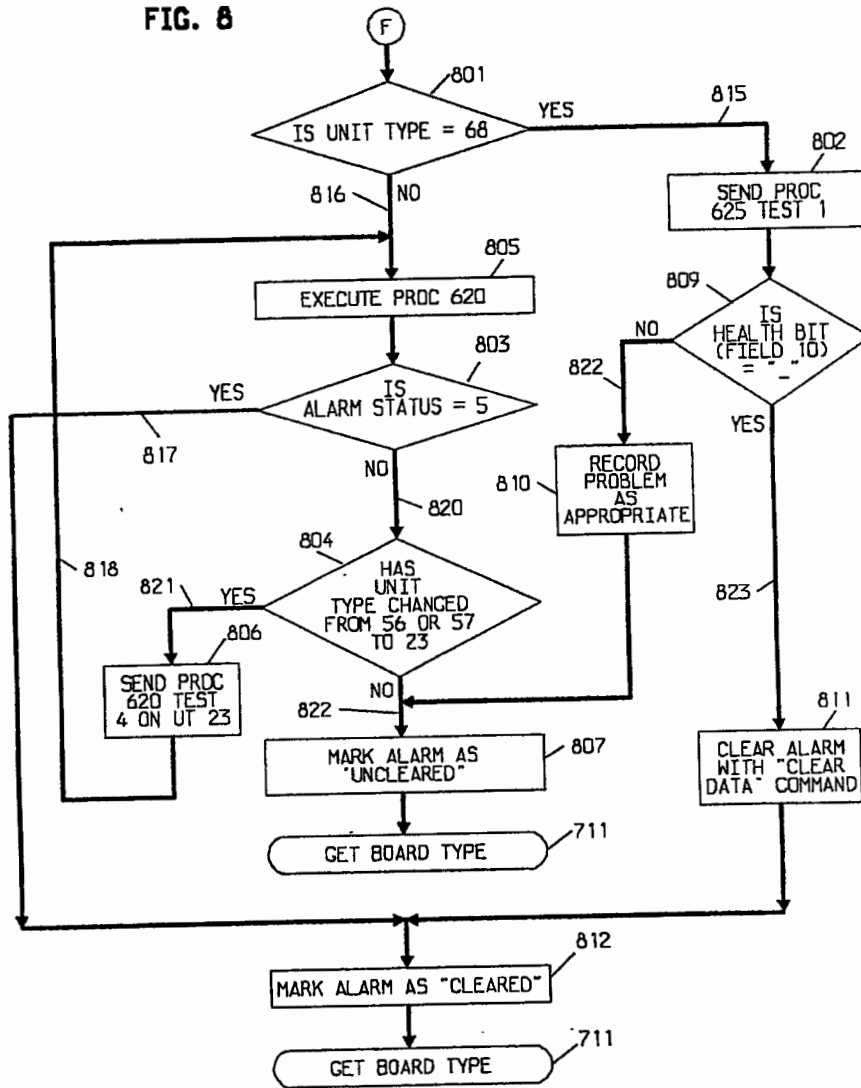
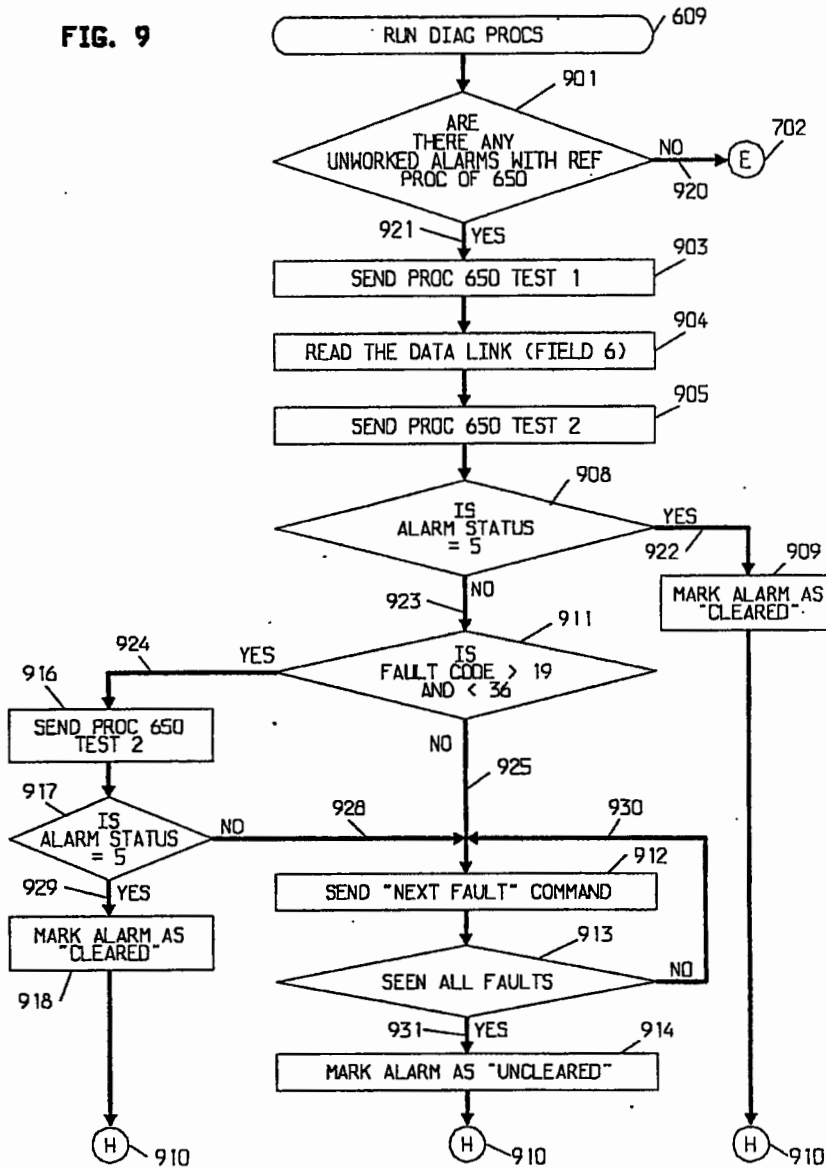


FIG. 9



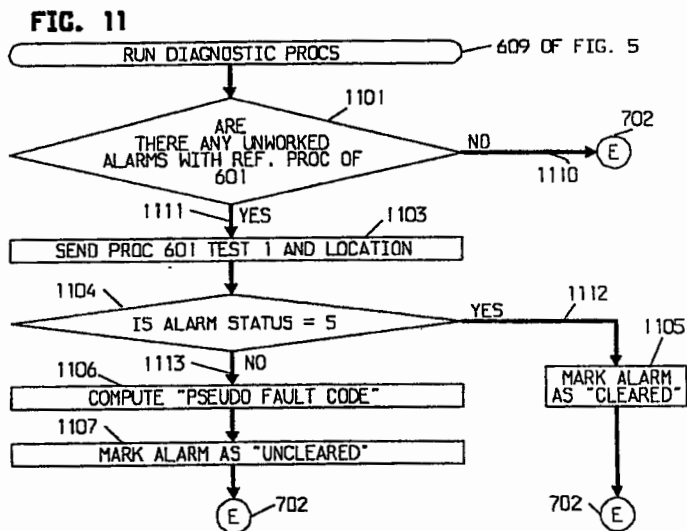
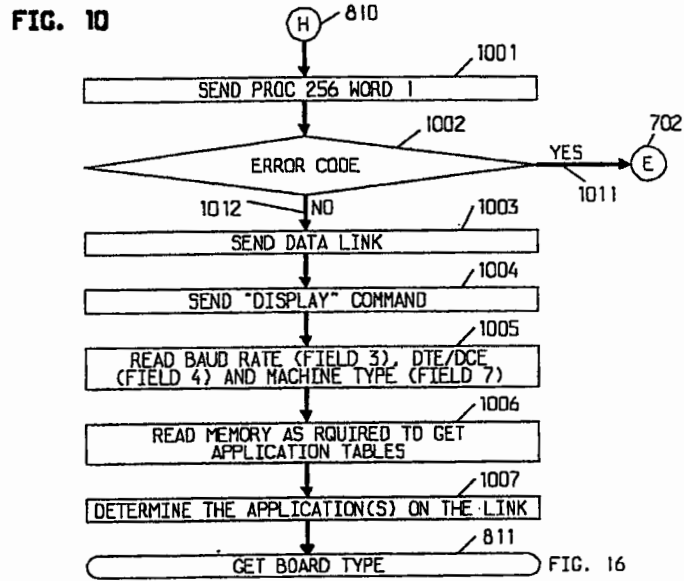


FIG. 12

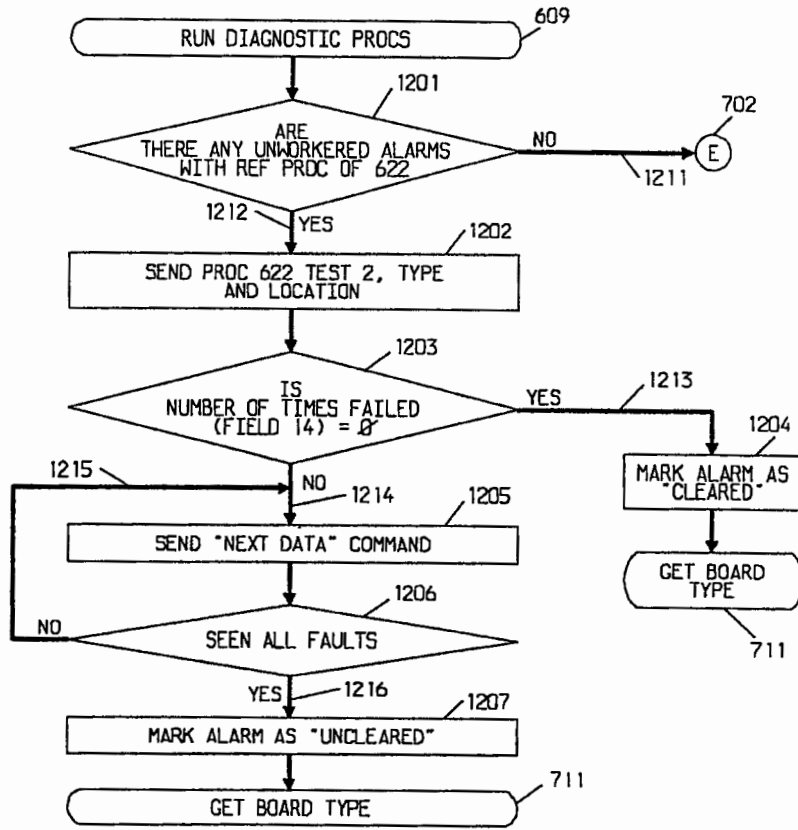


FIG. 13

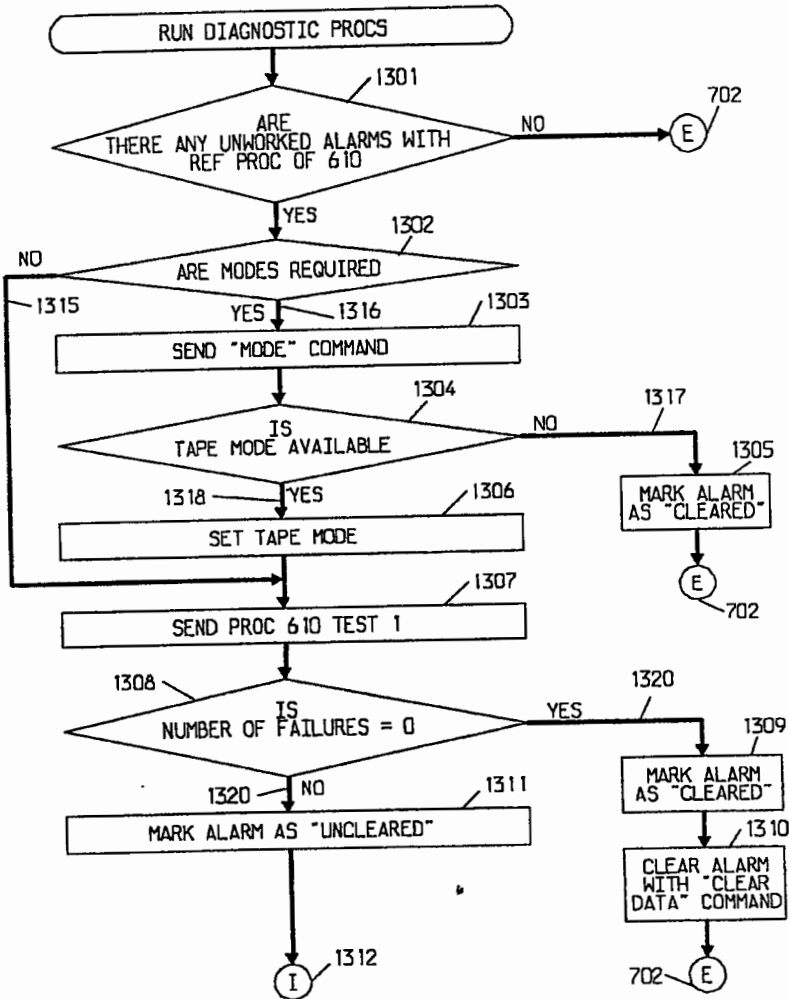


FIG. 14

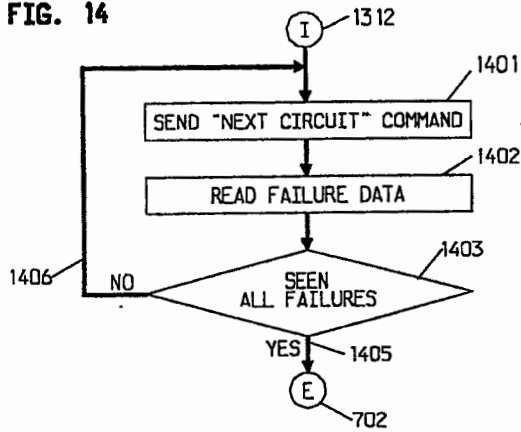


FIG. 15

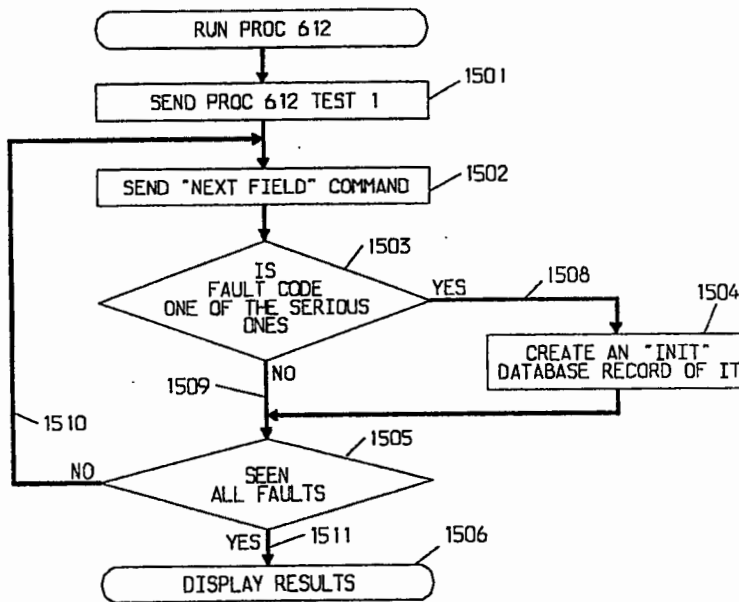


FIG. 17

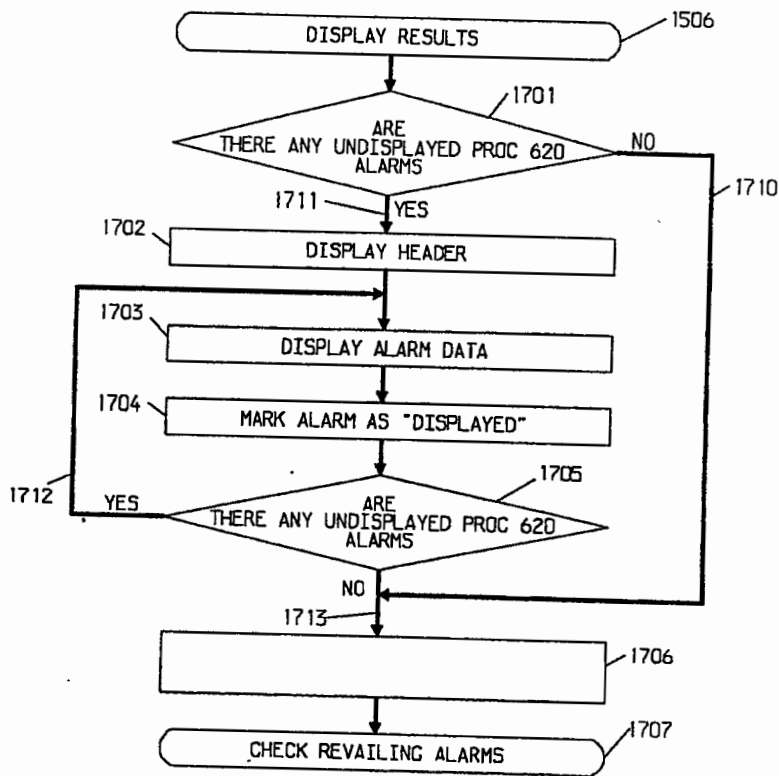


FIG. 18

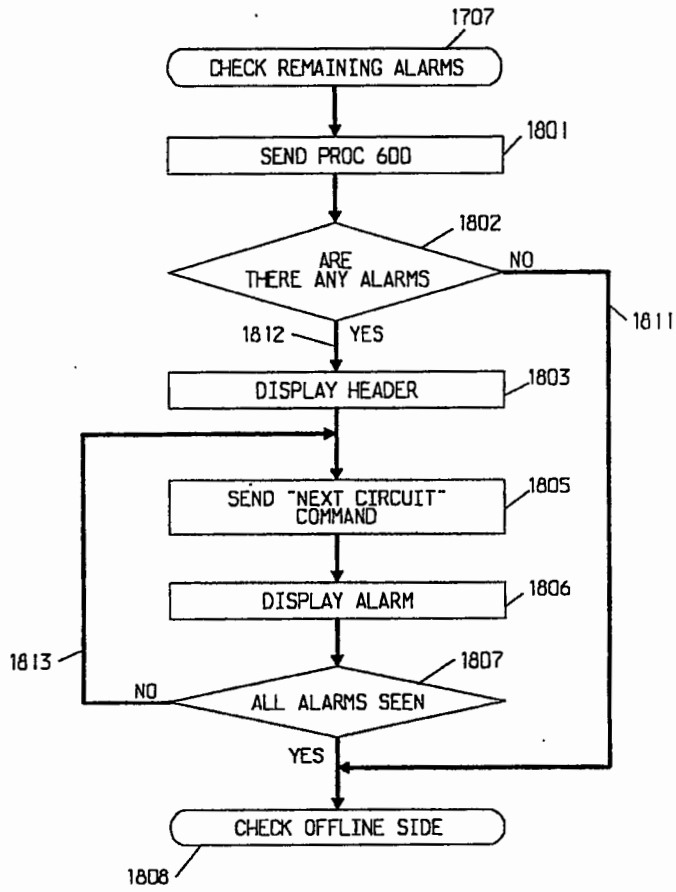


FIG. 19

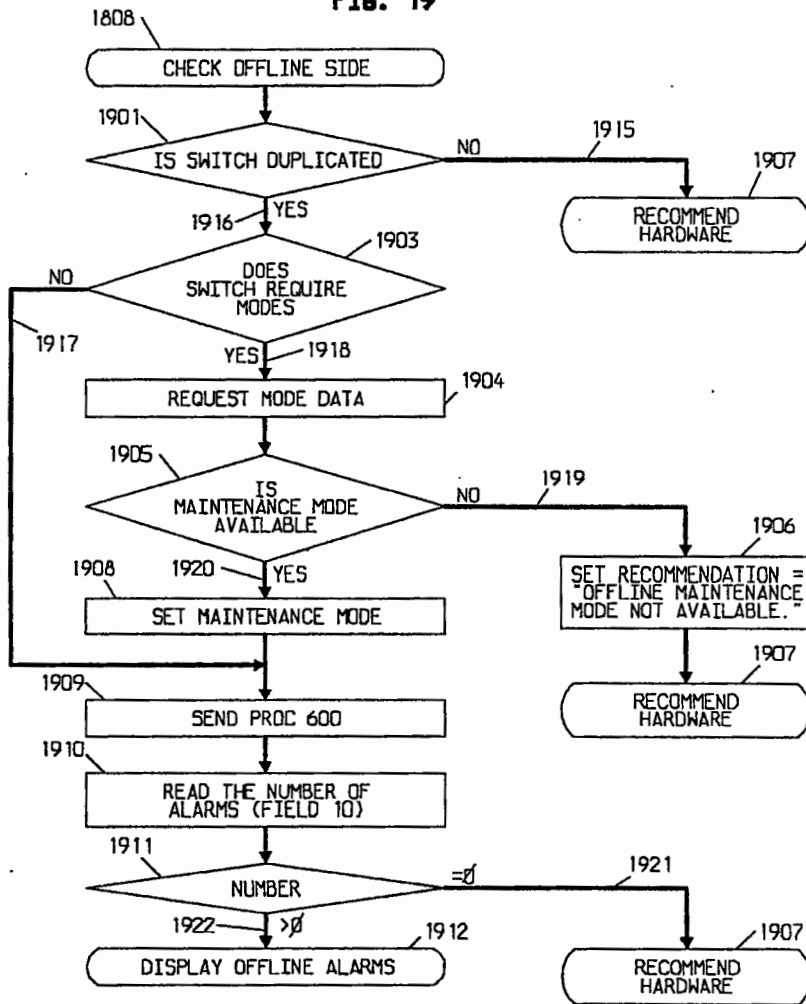


FIG. 20

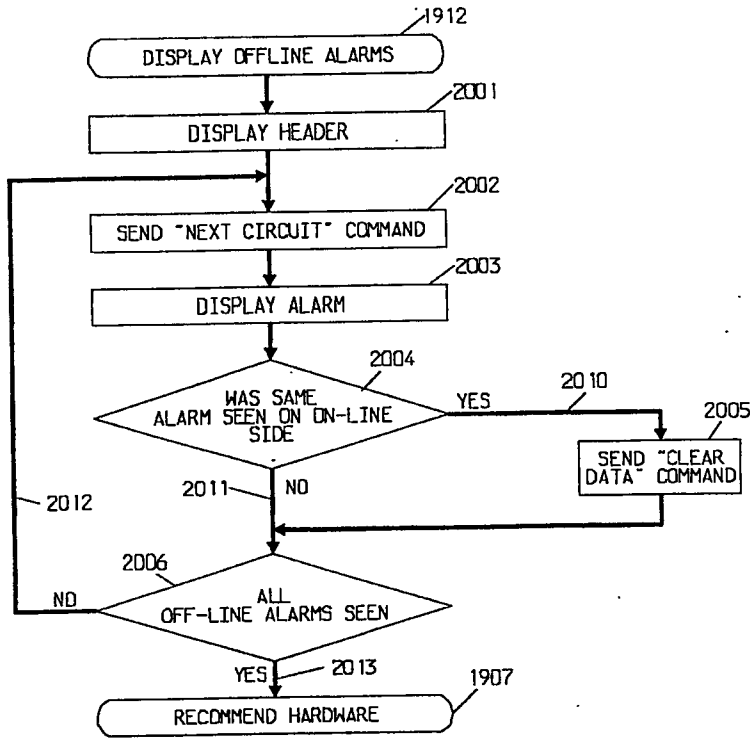


FIG. 21

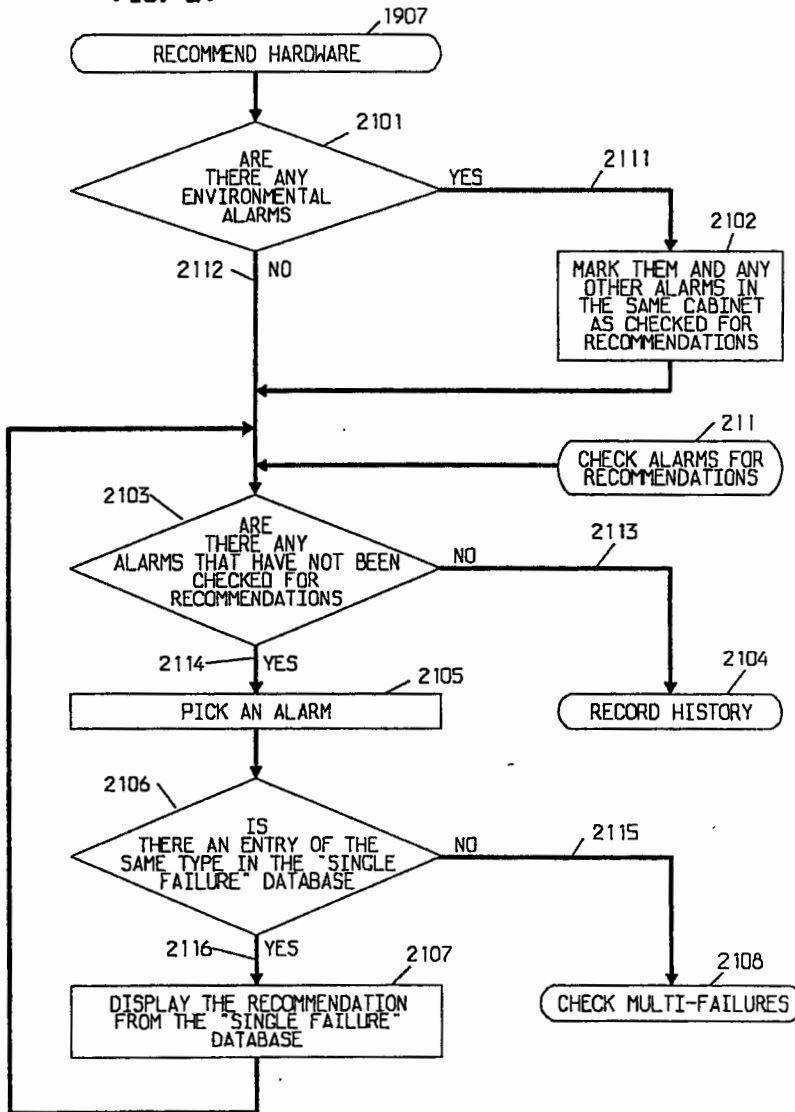


FIG. 22

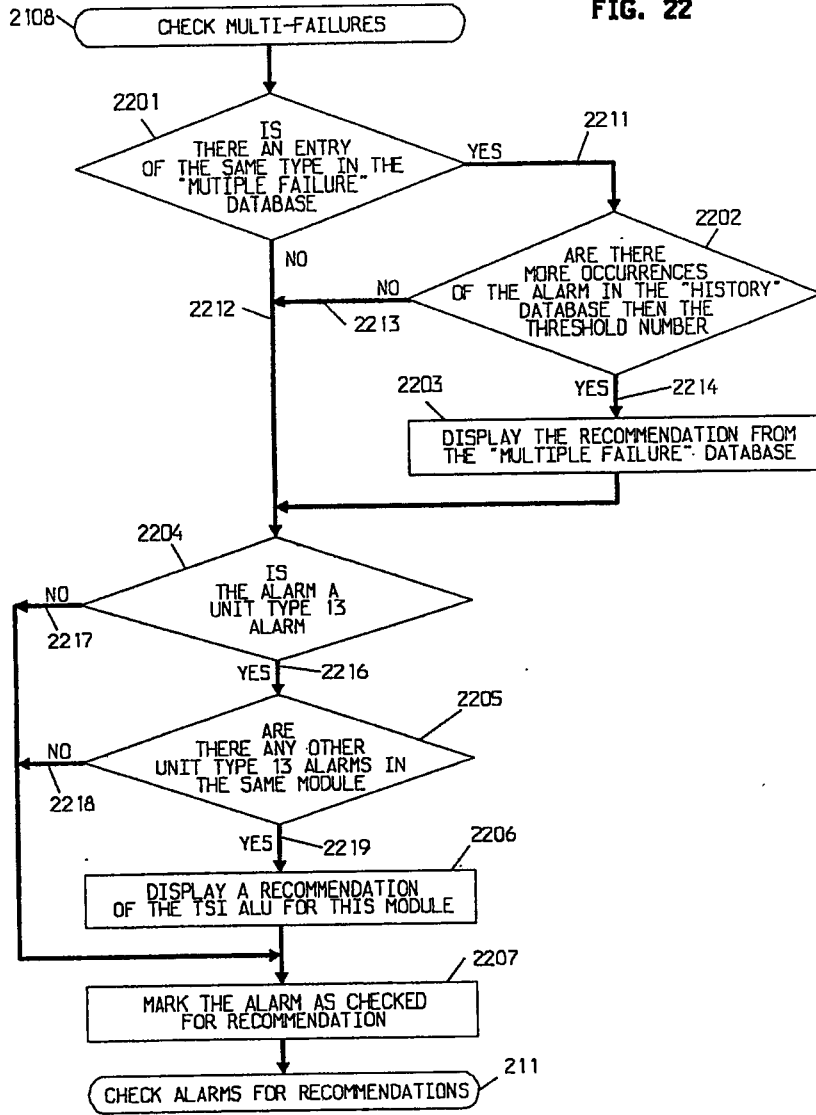


FIG. 23

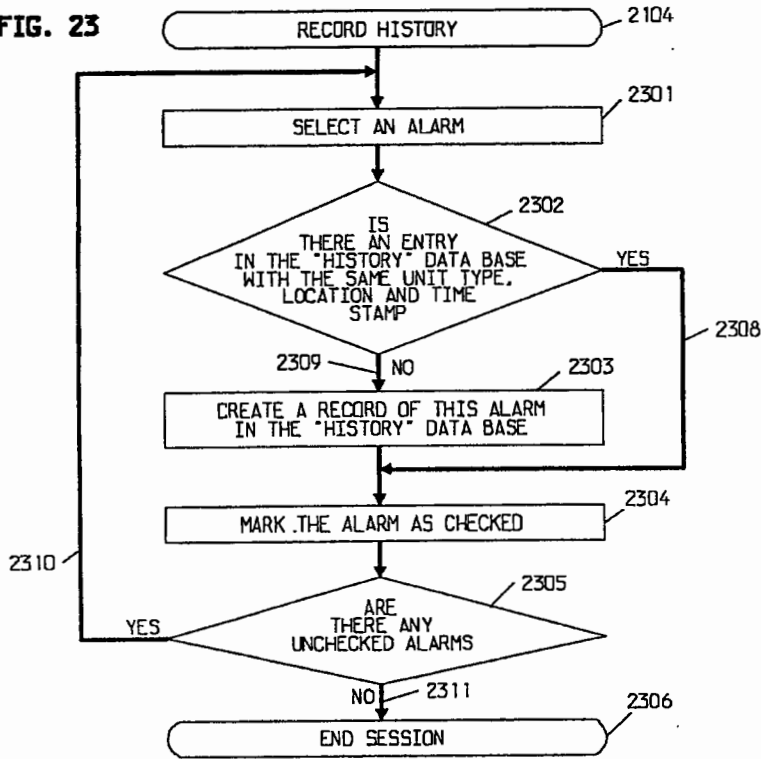


FIG. 24

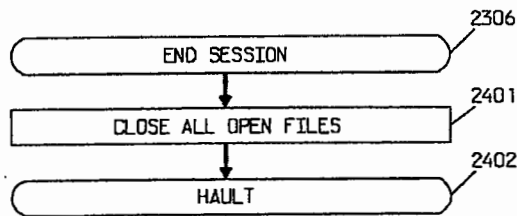


FIG. 25

SWITCH DATA

CUSTOMER : XYZ COMPANY
 RELEASE : R2V3
 VINTAGE : 1.2
 DUPLICATED: YES
 LOCAL TIME: 6:44 AM WEDNESDAY, JANUARY 18, 1989

2501

2502

2503

FIG. 26

ON-LINE PROCESSOR

ON-LINE PROCESSOR HAS 3 ALARM(S). 2604

FIG. 27

PROC 600 - ALARM CAUSES/ERROR LOG

UNIT TYPE	MOD	CAB	CAR	SLOT	CKT	ALARM STA	TOTAL FAIL	ENTRY INDEX	STAMP INDEX	DAY	HR	MIN	PROC REF
2	0	0	0	20	1	2	3	1	3	18	6	16	10
68	0	0	3	5	-	2	1	2	3	18	3	44	20
13	2	0	0	7	-	2	60	3	3	18	2	49	20

2701

2702

FIG. 28

PROC 610 - TAPE TESTS

EQUIPMENT										ALM NBR		FAULT		RUN		BRD		RESOLUTION
TI	CN	DA	XP	SV	PH	CT	STA	FAIL	IDX	CODE	DAY	HR	MIN	TAPE	TYPE	VINT		
-	-	-	-	-	-	-	4	1	1	949	18	6	16	29			UNCLEARED	
1	-	-	-	-	-	-	4	2	1	803	26	9	4	29			UNCLEARED	
-	1	-	-	-	-	-	2	3	1	925	18	6	12	29			UNCLEARED	

2801

FIG. 29

PROC 620 - NETWORK PROCEDURE

UNIT TYPE	MOD	CAB	CAR	SLOT	CKT	RMA STA	ALARM STA	CKT STA	INIT FL	FINAL CD	BRD TYPE	BRD VINT	RESOLUTION
68	0	0	3	5	-	-	2	1	1856	-	ANN11E	2	UNCLEARED
THIS DS-1 CIRCUIT REPORTS EXCESSIVE SLIPS.													
13	2	0	0	7	-	-	2	1	407	-	TN440B	2	CLEARED

FIG. 30

PROC 612 - INITIALIZATION CAUSES

UNIT TYPE	FAULT CODE	MEMORY BLOCK	ADDRESS BLOCK	DAY	HOUR	MINUTE	RELOAD COUNT	PROCESSOR HEALTH
3	8	0	771015	18	6	46	1	0

FIG. 31

REMAINING ON-LINE ALARMS

ON-LINE PROCESSOR HAS 2 ALARM(S) REMAINING.

PROC 600 - ALARM CAUSES/ERROR LOG

UNIT TYPE	MOD	CAB	CAR	SLOT	CKT	ALARM STA	TOTAL FAIL	ENTRY INDEX	STAMP INDEX	DAY	HR	MIN	PROC REF
2	0	0	0	20	1	2	3	1	3	18	6	16	10
68	0	0	3	5	1	2	1	2	3	18	3	44	20

FIG. 32

OFF-LINE PROCESSOR

OFF-LINE PROCESSOR HAS NO ALARMS.

FIG. 33

HARDWARE REPLACEMENTS

REPLACE TAPE CARTRIDGE.

FIG. 34

2:793::TAPE CARTRIDGE::
10:407:EXIST::
14:521:::IF BOARD REPLACEMENTS FAIL, TRY REPLACING THE IOBI-UPCI CABLE:
27:353.1:EXIST:70G (3/4 AMP) FUSE::

FIG. 35

2:940:::2:10000:IUPGRADE TO LATEST SOFTWARE DOT ISSUE.I:ALARM DISABLED.I
13:407-1:LOCATION:IEXISTING BOARD WITH TN440B V31:2:10000::IEXCESSIVE PARITY ERRORS.I
13:407-2:EXIST::2:10000::I EXCESSIVE PARITY ERRORS.I
16:219:EXIST::5:10000::IIGNORE IF COINCIDES WITH MODULE PROCESSOR FAILURE.I
24:286-1:LOCATION:IEXISTING BOARD WITH SN252 V31:3:10000::
24:286-2:EXIST::3:10000::

FIG. 36

555-555-5555,PBX.592,69,0,0,3,8,1,ON,2,573948,2/2/1989,12:9,283,-,-,ANN 17B,5,CLEARED,-
555-555-5555,PBX.592,69,0,1,1,18,1,OFF,2,613440,2/2/1989,0:24,-,-,-,-,UNCLEARED,-
555-555-5555,PBX.592,69,0,1,0,16,4,OFF,2,615671,2/2/1989,23:37,-,-,-,-,UNCLEARED,-
555-555-5555,PBX.1103,11,0,0,0,13,-,ON,1,574404,2/2/1989,12:47,84,-,-,TN445,1,CLEARED,-
555-555-5554,PBX.1296,32,0,2,1,20,1,ON,2,574020,2/2/1989,15,140,140,-,SN230B,1,UNCLEARED,-
555-555-5553,pbx.1310,56,4,0,0,3,-,ON,1,569393,1/29/1989,19:47,639,-,-,TN441,5,UNCLEARED,-

AUTONOMOUS EXPERT SYSTEM FOR DIRECTLY MAINTAINING REMOTE TELEPHONE SWITCHING SYSTEMS

REFERENCE TO A MICROFICHE APPENDIX

This application contains a microfiche appendix, designated A, which lists program instructions incorporated in the disclosed expert system. The total number of microfiche is 2 sheets and the total number of frames is 135.

TECHNICAL FIELD

This invention relates to maintaining computer systems and in particular to maintaining such systems using an expert system.

BACKGROUND OF THE INVENTION

Modern private branch exchanges (PBX) use a computer to control a switching network. PBXs are also referred to as customer switching systems or private automatic branch exchanges (PABX). In addition to controlling the PBX, the computer is continuously running basic diagnostic tests not only on itself but also on the switching network and communication facilities interconnecting the PBX to other PBXs and other types of computer systems. In addition to permanent faults/alarms, these diagnostic tests find many transitory faults within the PBX. The transitory faults may indicate that a component of the PBX is marginally faulty or that the PBX's environmental conditions have induced a failure in the PBX. Such environmental conditions result from a variety of sources ranging from error conditions on the communication facilities to electrical noise in the AC power supplied to the PBX at its site. Each fault occurring on a PBX must be investigated by a service technician to determine the severity of the fault. When a PBX manufacturer has thousands of PBXs to maintain in the field, the cost of making such investigations becomes enormous.

Some manufacturers have equipped their PBXs to report all faults to a centralized service reporting center. A technician at the service reporting center reviews the faults reports and then remotely accesses the PBX to determine the cause of the faults. Whereas the ability of a technician to remotely maintain PBXs is an improvement, the manufacturer still incurs considerable costs in maintaining PBXs in the field because of the labor cost of technicians.

Expert systems have been extensively used to assist in the maintenance of remote systems by directly supporting maintenance technicians. U.S. Pat. No. 4,697,243 discloses an expert system which assists technicians in the maintenance of elevators. In that system, an expert system running on a central computer leads an on-site technician through a diagnostic session with menus, questions, and directions displayed to the technician on a remote terminal. The technician communicates fault and test data to the expert system via the terminal. The expert system then diagnoses the elevator fault and sends the diagnosis back to the technician who then repairs the elevator.

U.S. Pat. No. 4,517,468 discloses an expert system executing on a central computer for collecting data from remote steam turbine generator power plants. After collecting the data from a plant, the expert system determines if a fault condition exists in that plant by using field knowledge incorporated into the expert sys-

tem. Upon detection of a fault condition, the expert system communicates the information to the plant operator with suggested actions to be taken. However, the expert system does not directly run any tests on the plant or alter the state of data within the plant. Further, the system requires a unique mechanism for accessing the data from the plant since the system cannot use the same means to gather data as used by technicians.

The problem with prior expert systems that diagnose fault conditions in remote computer systems, is that they require a human technician to determine the fault condition or to test and retire fault alarms in those systems. Also those expert systems require special mechanisms for gaining access to remote system data which add to the operating costs.

SUMMARY OF THE INVENTION

A technical advancement is achieved by an expert system that maintains remote computer systems by directly accessing the remote computer systems, diagnosing, and clearing fault conditions on those computer systems. The expert system performs those functions by first accessing a fault report from a centralized service reporting center, establishing a data connection to the computer system reporting the fault, invoking diagnostic routines on the computer system to gather data about the reported fault, analyzing the data, and, if appropriate, clearing the reported fault from the computer system. Advantageously, if the fault cannot be cleared, the expert system recommends maintenance procedures and replacement parts for a technician who the expert system dispatches to the remote computer. The recommendations are based on field experience stored in rules and databases maintained by the expert system. The centralized service reporting center receives faults or alarms directly from the computer systems, and the expert system accesses the reporting center via a digital link.

The expert system accesses the remote computer system in the same manner as a technician by placing a data call through the public telephone switching network. After gaining access to the remote computer system, the expert system invokes test procedures to obtain further data from the computer system and retires alarms representing transitory faults. When the remote computer system is controlling a customer switching system (PBX), the expert system only invokes testing procedures in the computer system which do not disrupt stable telephone calls. In addition, the expert system is capable of maintaining different vintages of the same PBX and identifies the vintage by interrogating each PBX.

In addition, the expert system maintains databases in which the results of previous accesses to any individual PBX are recorded. That information is continuously reused for each access to an individual PBX to diagnose alarms on that system and identify recurrent problems.

These and other features and advantages of the invention will become more apparent from the following description of an illustrative embodiment of the invention considered together with the drawing.

BRIEF DESCRIPTION OF THE DRAWING

FIG. 1 is a block diagram of a plurality of systems including a computer that executes an expert system embodying the principles of the invention for maintaining the illustrated PBXs;

FIGS. 2 through 24 illustrate, in flow diagram form, the logical flow of the expert system of Microfiche Appendix A;

FIGS. 25 through 33 provide an example of information displayed by the expert system;

FIG. 34 illustrates a portion of the SINGLE FAILURE database;

FIG. 35 illustrates a portion of the MULTI-FAILURE database; and

FIG. 36 illustrates a portion of the HISTORY database.

DETAILED DESCRIPTION

FIG. 1 illustrates expert system 102 which embodies the principles of the invention for performing maintenance on a plurality of PBXs (105 and 114) via public telephone network 113. Expert system 102 is executing on computer 100 which advantageously may be of the AT&T 6300 family of personal computers. PBX 114 and 105 (also referred to as customer switching systems) are telephone switching systems whose telephone switching network is controlled by a stored program computer. Within each PBX, the computer is constantly executing diagnostic routines checking for fault conditions. For example, within PBX 105, computer 122 is periodically running diagnostic routines to verify not only the state of switch 123 but also the state of the central office trunks such as DS-1 120, digital transmission facility (DCIU) 124 and tape unit 121. If a fault is detected with one of these units, PBX 105 records that fault. Such a fault is commonly referred to as an alarm or an alarm condition and results in a call being placed by PBX 105 via the public telephone network 113 to the service reporting center 104. Upon completion of the call, computer 122 transmits the alarm information to service reporting center 104 where it is recorded. The process of recording alarms by service reporting center 104 is referred to as generating a trouble report or trouble ticket.

Once service reporting center 104 has recorded the trouble ticket in its internal database, then either a human technician or expert system 102 accesses service reporting center 104 to obtain the trouble ticket. Either the technician or expert system 102 accesses PBX 105 via public telephone network 113 to run diagnostic procedures (PROCs) on computer 122 to perform a complete diagnosis of the state of PBX 105. The accessing of PBX 105 for this purpose is referred to as a session. For the AT&T System 85, detailed information on the operation of the PROCs is set forth in the manual entitled, "AT&T System 85, Release 2, Version 4, Maintenance."

After either the technician or expert system 102 has initiated a session with PBX 105, a listing of the alarms found by computer 122 is obtained and then PROCs are run to perform diagnostic tests and gather additional information concerning the nature of the alarms. Many alarms can be resolved through the execution of PROCs and do not require the replacement of any parts within PBX 105. After finishing the session with PBX 105, both the technician and expert system 102 generate a report indicating what alarms, if any, still exist on the system and a recommendation about the desirability of sending a technician to the site. Expert system 102 also recommends what spare parts may be needed to resolve the remaining alarms.

Consider now in greater detail the operations of expert system 102 in maintaining PBX 105. As illustrated

in FIG. 1, PBX 105 consists of switch 123, computer 122, tape unit 121, and DCIU 124 which interconnects PBX 105 to message center 125. The remainder of this description uses the following example to illustrate the operation of expert system 102. The example assumes that computer 122 has determined the existence of alarms with respect to computer 122, tape unit 121, and a data transmission facility such as DS-1 120 which are unit-type alarms 13, 2, and 68, respectively. These alarms are illustrated in FIG. 27.

Expert system 102 is constructed using a rule-based methodology. Such a methodology allows expert system 102 to represent units of knowledge in the form of rules which allows easy change of the knowledge represented in each rule without disturbing the rest of the system. A rule-based system consists of three components: working memory, rule memory, and an inference mechanism. The working memory describes the current state of the rule-based system, and moderates all communication between rules. If a rule needs to pass values to another rule, then it must do so through the working memory. The programmer declares items in working memory using a format that is similar to type-declaration information in standard programming languages.

The rule memory is a collection of rules. Each rule consists of a set of conditions and a set of actions. The programmer constructs the rules so that each represents a functionally independent and meaningful portion of the problem solution. In addition, the rule base has access to databases where additional knowledge and PBX specific history information is stored. An example of such databases is in expert system 102, the SINGLE FAILURE and MULTI-FAILURE databases.

In procedural languages, the sequence of program statements and explicit control statements determine execution order. In rule-based programming, the inference mechanism regulates the matching, selection and execution of rules. The inference mechanism is similar to an interpreter executing the following four-step loop:

(1) In the match phase, the inference mechanism collects all rules whose conditions match the current state in working memory.

(2) During the select stage, the inference mechanism selects the rule to be executed. If there is more than one rule that matches the current state, a process called conflict resolution specifies how rule priority is determined.

(3) In the act stage, the actions specified by the selected rule are executed. This results in modifications to the state of working memory.

(4) After the rules actions are executed, the inference mechanism again begins the match stage. This loop continues until no more rules match working memory, or until an explicit halt is encountered.

Expert system 102 is based on the C5 programming language which is similar to OPS5 programming language designed and built by Carnegie-Mellon University. Further details concerning C5 can be found in the article entitled, "Rule-Based Programming in the Unix® System," G. T. Vesonder, AT&T Technical Journal, Jan.-Feb. 1988, Volume 67, Issue 1. Expert system 105 is illustrated in C5 source language in Microfiche Appendix A.

Expert system 102 not only incorporates engineering and field experience within the rules of the program, but also in databases. In particular, the SINGLE FAILURE and MULTI-FAILURE databases are used to store recommendations on whether to replace parts

which have caused alarms within PBX 105. In addition, expert system 102 maintains ADMINISTRATION, HISTORY, and SWITCH databases. The ADMINISTRATION database contains information detailing how the different components are utilized within a PBX. The SWITCH database records information about configuration reported to expert system 102 by each particular PBX with which it has communicated. Finally, the HISTORY database contains a history of the alarms found and action taken for each PBX session. The HISTORY database is used to anticipate serious problems within a particular PBX and to gather additional field experience for later incorporation into the rules (see Microfiche Appendix A) and into the SINGLE FAILURE and MULTI-FAILURE databases.

Expert system 102 is executed by processor 106 in computer 100 as illustrated in FIG. 1. Programs are stored in memory 107 whereas the databases are stored in disk 108. Initially, controller 101 accesses service reporting center 104 via modem 111 and public telephone network 113. From service reporting center 104, controller 101 obtains the trouble ticket information for PBX 105. Expert system 102 then obtains the trouble ticket from controller 101. Expert system 102 then opens a session with PBX 105 by accessing PBX 105 via VMAAP 103, modem 110, and public telephone network 113. As described in greater detail with respect to FIG. 2, expert system 102 now obtains customer data and data concerning the functions of PBX 105. After obtaining this information, expert system 102 requests and obtains from PBX 105 the number of alarms presently existing on the PBX and detailed information about the unit-type and location of each alarm.

For the present example, this information is illustrated in FIG. 27. A unit-type 2 alarm indicates that there is a tape unit problem. A unit-type 68 alarm indicates that there has been an error on a data transmission facility such as DS-1 120, and a unit-type 13 alarm indicates a failure on a port data section of the memory of computer 122. As will be described in greater detail with respect to FIGS. 13 and 14, expert system 102 performs the appropriate PROCs with respect to tape unit 121 and determines that the unit-type 2 alarm cannot be cleared since the trouble/fault still exists on tape unit 121. Then by utilizing the data within SINGLE FAILURE database, expert system 102 recommends that a service technician be dispatched to PBX 105 with a new tape cartridge to replace the existing one. Expert system 102 next analyzes the unit-type 13 and 68 alarms by executing the appropriate PROCs and utilizing the SINGLE FAILURE and MULTI-FAILURE databases. Expert system 102 will successfully cause PBX 105 to recover from these two alarms.

In addition, during each session, expert system 102 performs preventive maintenance with respect to PBX 105 by determining whether computer 122 has undergone any initializations. Based on an examination of these initializations using the HISTORY database, expert system 102 will recommend whether a service technician should be dispatched to PBX 105 to perform specified service procedures which can include the part replacement.

FIG. 2 illustrates the initial setup and logging on to PBX 105 by expert system 102 via VMAAP 103. Initially, expert system 102 obtains information concerning the trouble ticket from controller 101. Upon obtaining this information, expert system 102 creates initial working memory elements using block 203 and opens the

necessary output files via block 204. An example of such a working memory element is the switch memory element which contains the information illustrated in FIG. 25 in lines 2501 and 2502. An example of an output file opened by block 204 is one used to store information such as illustrated in FIGS. 25 through 33. Block 205 initiates database table structures.

Using block 206, the expert system 102 instructs VMAAP 103 to place a call to PBX 105 in order to open the session with PBX 105. Block 206 transmits a command to VMAAP 103 which contains the necessary information to identify PBX 105. Expert system 102 waits in block 206 until it receives information back from VMAAP 103 indicating that the connection has been made. Expert system 102 then requests the status of the connection via block 207. Based on the status determined by block 207, decision block 208 determines if a connection has been made. If the connection status is "A1" indicating a successful login to PBX 105, then path 216 is followed to block 209. However, if the status is not "A1", block 210 is executed via path 215. Execution of block 210 indicates that expert system 102 was unable to log on to PBX 105 via VMAAP 103. This information then is displayed via block 211 (see FIG. 22).

If it was possible to log on to PBX 105, expert system 102 requests, in block 209, the customer data from PBX 105 via VMAAP 103. Upon receiving the customer data, block 209 parses this data so that it is in a usable form.

Next, expert system 105 determines the available modes of operation and switch parameters of PBX 105. This is accomplished in FIG. 3. Blocks 301 through 307 determine what modes are implemented and available. The modes include the administration, maintenance, and tape modes. The modes are required in order to avoid interaction problems with other entities that may also be doing work on the PBX, such as an on-site craftsperson. The administration mode allows administration of the different characteristics of the PBX such as which telephone numbers are assigned to which physical ports. The maintenance mode allows the different maintenance procedures to be executed. The tape mode allows the administration data stored on-line in the PBX's memory to be transferred to tape unit 121.

The first decision that must be made is whether the PBX 105 is of a version that requires the modes. This is done by block 301. Certain earlier versions of PBX 105 did not have modes since only one entity at a time could access the PBX. If the decision is made in decision block 301 that the PBX under test does have modes, block 302 requests the data on which modes are available. Next, block 303 checks if the maintenance mode is available. If the maintenance mode is not available, path 325 is followed to block 305 where the recommendation is set so that the message "maintenance mode not available" is displayed during the display recommendation portion of the session by block 211. If the maintenance mode is not available, expert system 102 must stop the session with PBX 105 at this point since it cannot proceed without itself setting the maintenance mode. If the maintenance mode is available, decision block 303 via path 326 checks whether the administration mode is available. If the administration mode is available, then block 307 via path 328 sets both the administration and maintenance modes. If the administration mode is not available, path 327 is followed from decision block 304 to block 308 which sets the maintenance mode only. If

the administration mode is not available, the session does not stop since expert system 102 can perform limited maintenance functions without the administration mode.

Blocks 310 through 321 determine whether computer 5 105 in PBX 105 is duplicated, e.g., has two processors. One processor is online and the other one is offline waiting to be brought online if the current online processor fails. If PBX 105 has duplicated processors, it is necessary to test both of them. Two Procedures (PROC) perform the duplication testing. These PROCs are detailed in the aforementioned AT&T Maintenance Manual. First, PROC 275 is executed by block 309. Decision block 310 checks the information returned by PBX 105 in response to PROC 275. If no error code is returned by PBX 105, path 332 is followed to decision block 316. If an error code of "3" is returned, expert system 102 determines that an error may have occurred, and block 311 once again executes PROC 275 in PBX 105. Decision block 313 checks the results of the execution of block 311; and if no error code is returned, path 334 is followed to decision block 316. If decision block 310 finds an error code other than "3" or if decision block 313 finds any error code, paths 330 or 335, respectively, are followed from these decision blocks to block 314. Block 314 executes PROC 613. The results of the execution of PROC 613 are checked by decision block 319. If no error code is returned, path 340 is followed to block 318 since this indicates that PROC 613 has determined that computer 122 is duplicated. If error code 74 is returned, path 342 is followed to block 317 since this indicates that PROC 613 has determined that computer 122 is not duplicated. If any other error code is returned, path 341 is followed to block 320 which indicates that the state of duplication is unknown.

If no error code was returned from the execution of PROC 275 in block 309, then path 332 is followed from decision block 310. Decision block 316 examines field 10 of the information returned by PBX 105 in response to PROC 275. This field indicates whether computer 122 is duplicated. If computer 122 is not duplicated, then block 317 marks it as such. If PROC 275 indicates that the processor is duplicated, block 318 marks the system as having a duplicated processor.

The information displayed in lines 2501 of FIG. 25 was obtained in block 209. Blocks 309 through 321 obtained line 2502.

FIG. 4 illustrates the additional administrative tasks that are performed before the testing of PBX 105 can commence. Blocks 401 through 409 check the SWITCH database to determine whether PBX 105 has had maintenance performed on it in the past by expert system 102. First, block 401 queries the SWITCH database to determine if it contains any data with respect to the PBX under test. Decision block 402 then determines the number of entries. If the PBX has not previously been encountered before, path 418 is followed to blocks 407 and 409 which build an entry in the SWITCH database for this PBX. If paths 419 or 420 are followed, the switch has previously had maintenance performed on it. However, what must still be determined is whether the PBX's configuration has changed; and if it has changed what the newest configuration is.

Path 420 is taken if the PBX has only one recorded configuration. If path 419 is followed indicating the existence of more than one past configurations, then it is necessary to determine the newest entry which defines

the latest configuration that expert system 102 has encountered with respect to PBX 105. After this has been determined, block 405 is executed which determines (on the basis of the information received and, as shown in FIG. 25 in lines 2501 and 2502) whether the present configuration of PBX 105 represents a change from its last recorded configuration. If there has been a change, block 407 via path 420 creates a new database entry for this PBX. If there has not been a change, path 421 is followed to decision block 410.

Blocks 410 to 413 are concerned with obtaining the local time maintained by PBX 105. The local time is important since PBX 105 may be located anywhere in the world, and the PBX alarms which will be discussed later are time-stamped relative to this local time. If decision block 410 determines that the administration mode has been set by expert system 102, path 423 is followed to blocks 412 and 413 which obtain the local time from PBX 105. Block 414 then displays the local time as indicated in line 2503 of FIG. 25. If the administration mode is not available, path 422 is followed to connector 416.

Having obtained the information defining the system parameters of PBX 105, expert system 102 now obtains an overall view of the maintenance condition of PBX 105 by execution of PROC 600. Execution of PROC 600 on PBX 105 obtains the number of alarms on the PBX and then requests are made for detailed information about each alarm. Block 501 transmits the PROC 600 request to PBX 105 via VMAAP 103. PBX 105 responds to this request with the number of alarms which are outstanding within the PBX. This number is read by block 502, and decision block 503 makes a determination of whether any alarms exist on PBX 105. If no alarms exist, path 522 is followed to block 506 which proceeds to check the off line processor, if any, for alarms.

If alarms exist, then path 523 is followed to block 504 which displays lines 2701 of FIG. 27. Expert system 102 obtains information concerning each of these alarms by the repetitive transmission of the "next circuit" command of PROC 600 by block 507. As information about each alarm is received, it is displayed by block 508 to create each line in lines 2702 of FIG. 27. Warning alarms and the trunk software alarms identified by decision blocks 510 and 512, respectively, are immediately cleared by block 511 via paths 524 and 527, respectively. The sequence of block 504 through 512 is terminated by decision block 514 after information on all the alarms has been obtained. As long as there is still an outstanding alarm, path 530 is followed back to block 507. After information about all alarms has been obtained from PBX 105, path 531 is followed to decision block 515. Blocks 515 through 518 account for the fact that DCIU 124 and tape unit 121 within PBX 105 can each have multiple alarms. Since the diagnostic tests performed for these units are complete, it is desirable only to perform each test once per session. Hence, blocks 515 through 518 remove all but at most one alarm for DCIU 124 and tape unit 121.

There are three overall goals that must be achieved in the execution of each of the diagnostic PROCs. First, expert system 102 determines the failures that caused each alarm by executing diagnostic routines on PBX 105. Secondly, expert system 102 generates a report detailing the results of the diagnostic routines and indicating the remaining alarms on the system. Finally, expert system 102 provides recommendations for re-

placing hardware on PBX 105 if necessary. An example of such a recommendation is illustrated in FIG. 33 where it is recommended to replace the tape cartridge of PBX 105.

FIG. 6 is concerned with the order in which the alarms obtained in FIG. 5 should be processed. The aforementioned AT&T Maintenance Manual indicates that the alarms should be processed in the order in which they are received upon execution of PROC 600. This order is given by the entry index number as illustrated in FIG. 27. However, field knowledge obtained and incorporated into expert system 102 indicates that if both unit-type 11 and 23 alarms are encountered, then the alarm with a unit type of 23 should be processed first. Decision block 601 and block 602 accomplish this. Similarly, experience has shown that if unit-type 55, 56 and 57 alarms are present together, then the alarms should be processed in descending order by unit type (i.g. 57 then 56 then 55). This is accomplished by blocks 603 and 605. Otherwise, if none of these special cases are encountered, block 606 simply chooses the unprocessed alarm with the highest index number.

Decision block 607 results in the execution of the diagnostic PROCs detailed in FIGS. 7, 9, 11, 12, 13, and 15. It is important to remember expert system 103, whose overall logical flow is illustrated in FIGS. 2 through 24 is implemented in a rule-based language (see Microfiche Appendix A.) For more complete details on how each diagnostic routine is implemented, the code starting at page 36 and entitled, "Referred PROCs" in Microfiche Appendix A should be examined.

The following is a description of the operation of each of these PROCs. Note that in our present example alarm information received from PBX 105 as illustrated in FIG. 27 indicates the existence of alarms of unit-types 2, 68, and 13. Unit-type 2 alarms indicate a problem with the tape unit. Unit-type 68 alarms indicate that there has been an error on a data transmission facility such as DS-1 120. Unit-type 13 alarms indicate a failure in the port data section of the memory of computer 122. However for unit-type 13 alarms, field experience incorporated into expert system 102 has shown that a variety of different equipment failures within PBX 105 may result in that type of alarm. Those failures will be further detailed during the discussion of unit-type alarm 13.

FIG. 13 shows the logical operations performed by expert system 102 in response to a unit-type 2 alarm. This alarm indicates that an error has been encountered on tape unit 121. Decision block 1301 represents the logical select stage of rule-based expert system 103 for a unit-type alarm 2.

Logical blocks 1302 through 1306 are concerned with versions of PBX 105 which require modes. If PBX 105 is of a non-mode version, then path 1315 is followed to block 1307. If modes are required, then blocks 1303 and 1304 determine whether the tape mode is available to expert system 102. If the tape mode is not available, path 1317 is followed from decision block 1304 to block 1305 which marks the alarm as "uncleared." Marking the alarm as unclear causes a message indicating that fact to be printed. If the tape mode is available, then block 1306 is executed via path 1318 to set the tape mode.

Because of the nature of tape unit 121, failure status information only is collected from this unit and no diagnostic tests are actually run on it. However, as will be illustrated later, a recommendation may be made to

replace the tape cartridge. In response to the execution of PROC 610, decision block 1308 determines whether tape unit failures have occurred. If no tape unit failures are outstanding, blocks 1309 and 1310 are executed via path 1320. These blocks note that the alarm has been cleared, and a command is sent to PBX 105 to clear the indication in the PBX. In FIG. 14, blocks 1401 through 1403 collect all the failures associated with the unit-type alarm 2 on PBX 105.

FIG. 7 illustrates the logical flow performed in utilizing PROC 620 to determine the cause of a network alarm. First, logical decision block 701 ascertains whether there are any such alarms. If there is an outstanding network alarm, then control is passed to block 703 via path 729. Block 703 first obtains the unit-type and location of the failing network unit by execution of PROC 620. Decision blocks 705 through 709 check whether there are special cases which make it undesirable to execute the diagnostic portion of PROC 620. Decision block 705 determines whether there are intermodule calls that could be dropped if the diagnostic portion of PROC 620 is executed. If intermodule calls could be dropped, control is transferred to block 710 via path 716. Decision block 706 checks whether the failing network unit is the attendant console interface (unit-type 44). If the attendant console interface is failing, this test cannot be performed since to properly perform the test, the attendant console headset must be unplugged which requires a craftsman on site. If the unit failing is unit-type 44, path 718 is followed to block 710; if not, path 719 is followed to decision block 707. The latter decision block checks whether the alarm is of unit-type 45 which indicates an auxiliary trunk circuit pack. In order to test that circuit pack, the DIP switches must be set to a particular setting which is impossible to verify remotely. If the alarm is of unit-type 45, path 720 is followed to block 710, otherwise path 721 is followed to decision block 708.

Decision block 708 determines whether the failing circuit pack has been marked as "maintenance busy" indicating that there is a craftsman on site performing maintenance tests on this particular circuit pack. If the circuit pack is marked as maintenance busy, path 722 is followed to block 710; otherwise, path 723 is followed to decision block 709. Decision block 709 checks a number of special situations where stable calls could be dropped/disconnected if the diagnostic portion of PROC 620 is executed. If stable calls could be dropped, path 724 is followed to block 710 since it is undesirable to perform a test that could potentially drop calls. If the testing would cause no stable calls to be dropped, path 725 is followed to connector 712. Block 710 marks the alarm as "uncleared," which terminates the session. Connector 711 interconnects to a portion of the program which obtains the board type. This portion is executed so that the board can be checked to insure that it is of the proper vintage and is properly administered.

FIG. 8 illustrates another special case where the diagnostic portion of PROC 620 often cannot be run. Unit-type 68 alarms indicate a failing DS-1 trunk unit such as unit 120 which has 23 separate channels, each capable of carrying a telephone conversation. Since the probability is extremely high that at least one of these channels will be active at any given point in time, a non-invasive PROC (PROC 625) is used to investigate the status of this particular facility. Decision block 801 checks if the failing facility is of unit-type 68; and if it is, path 815 is followed to block 802. Blocks 802 through 811 utilize

PROC 625 to check the status of the failing DS-1 trunk unit. If the failing facility is not of unit-type 68, then path 816 is followed to block 805 which executes the diagnostic portion of PROC 620.

After execution of the diagnostic portion of PROC 620, decision block 803 checks if the alarm status after execution of block 805 indicates that the alarm had been cleared. If the alarm has been cleared, path 817 is followed to block 812. If the alarm has not been cleared, path 820 is followed to decision block 804. Decision block 804 checks whether the alarm after execution of PROC 620 has changed from a unit-type 56 or 57 alarm (intramodule data store or light guide interface fault) to unit-type 23 alarm (duplication channel fault.) If this change has occurred, field experience has found that the unit-type 23 alarm must first be cleared before any other alarms can be processed. The unit-type 23 alarm is cleared by following path 821 to decision block 806 which via path 818 re-executes PROC 620 on the duplication channel via block 805.

If there has not been a change in the unit-type of the alarm, decision block 804 transfers control to block 807 via path 822. Block 807 marks the alarm as "uncleared" and transfers control to connector 711 to obtain the board type. This is done so that a replacement recommendation can be made.

After execution of PROC 625 by block 802, decision block 809 is executed to determine whether the test indicated that the DS-1 trunk facility is failing. If the facility indicates no failures, then block 811 is executed to clear the alarm in PBX 105 via path 823. If the facility indicates a problem, path 822 is followed to block 810 which records the problem. The information recorded in block 810 is utilized to print the information of FIG. 29, line 2902.

In the present example, FIG. 27 illustrates the results of executing PROC 600 to obtain the initial alarms of PBX 105. The third line of block 2702 indicates that there is another failure (unit-type 13 alarm, port data storage unit) which requires the execution of PROC 620. When PROC 620 is executed to investigate this particular alarm at decision block 805, path 816 is followed to block 803 which checks the result. Then, since the port data unit in the present example shows no failures, control follows path 817 to block 812. Block 812 marks this alarm as "cleared" and transfers control to the "get board-type" procedures via connector 711. Therefore, the results of executing PROC 620 as displayed in line 2903 of FIG. 29 indicate that the unit-type 13 alarm has been cleared.

FIG. 9 illustrates the logical flow for PROC 650. This PROC is used to test DCIU 124 of PBX 105. This data transmission facility has eight distinct data links. Each data link interconnects PBX 105 to a computer or communications systems. Examples of such systems are voice mail and message center systems. PBX 105 is connected only to message center 125. First, PROC 650 (test 1) is utilized to determine which of these data links is failing. This information is read from PBX 105 utilizing block 904. Block 905 is then utilized, to execute the diagnostic portion of PROC 650 (test 2) to perform transmission testing on that data link. Decision block 908 then checks if the results of the diagnostic portion of PROC 650 indicate that the transmission test was successful. If so, then blocks 909 and 910 are executed via path 922. If the transmission test failed, path 923 is followed to decision block 911. The latter decision block determines if the fault code returned by the exe-

cution of the diagnostic portion of PROC 650 is between 19 and 36. If the returned code is in that range, then field experience has shown that PROC 650 should be re-executed since those alarms may clear themselves.

This is performed by following path 924 to block 916. If the returned fault code is not in that range, path 925 is followed to block 912. After re-execution of PROC 650 in block 916, decision block 917 determines whether the alarm has been cleared on the second pass. If the alarm has been cleared, path 929 is followed to block 918 which performs a function similar to that performed by block 909. If the alarm has not been cleared, decision block 917 transfers control via path 928 to block 912. This latter block in conjunction with decision block 913 utilizes the "next fault" command of PROC 650 to obtain all of the outstanding fault information. Once all the fault information has been collected, path 931 is followed to block 914 which marks the alarm as "uncleared" and transfers to connector 910.

FIG. 10 is a continuation of FIG. 9. Blocks 1001 through 1007 obtain information to make up a report similar to FIG. 28. Block 1001 is used to execute PROC 256 to determine additional information about the data link. Decision block 1002 determines whether an error has occurred during the execution of PROC 256. If there is an error, path 1011 is followed to connector 702 which results in error processing as illustrated in FIG. 7. If an error did not occur, path 1012 is followed to block 1003. Block 1003 once again uses PROC 256, but this time specifies the failing data link. Further information about that link is obtained using the display portion of PROC 256 in block 1004. After execution of this block, the specified information is read using block 1005. Then block 1006 reads the translation tables in the memory of PBX 105 to determine the nature of the applications assigned to the logical channels of the indicated data link to PBX 105. For example, the same computer can run applications to function either as a message center or as a telemarketing center. Each application is assigned one or more logical channels on the physical data link between the computer and PBX 105. After executing block 1007, connector 811 transfers control to the procedures that obtain the board type.

FIG. 11 illustrates the logic flow of PROC 601. This PROC obtains information concerning the units in PBX that control the physical environment, e.g., fans, power supplies, battery back-up units, etc. Decision block 1101 first determines whether this PROC is to be run. If so, path 1111 is followed to block 1103 which executes PROC 601. Decision block 1104 checks if the alarm has been marked clearly by PBX 105. If it has, path 1112 is followed to block 1105 and from there to connector 702. If the alarm is not cleared, path 1113 is followed to block 1106. The latter block takes the information returned from PBX 105 as a result of the execution of PROC 601 and computes a pseudo-fault code that summarizes information concerning the failure that was identified. This pseudo-fault code records the possible multiple causes of the alarm in the history database. After execution of block 1106, block 1107 is executed to mark the alarm as unclear, and control is passed to connector 702.

FIG. 12 illustrates the logical flow of PROC 622. This latter PROC tests peripheral equipment attached to PBX 105, such as telephone stations and data terminals. Block 1202 executes the diagnostic portion of PROC 622 after providing the type and location of the failing peripheral equipment. PROC 622 runs the diag-

nostic portion of the test several times on the failing peripheral unit to fully evaluate the state of the unit. Decision block 1203 determines whether the unit failed under test. If the indicated unit did not fail, path 1213 is followed to block 1204 which marks the alarm as cleared. If the unit failed under test, control is transferred via path 1214 to block 1205. Blocks 1205 and 1206 obtain detailed results of the multiple diagnostic tests performed on the peripheral unit. After all this information has been obtained, path 1216 is followed to block 1207 which marks the alarm as uncleared and control is transferred to connector 711.

FIG. 15 illustrates the logical flow of PROC 612. This latter PROC is executed every time that expert system 102 contacts a PBX such as PBX 105. PROC 612 interrogates the PBX to determine whether the PBX has undergone any software initializations. For example, software initializations occur if the program executed by PBX 105 is repeatedly interrupted by a parity error or programming problem. The information obtained by PROC 612 is utilized to predict in advance whether a particular PBX is approaching a critical point where it may have a severe service outage. If such a situation is detected, a craftsman may be dispatched to prevent an actual outage from occurring.

Block 1501 executes PROC 612; and block 1502 obtains information about any initialization causes appearing in the PBX's log. Decision block 1503 looks at the resulting fault codes to determine their seriousness. If a serious fault code is detected, then path 1508 is followed to block 1504. This latter block makes a record to track these conditions in the INIT database maintained by expert system 102. An example of a minor fault is an on-site craftsman who simply stopped the PBX processor to perform maintenance functions. Decision block 1505 insures that all initialization log entries have been checked. After all entries have been checked, decision block 1505 transfers control via path 1511 to connector 1506.

FIG. 16 illustrates the logical flow for obtaining the board or circuit pack type. PROC 600 may provide incomplete information concerning the location of the failing circuit board. Block 1601 determines whether the location of the failing unit is completely known. If the location of the failing circuit board is not completely known, control is transferred to connector 702. If the location is known, decision block 1602 is executed to determine whether the administration mode has been set. If the administration mode has not been set, then the board type cannot be determined since this information is stored within PBX 105 and the administration mode is necessary to obtain that information. If the administration mode has been set, then block 1603 is executed. The latter block executes PROC 290 on PBX 105 to obtain the board type and board vintage from the PBX. This information is read by block 1604. Next, the ADMINISTRATION database of expert system 102 is interrogated to ascertain whether the board is correctly administered on PBX 105. The decision of whether the administration is correct is performed in decision block 1607. If the board is incorrectly administered, block 1608 is executed to highlight this fact so that a craftsman can readminister the board within PBX 105. Lastly, control is passed to connector 702.

FIG. 17 illustrates the logical flow for printing the information gathered by the diagnostic PROCs for this session with PBX 105. See FIGS. 28 through 30 for an example of this printed information. Blocks 1701

through 1705 illustrate the logical flow for printing the information for PROC 620. Block 1701 determines if there is any information to be displayed for PROC 620. If there is no information to be displayed, path 1710 transfers control to block 1706. If there is information to be displayed, control is transferred to block 1702 via path 1711. The latter block prints the display header and then transfers control to blocks 1703 through 1705. These blocks display the data on 2901 through 2903 of FIG. 29. Similar logical flow as used by blocks 1702 through 1705 is utilized in block 1706 for printing information gathered by the execution of PROCs 650, 601, 622, 610, and 612 provide for PROC 620. After execution of block 1706, control is transferred to connector 1707.

FIG. 18 illustrates a check for remaining alarms and the display of alarm information which, for the present example, results in the generation of FIG. 31. The alarm state of the PBX is again checked to ensure that it is consistent with the text results seen by expert system 102. In the present example, unit-type alarm 13 should be cleared; however, unit-type alarms 2 and 68 should still remain on PBX 105. Block 1801 executes PROC 600 to interrogate PBX 105 regarding alarms existing on that PBX. Decision block 1802 checks if there are any remaining alarms. If there are no remaining alarms, control is passed to connector 1808 via path 1811. If there are remaining alarms, block 1803 is executed via path 1812. This latter block displays the header information illustrated in FIG. 31 for the present example. Blocks 1805 through 1807 then determine what alarms are present on PBX 105 and display this information as illustrated in FIG. 31. After all the alarms have been displayed, control is passed via path 1814 to connector 1808.

FIG. 19 illustrates the logical flow of checking the off-line processor of PBX 105. First, decision block 1901 checks if computer 122 is duplicated. If it is not, path 1915 is followed to connector 1907 which executes the program segment which recommends which hardware, if any, should be replaced on PBX 105. If computer 122 is duplicated, decision block 1903 is executed via path 1916. Before maintenance, administration, and tape functions can be accessed in this PBX, decision block 1903 checks whether PBX 105 requires mode permission. If modes are not required, path 1917 is followed to block 1909. If modes are required, block 1904 is executed via 1918. Block 1904 requests the mode data from the off-line processor of PBX 105. Note that the mode data for the on-line processor of PBX 105 was previously obtained in FIG. 3. Decision block 1905 checks whether the off-line maintenance mode is available. If the maintenance mode is not available on the off-line processor of PBX 105, then control is transferred via path 1919 to block 1906. This transfer results in the recommendation being set to display the fact that the off-line maintenance mode is not available. Then, control is passed to connector 1907. If decision block 1905 determines that the maintenance mode is available, control is transferred via path 1920 to block 1908 which sets the maintenance mode in PBX 105. Next, block 1909 executes PROC 600 and block 1910 obtains the number of outstanding alarms on the off-line processor of PBX 105. Decision block 1911 determines whether there are any outstanding alarms on the off-line processor of PBX 105. If there are no outstanding alarms, control is transferred to connector 1907 via path 1921 so that the "recommend hardware" portion of the pro-

gram can be executed. If there are alarms on the off-line processor, then control is transferred via path 1922 to connector 1912 so that these off-line alarms can be displayed and eventually the "recommend hardware" portion of the program is executed.

FIG. 20 illustrates the logical flow for displaying the results of the PBX 105 interrogation performed by the logical flow illustrated in FIG. 19. The logical flow of blocks 2001 through 2003 and block 2006 is similar to that of FIG. 17. The difference between FIGS. 17 and 20 is the actions taken by blocks 2004 and 2005. These two blocks determine if any alarms noted on the off-line processor of PBX 105 had been previously encountered during testing of PBX 105's on-line processor. If an alarm had been previously found during testing of the on-line processor, then that alarm is cleared in the off-line processor by block 2005 since the alarm probably resulted from an unduplicated portion of the system which was reported to both processors. After all of the off-line alarms have been displayed, control is transferred to connector 1907 via path 2013. The present example assumes that the off-line processor of PBX 105 had no alarms and results in the display illustrated in FIG. 32.

FIG. 21 illustrates the logical flow of the portion of the program that determines what replacement parts, if any, should be recommended. A service technician is dispatched to take those parts and to perform the necessary maintenance on PBX 105 to clear the alarms remaining after expert system 102 has finished the session with PBX 105. First, decision block 2101 checks if there are any environmental alarms. If there are environmental alarms, then control is transferred to block 2102 via path 2111. The reason is that an environmental alarm in a cabinet often results in spurious reports of other hardware failures within that cabinet. An example of an environmental condition is an over temperature alarm. When circuit packs are operated outside of their recommended operating temperature range, the packs exhibit error conditions that disappear when normal conditions are restored. Therefore, no hardware recommendations are made for replacement of boards operating under these conditions.

After block 2102 has been performed or if there are no environmental alarms, decision block 2103 is executed. The latter decision block determines whether there are any remaining alarms that are not in a cabinet exhibiting environmental alarms. If there are no such alarms, then control is transferred to connector 2104 via path 2113 and no recommendations will be made. If there are alarms which are not in a cabinet that has an environmental alarm, control is transferred to block 2105 via path 2114. Block 2105 picks a particular alarm. The present example uses the unit-type 2 alarm. Next, decision block 2106 checks if this alarm has an entry in the SINGLE FAILURE database illustrated in FIG. 34 for the present example. Since in the present example the unit-type 2 alarm, indicating tape unit 121, with a fault code of 925 is found within this database, control is passed via path 2116 to block 2107 which displays the recommendation from the SINGLE FAILURE database. In the present example, the recommendation is that the tape cartridge should be replaced (see FIG. 33.) For a unit-type alarm and fault code not found in the SINGLE FAILURE database, control is transferred to connector 2108 via path 2115.

FIG. 22 illustrates the logical flow for checking whether the alarm condition is a transient one that has

occurred enough times to warrant a hardware replacement. First, decision block 2201 checks if the unit-type alarm and fault code appears in the MULTI-FAILURE database, a sample entry of which is illustrated in FIG.

35. If there is an entry for the alarm under investigation within the latter database, path 2211 is followed to decision block 2202. The latter decision block first obtains from the HISTORY database the number of occurrences and time period of this particular alarm in PBX 105. This information is compared the MULTI-FAILURE database record to determine if there have been enough identical failures within the specified time interval to exceed the threshold set in the MULTI-FAILURE database. If this threshold is exceeded, then block 2203 is executed via path 2214. Block 2203 displays the replacement equipment recommendations obtained from the MULTI-FAILURE database. Blocks 2204 through 2206 governs a condition which field experience has shown to require special handling. The situation arises when multiple unit-type 13 alarms indicating failure of several port data store units appear within the same module. This condition does not indicate that the circuit packs containing the port stores should be replaced but rather that the time slot interchange arithmetic logic unit in this module is at fault and should be replaced. If decision block 2204 finds that the alarm is a unit-type 13 alarm, then decision block 2205 via path 2216 checks if other unit-type 13 alarms exist in the same module. If multiple alarms of this unit-type exist, then block 2206 is executed via path 2219. The latter block displays the recommendation that the time slot interchange arithmetic logic unit should be replaced. Finally, block 2207 is executed which marks the alarm as having been checked for a recommendation; and control is then transferred to connector 2211.

FIG. 23 illustrates the logical flow for updating the HISTORY database which contains information on the alarms handled by expert system 102 on PBX 105. To be included in the HISTORY database, the new alarm must have a time of occurrence distinct from any other occurrence of the same type and for the same facility already recorded in the HISTORY database. This time of occurrence is determined by the time stamp information received from PBX 105 by PROC 600 and is displayed in FIG. 27 in lines 2702 under the day, hour and minute columns. If decision block 2302 determines that a new entry should not be created, path 2308 is followed to block 2304. If a new entry is required, block 2303 is executed via path 2309. Block 2303 creates a new record for this alarm in the HISTORY database. If a particular component or circuit pack is failing routinely, then there will be multiple entries for that unit within the HISTORY database. Block 2304 marks the alarm as checked and decision block 2305 determines whether there are any remaining unchecked alarms for this session. If there are no remaining alarms, then path 2311 is followed to connector 2306.

FIG. 24 shows the final steps performed to end the current PBX 105 session. Block 2401 indicates that all files are closed and the proper steps taken to exit from this session by expert system 102.

It is to be understood that the above-described embodiment is merely illustrative of the principles of the invention and that other arrangements may be devised by those skilled in the art without departing from the spirit and scope of the invention.

We claim:

1. A method for remotely maintaining computer systems by an expert system in conjunction with a central reporting center to which said computer systems report self-detected faults, comprising the steps of:

accessing said central reporting center by said expert system to obtain the identity of one of said computer systems reporting detected faults; 5
opening a maintenance session with said identified computer system by said expert system via the public telephone network in a similar manner as a human technician; 10
invoking diagnostic procedures on said identified computer system by said expert system to gather data about said reported faults; 15
analyzing said data to determine the severity of each of said reported faults by said expert system with respect to said reported faults being transitory and permanent type of faults; and
clearing transitory ones of said reported faults by said expert system upon said transitory ones of said reported faults being determined to be less severity than permanent ones of said reported faults. 20

2. The method of claim 1 further comprising the step of establishing a plurality of databases containing field experience on components in each of said computer systems concerning replacement of said components; 25

interrogating said plurality of databases for each of said permanent faults to determine when to replace components in said identified computer system; and 30
displaying a message defining each of said components to be replaced.

3. The method of claim 2 further comprises the step of interrogating said plurality of databases by said expert system to determine the number of fault occurrences of each component having a transitory fault in said identified computer system; 35

interrogating said plurality of databases to determine whether said number for each of said components exceeds a predefined threshold; and 40
recommending replacement of each of said components whose number exceeds said predefined threshold.

4. A method for remotely maintaining telephone switching systems by an expert system in conjunction with a central reporting center to which said switching systems report self-detected faults, comprising the steps of: 45

accessing said central reporting center by said expert system to obtain the identity of one of said switching systems reporting detected faults; 50
opening a maintenance session with said identified switching system by said expert system via the public telephone network in a similar manner as a human technician; 55
invoking diagnostic procedures on said identified switching system by said expert system to gather data about said reported faults; 60
analyzing said data to determine the severity of each of said reported faults by said expert system with respect to said reported faults being transitory and permanent type of faults; and
clearing transitory ones of said reported faults by said expert system upon said transitory ones of said reported faults being determined to be less severity than permanent ones of said reported faults. 65

5. The method of claim 4 further comprising the step of establishing a plurality of databases containing field

experience on components in each of said switching systems concerning replacement of said components;

interrogating said plurality of databases for each of said permanent faults to determine when to replace components in said identified switching system; and

displaying a message defining each of said components to be replaced.

6. The method of claim 5 further comprises the step of interrogating said plurality of databases by said expert system to determine the number of fault occurrences of each of said components having a transitory fault in said identified switching system; 60

interrogating said plurality of databases to determine whether said number for each of said components exceeds a predefined threshold; and
recommending replacement of each of said components whose number exceeds said predefined threshold.

7. The method of claim 6 wherein said identified switching system has a control computer including duplicated processors with one of said processors actively controlling said identified switching system and the other of said processors being in a standby condition, the method further comprising the step of testing said other processor in said standby condition.

8. The method of claim 7 wherein said switching systems are of different manufactured vintages and said invoking step comprises the step of requesting from said identified switching system the vintage of said identified switching system thereby being able to utilize the correct diagnostic procedures.

9. A method for remotely determining replacement of components in telephone switching systems by an expert system in conjunction with a central reporting center to which said switching systems report self-detected faults, comprising the steps of: 65

maintaining a history database of said expert system to record detected faults by component type and component location and time of fault for each of said switching systems;

maintaining a multifault database to store on the basis of field experience recommendations on the replacement of components in said switching systems on the basis of component type and component location and time of fault by said expert system;

accessing said central reporting center by said expert system to obtain the identity of one said switching systems reporting detected faults;

opening a maintenance session with said identified switching system by said expert system in a similar manner as used by a human technician;

invoking diagnostic procedures on said identified switching system by said expert system to gather data about said reported faults and the gathered data including component type and component location and time of fault;

analyzing said data to determine the severity of each of said reported faults by said expert system with respect to being transitory and permanent types of faults;

interrogating said history database with the gathered data by said expert system to determine the number of fault occurrences of each of said components having a transitory fault in said identified switching system; and

19

20

recommending replacement of each of said components having said number that exceeds a predefined threshold in said multifault database.

10. The method of claim 9 further comprising the step of maintaining a single fault database to store on the basis of field experience recommendations on the replacement of components for permanent type faults on the basis of component type and component location by said expert system;

interrogating said single fault database by said expert system for each of said components having a permanent fault in said identified switching systems; and

recommending replacement of each of said components found in said single fault database.

11. The method of claim 10 wherein said step of recommending comprises the step of displaying information to dispatch a service technician to replace the recommended components.

12. The method of claim 11 wherein said step of opening comprises the steps of dialing a connection via a

public telephone network to said identified switching system; and

logging on to said identified switching system by said expert system.

13. The method of claim 12 wherein said interrogating step of said history database comprises the step of clearing each of said transitory fault not having an occurrence in said history database.

14. The method of claim 13 wherein said identified switching system has a control computer including duplicated processors with one of said processors actively controlling said identified switching system and the other of said processors being in a standby condition, the method further comprising the step of testing said other processor in said standby condition.

15. The method of claim 14 wherein said switching systems are of different manufactured vintages and said invoking step comprises the step of requesting from said identified switching system the vintage of said identified switching system thereby being able to utilize the correct diagnostic procedures.

* * * * *

25

30

35

40

45

50

55

60

65



US005535338A

United States Patent [19]
Krause et al.

[11] Patent Number: 5,535,338
[45] Date of Patent: Jul. 9, 1996

[54] MULTIFUNCTION NETWORK STATION
WITH NETWORK ADDRESSES FOR
FUNCTIONAL UNITS

5,379,289 1/1995 DeSouza et al. 370/85.13

FOREIGN PATENT DOCUMENTS

0222584A2 5/1987 European Pat. Off. .
0522743A1 1/1993 European Pat. Off. .

[75] Inventors: Jeffrey Krause, Los Altos; Niles E.
Strohl, Tracy; Michael J. Seaman, San
Jose; Steven P. Russell, Menlo Park;
John H. Hart, Saratoga, all of Calif.

Primary Examiner—Alpesh M. Shah
Attorney, Agent, or Firm—Haynes & Davis

[73] Assignee: 3Com Corporation, Santa Clara, Calif.

[57] ABSTRACT

[21] Appl. No.: 452,498

[22] Filed: May 30, 1995

DLL devices are built with multiple MAC address instead of a single MAC address, and provide a multiple virtual DLL interfaces to the upper layers (3-7) in a computer. This results in a new class of multi-function computers for attachment to a network system which take advantage of the multiple virtual DLL interfaces, to increase performance of the respective functions executed by the computer. Thus, a new network interface control apparatus and a new class of multi-function computer systems for attachments to networks are provided. The memory in the medium access control device stores a plurality of additional network addresses in addition to the assigned network addresses. The address filtering logic includes circuits responsive to the additional network addresses, such as logic for blocking a particular frame on at least one of the plurality of data channels when the source and destination address of a particular frame are found in the additional addresses stored in the memory. The plurality of data channels served by the media access control device may reside on a single physical interface, or in independent physical interfaces as suits the needs of a particular design. A high performance design would include independent buffering and queuing structures for each of the data channels. An alternative design may include shared buffering and queuing structures for a plurality of functional modules in the connected computer which have independent side network addresses.

Related U.S. Application Data

[62] Division of Ser. No. 98,616, Jul. 28, 1993.

[51] Int. Cl.⁶ 709/222 G06F 13/00

[52] U.S. Cl. 395/200.20; 395/800; 370/94.3;
364/228.5; 364/241.9; 364/242.95; 364/DIG. 1

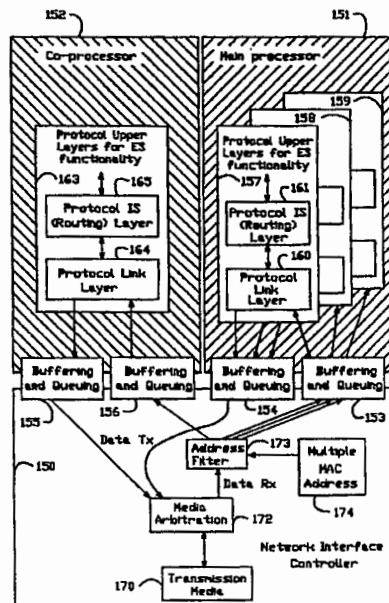
[58] Field of Search 395/200.01, 200.11,
395/200.16, 200.20, 287, 728, 800, 829,
858; 370/54, 60, 92, 94.3; 371/8.2, 11.2;
340/825.06, 825.07

[56] References Cited

U.S. PATENT DOCUMENTS

4,652,874	3/1987	Loyer	340/825.05
4,692,918	9/1987	Elliott et al.	370/85
4,930,123	5/1990	Shimizu	370/94.1
5,058,110	10/1991	Beach et al.	370/85.6
5,058,163	10/1991	Lubarsky et al.	380/49
5,095,381	3/1992	Karol	359/123
5,148,433	9/1992	Johnson et al.	371/11.3
5,307,413	4/1994	Denzer	380/49
5,319,752	6/1994	Petersen et al.	395/250
5,321,819	6/1994	Szczepanek	395/200.2

23 Claims, 16 Drawing Sheets



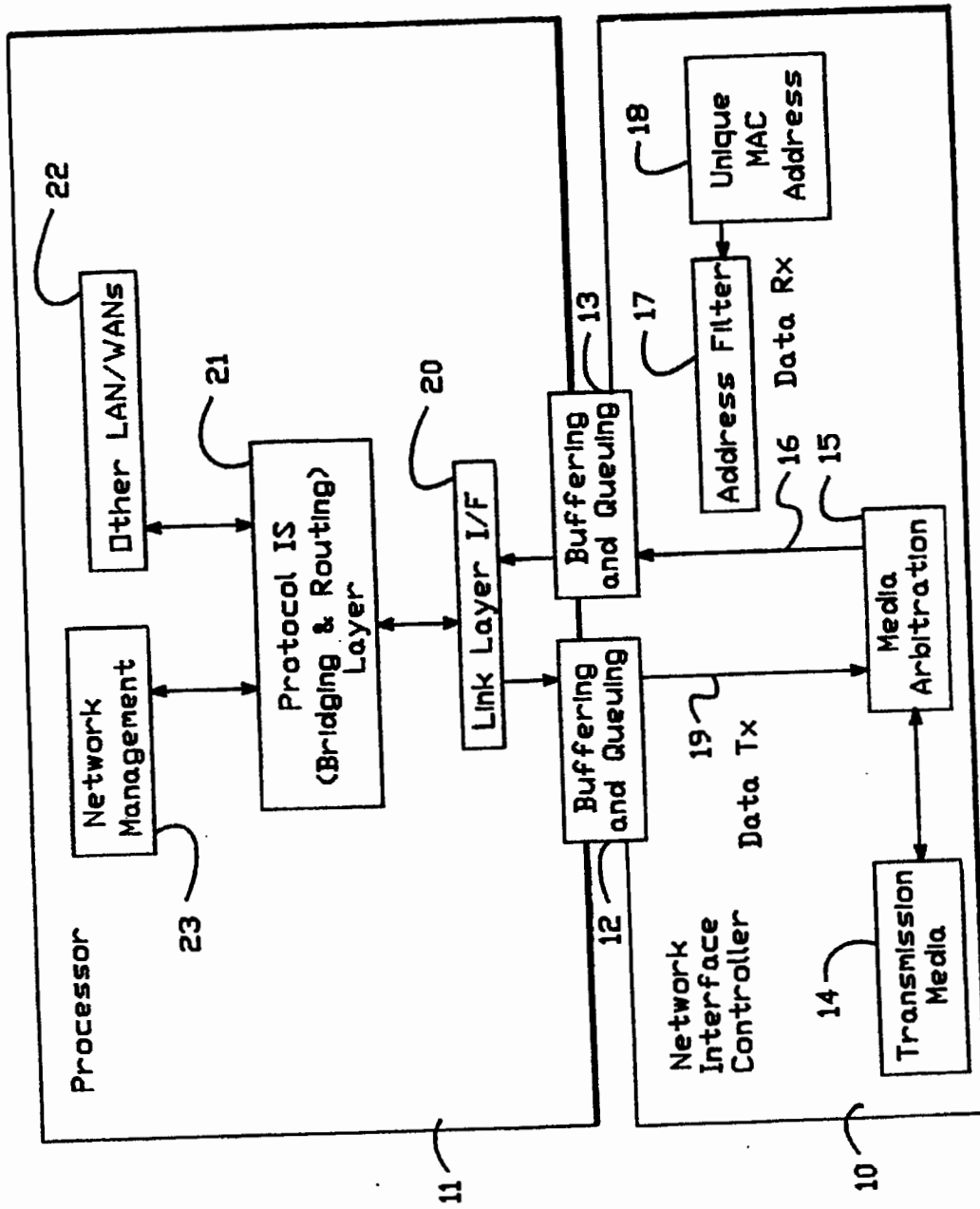


FIG. 1
(PRIOR ART)

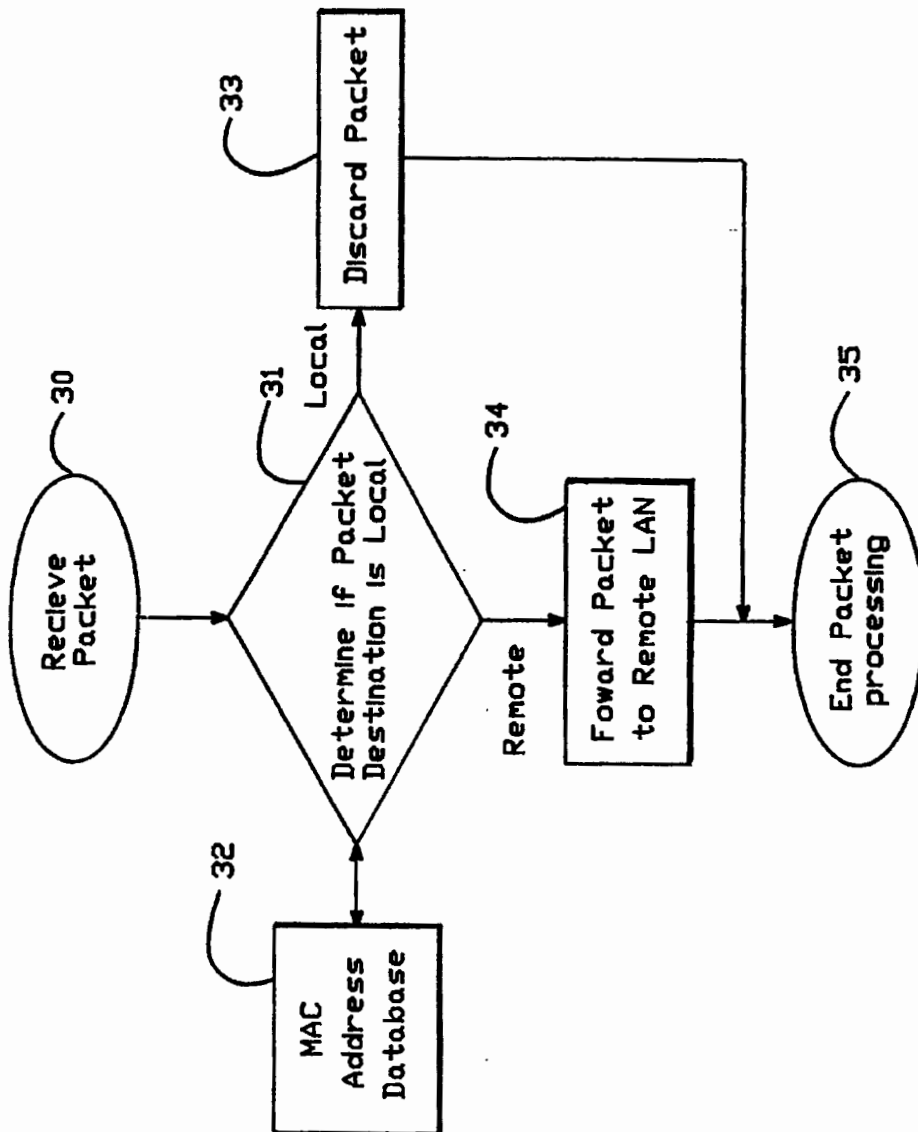


FIG. 2
(PRIOR ART)

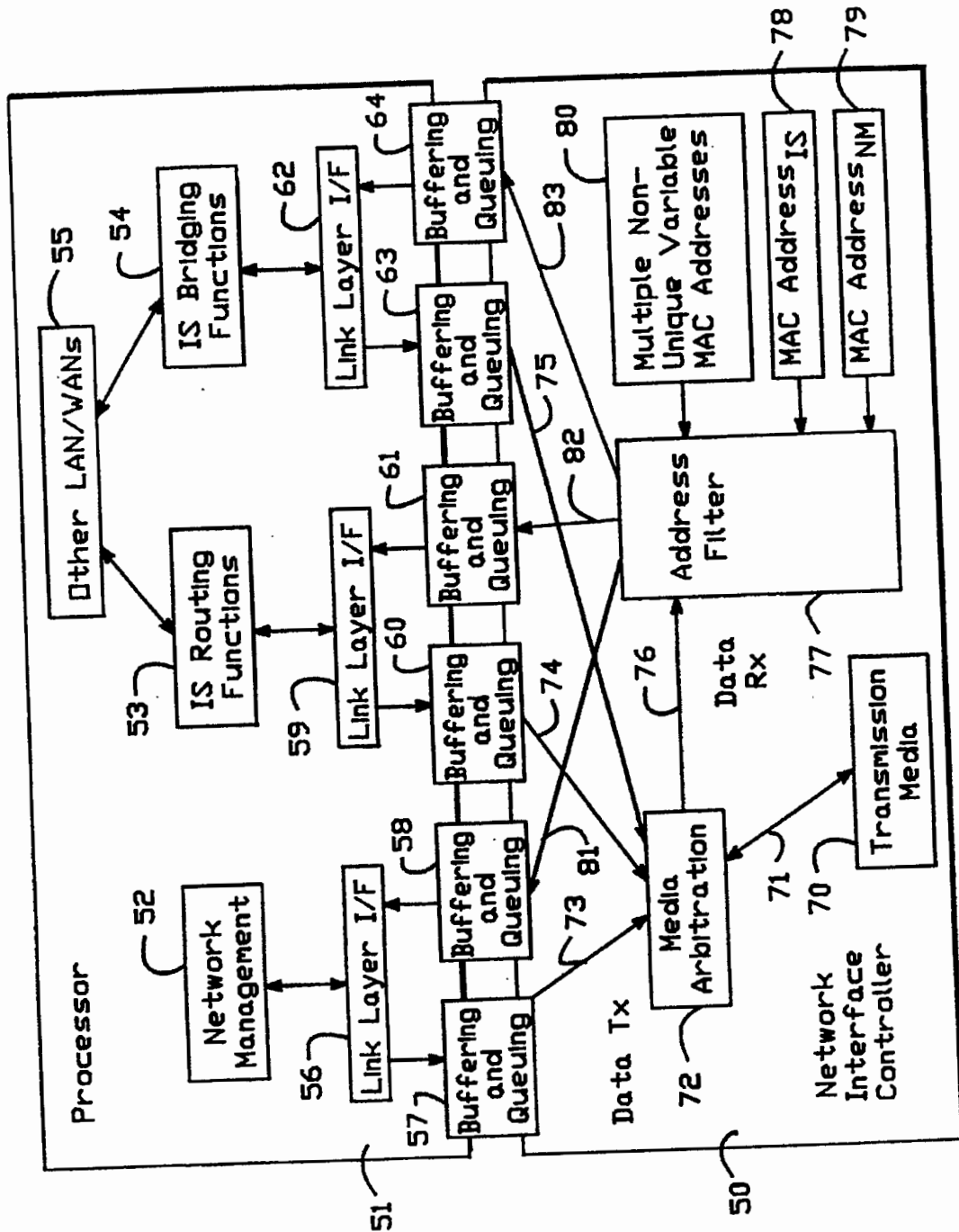


FIG. 3

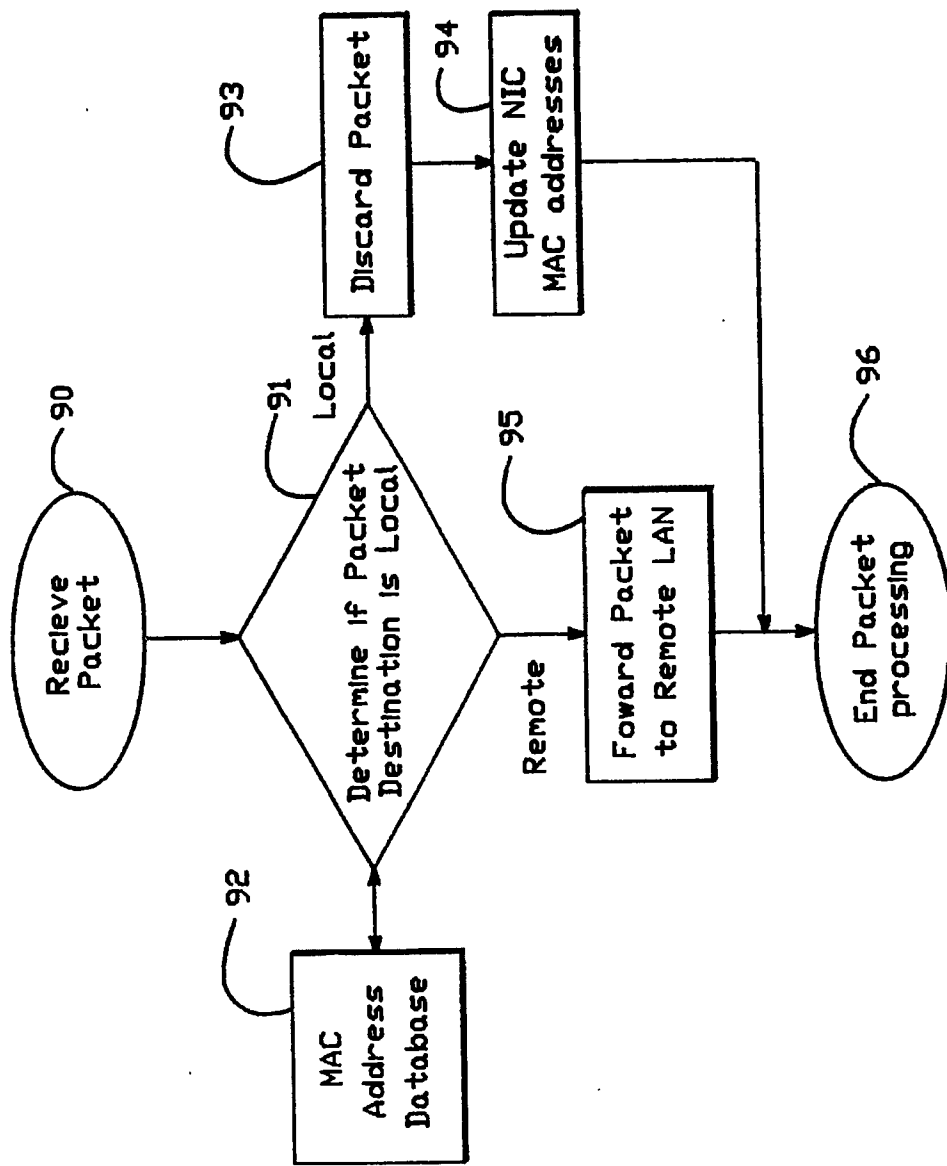


FIG. 4

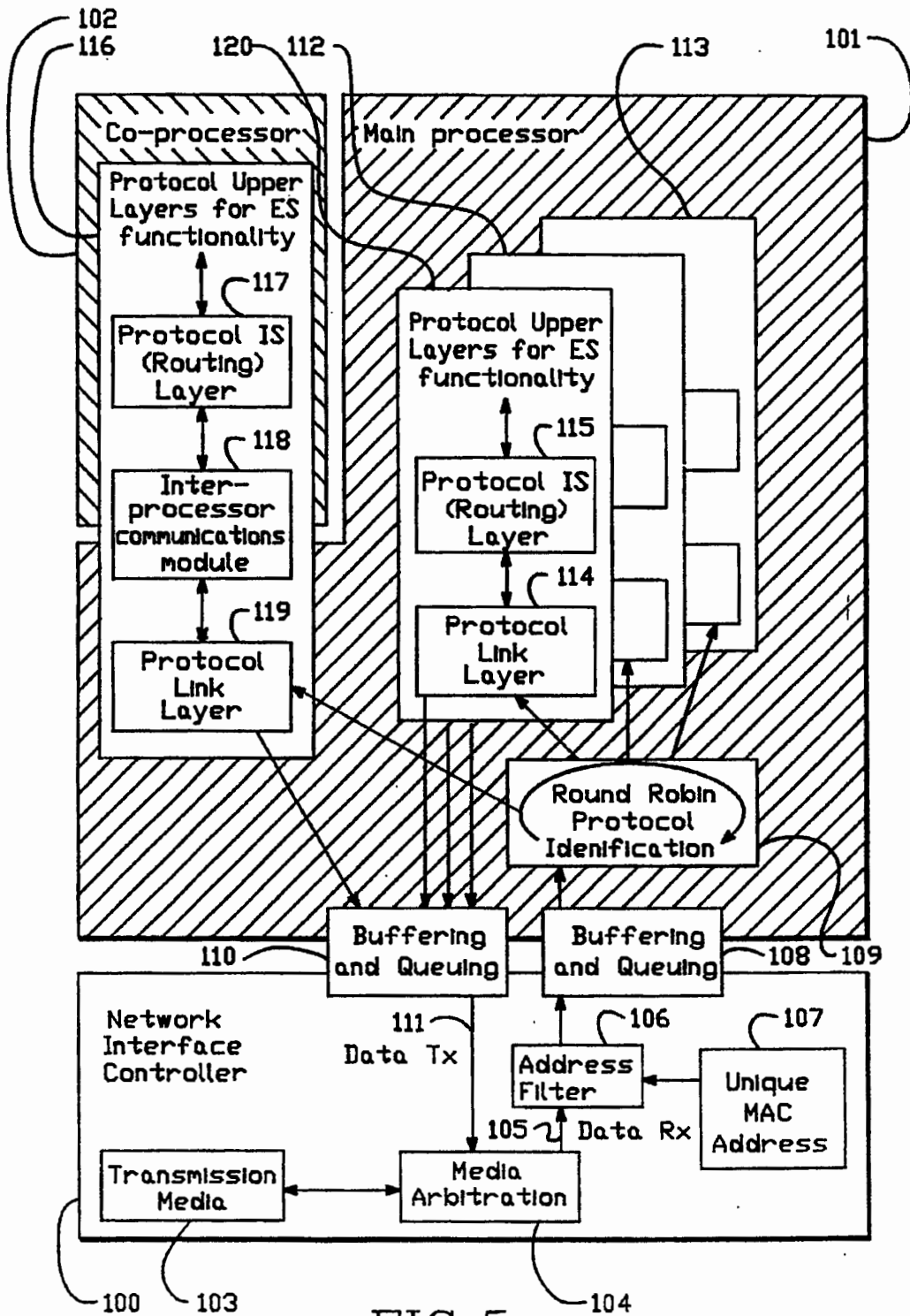


FIG. 5
(PRIOR ART)

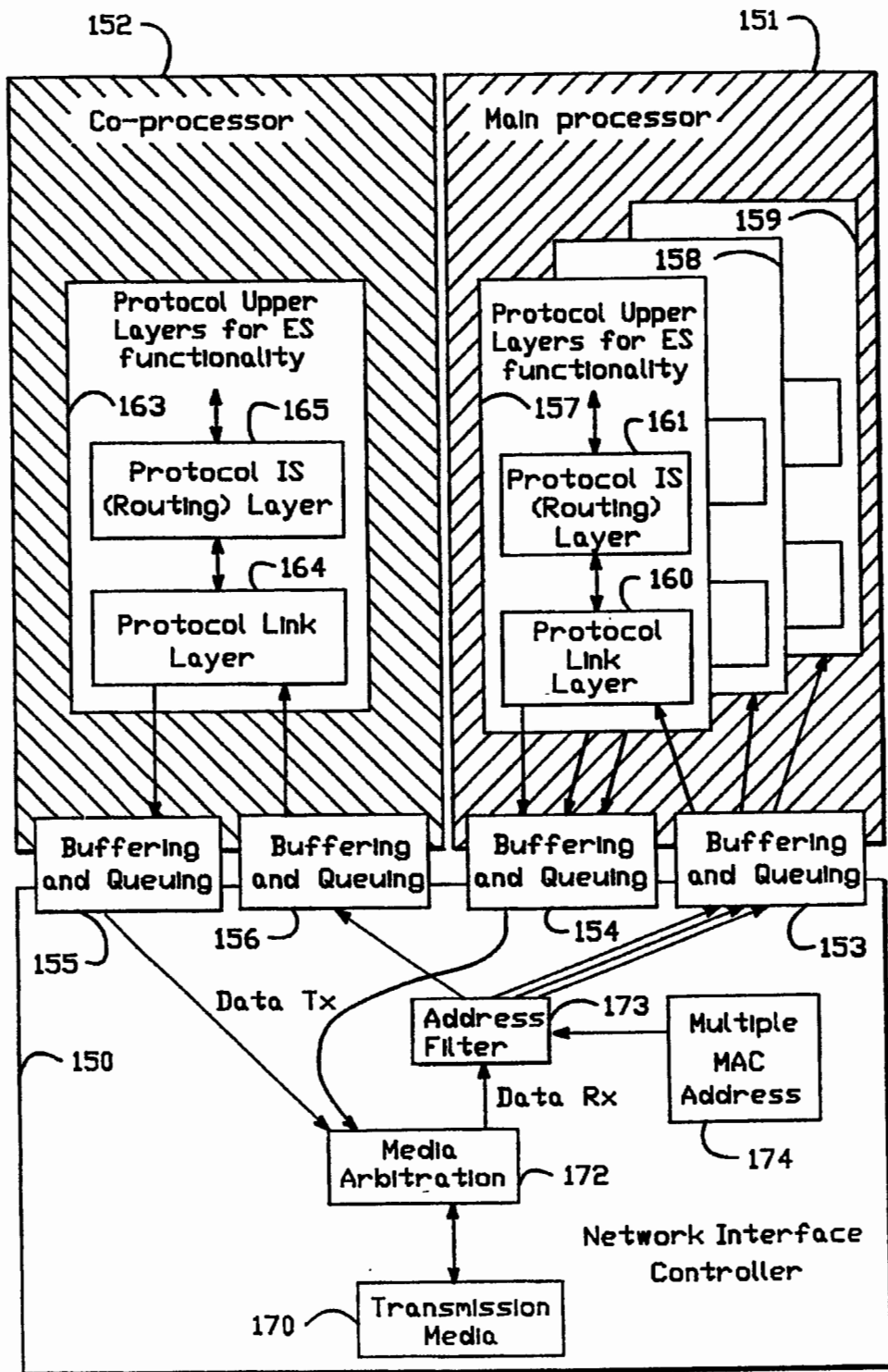


FIG. 6

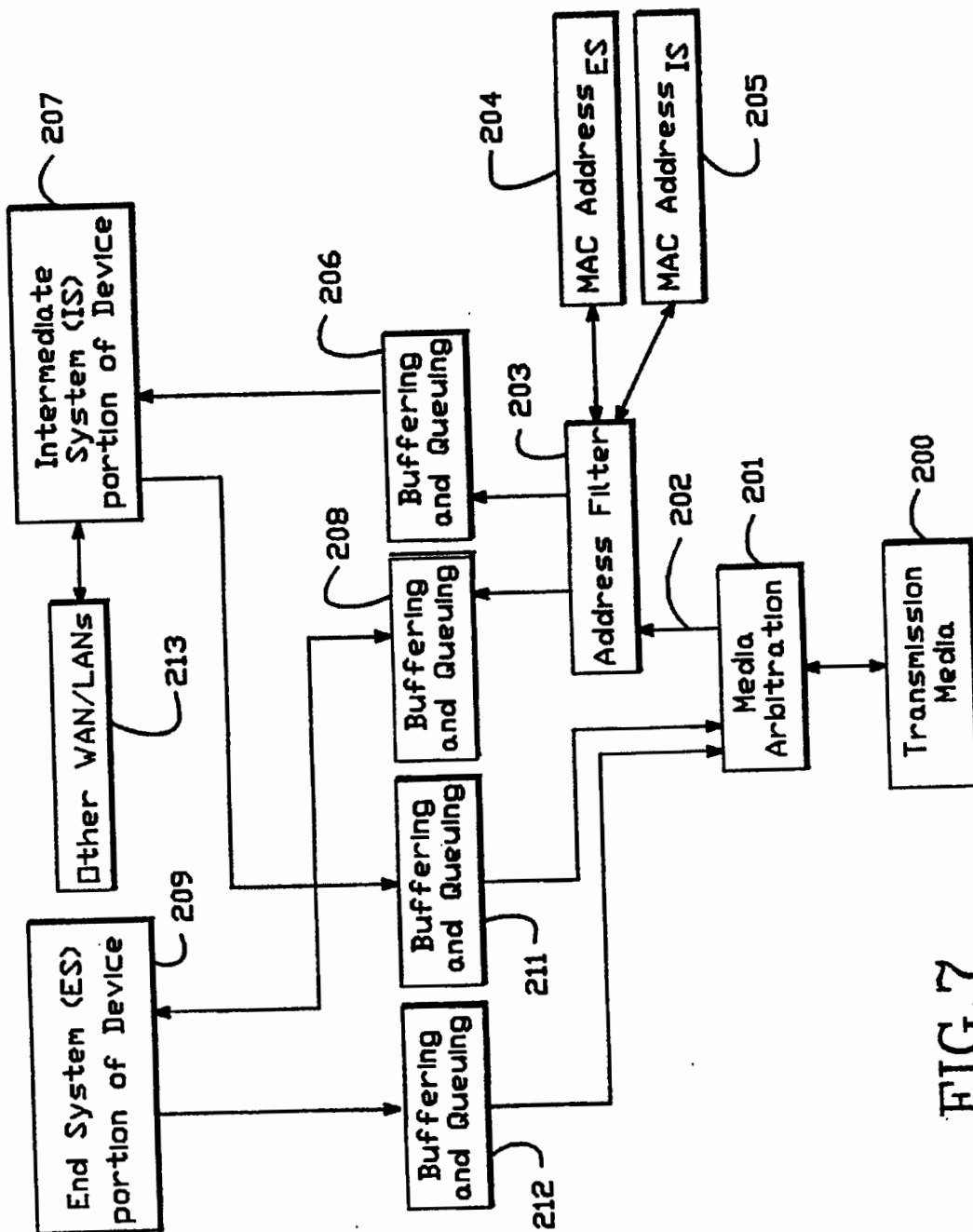


FIG. 7

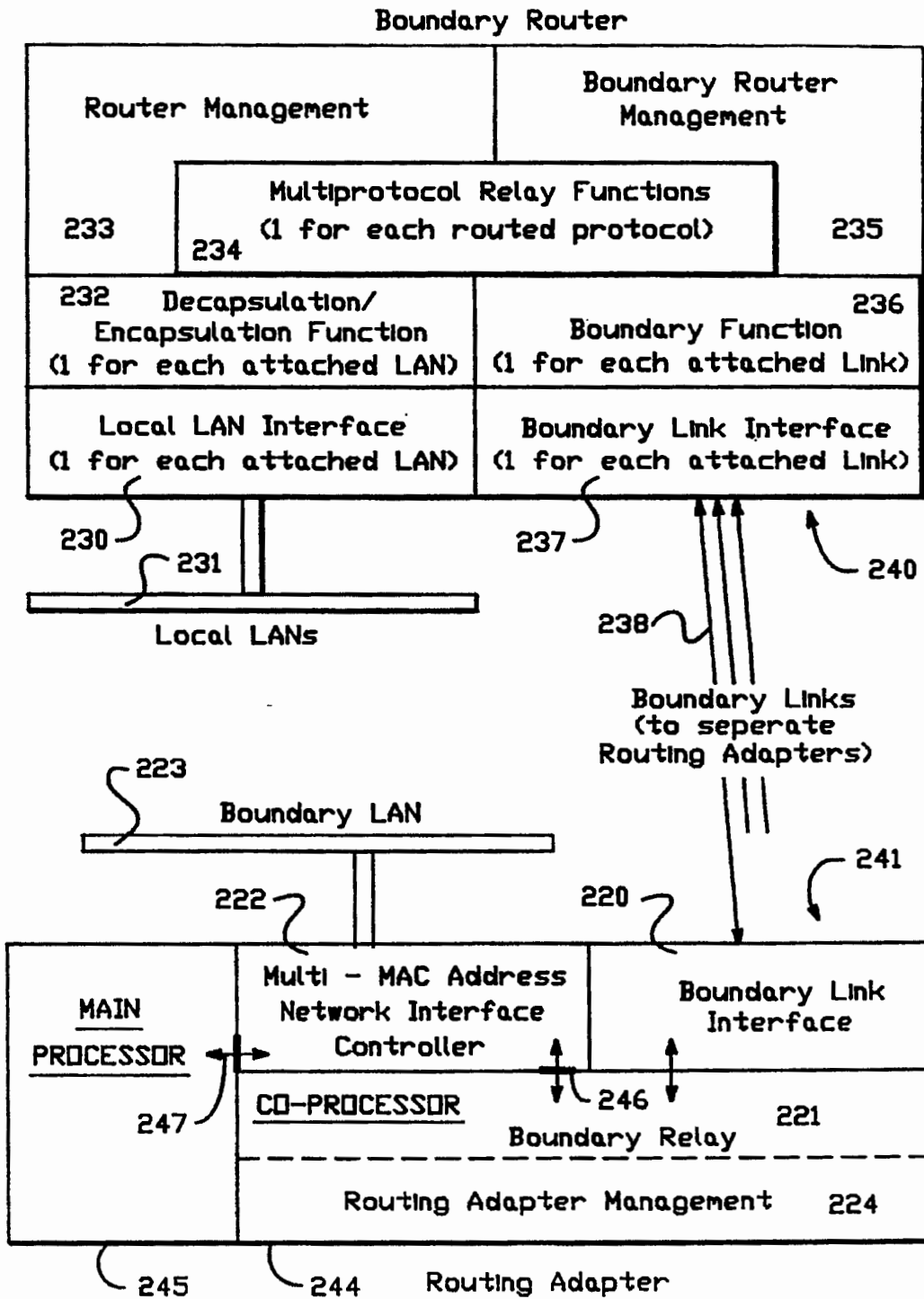


FIG. 8

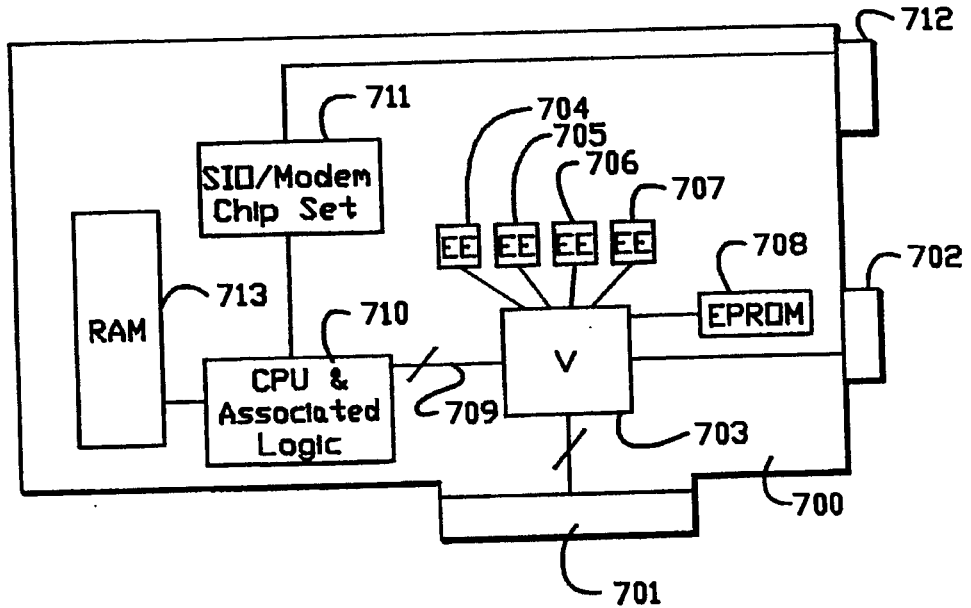


FIG. 9

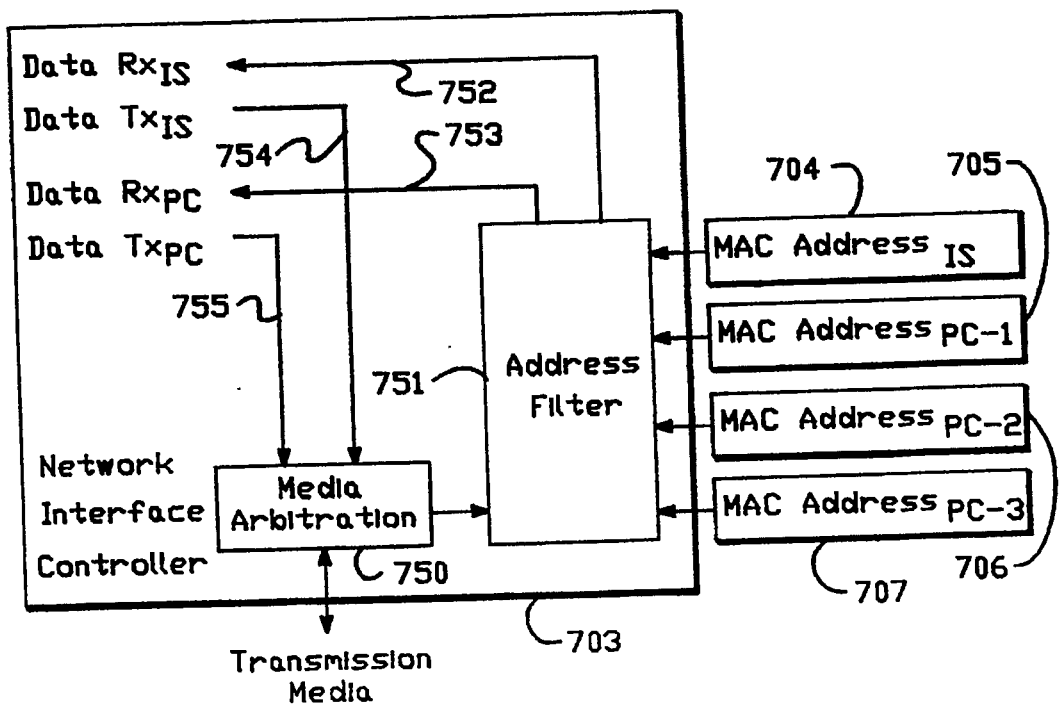
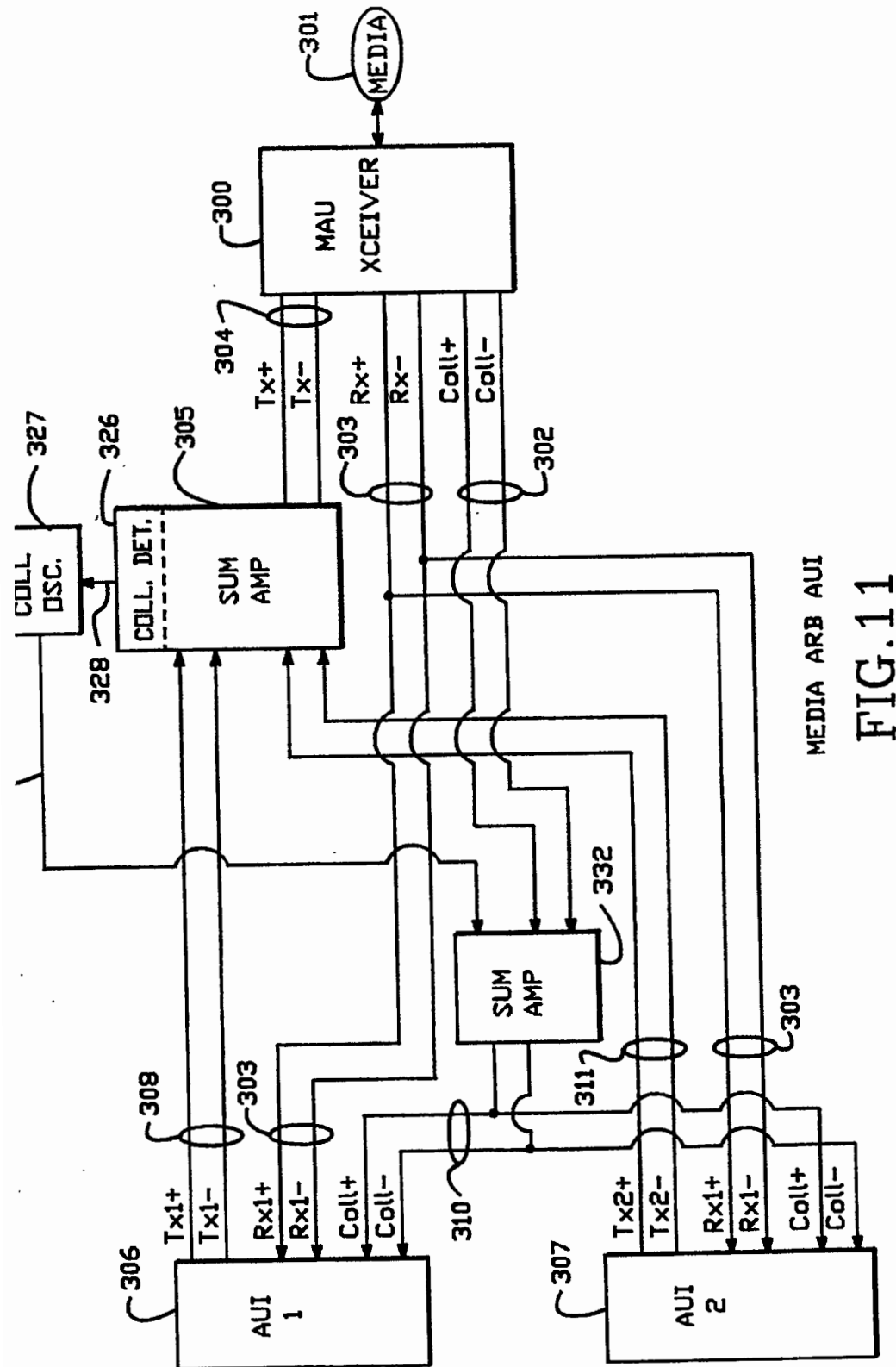
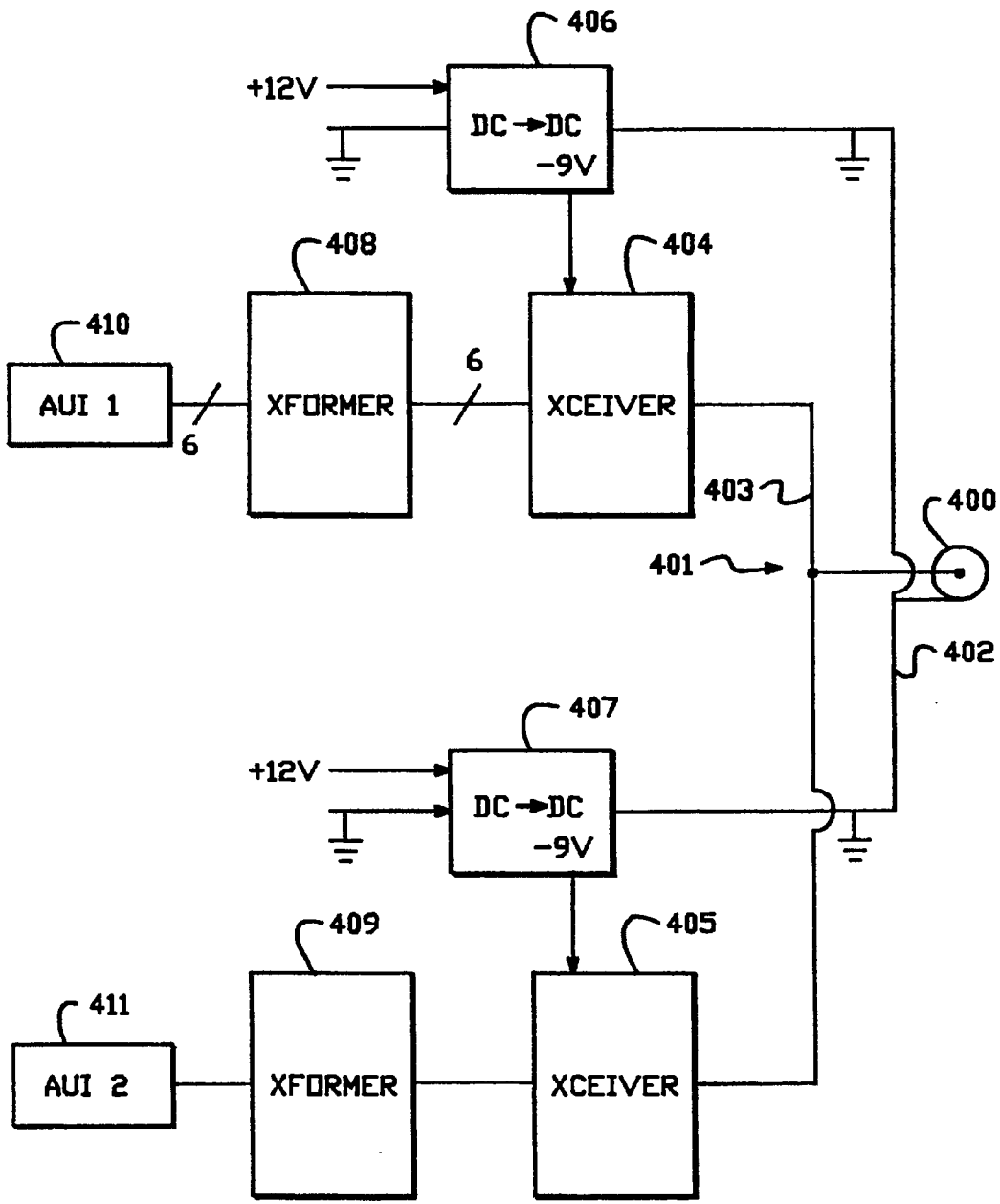


FIG. 10

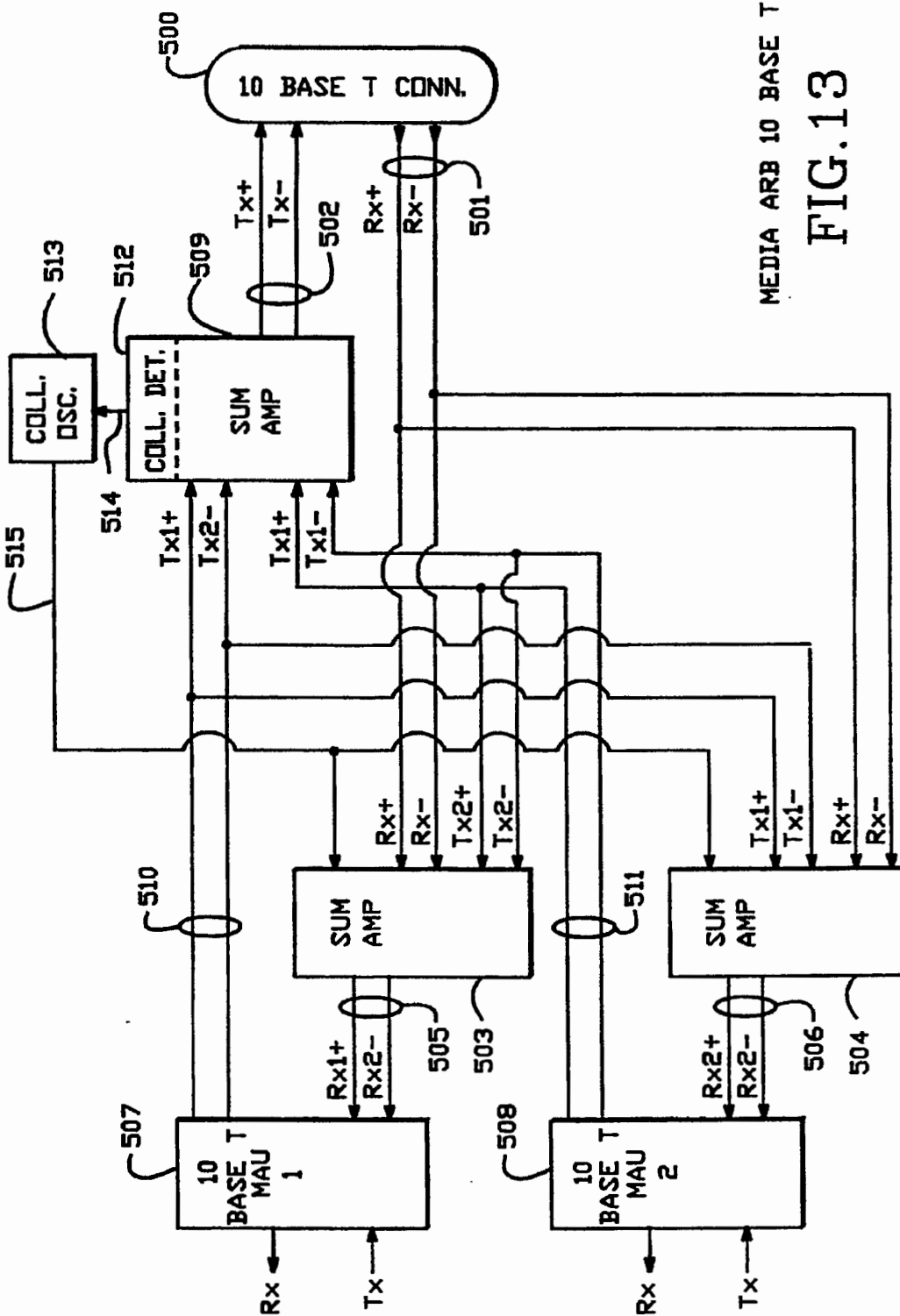


MEDIA ARB AUI
FIG. 11



MEDIA ARB BNC

FIG. 12



MEDIA ARB 10 BASE T

FIG. 13

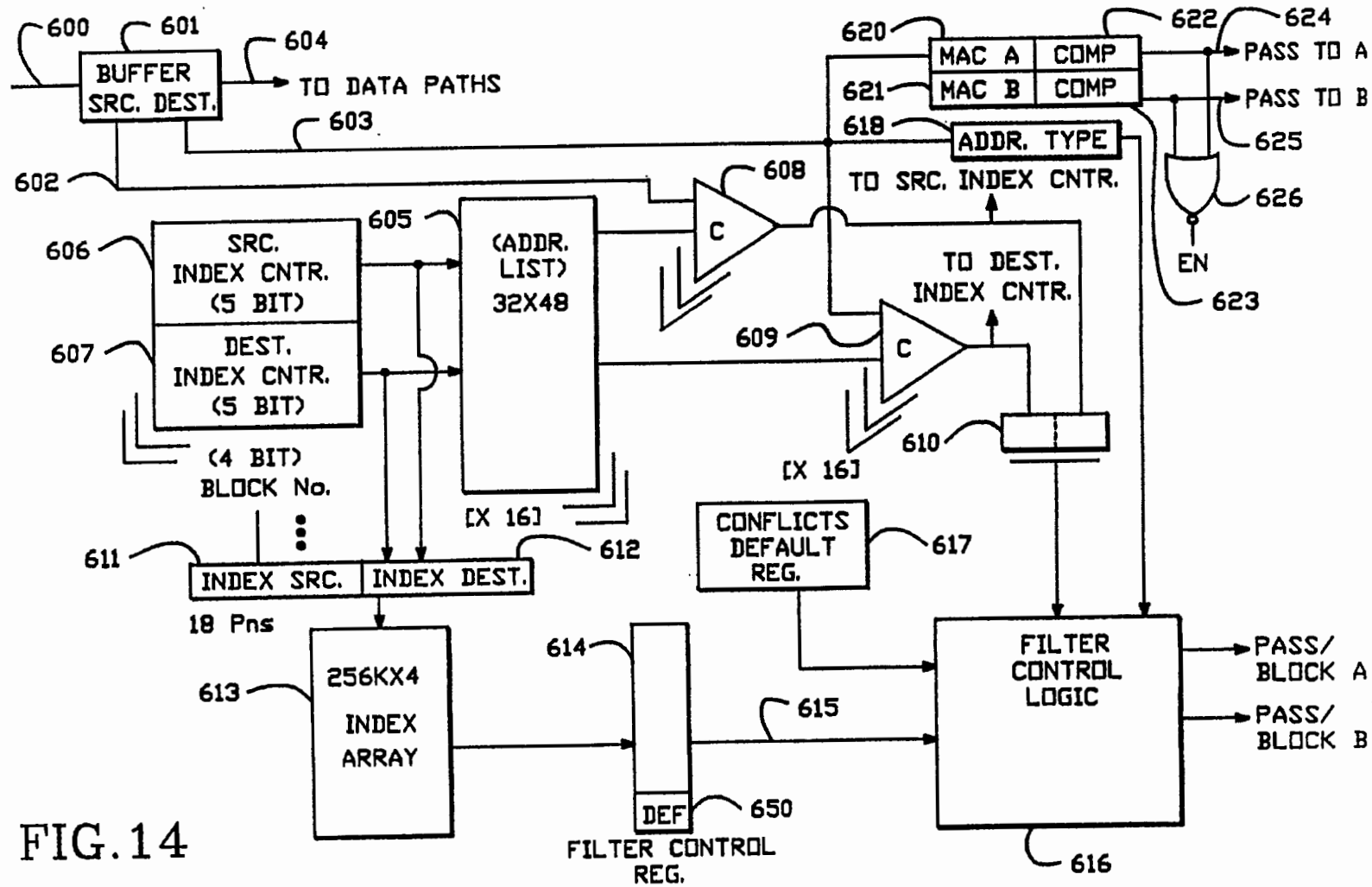


FIG. 14

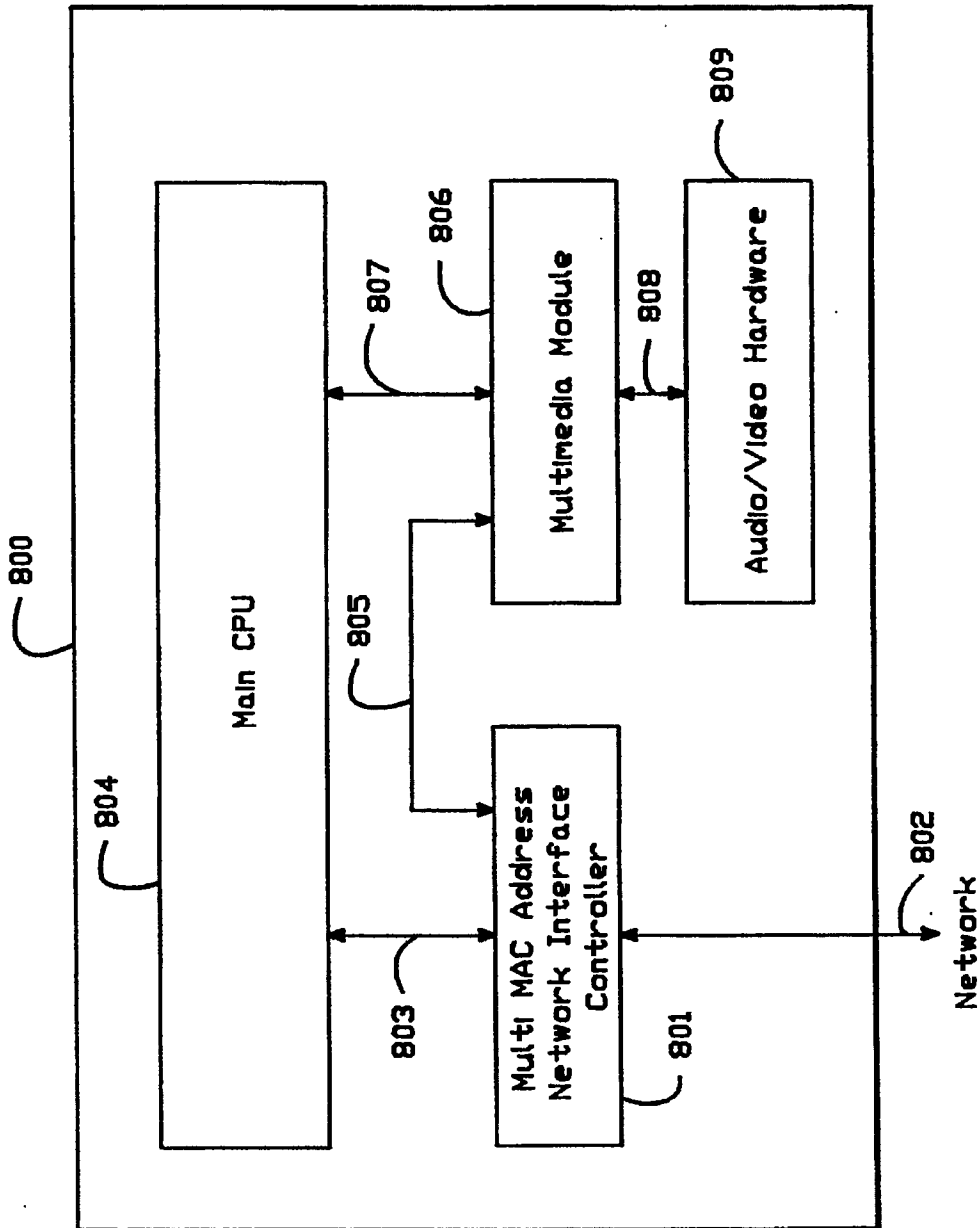


FIG. 15

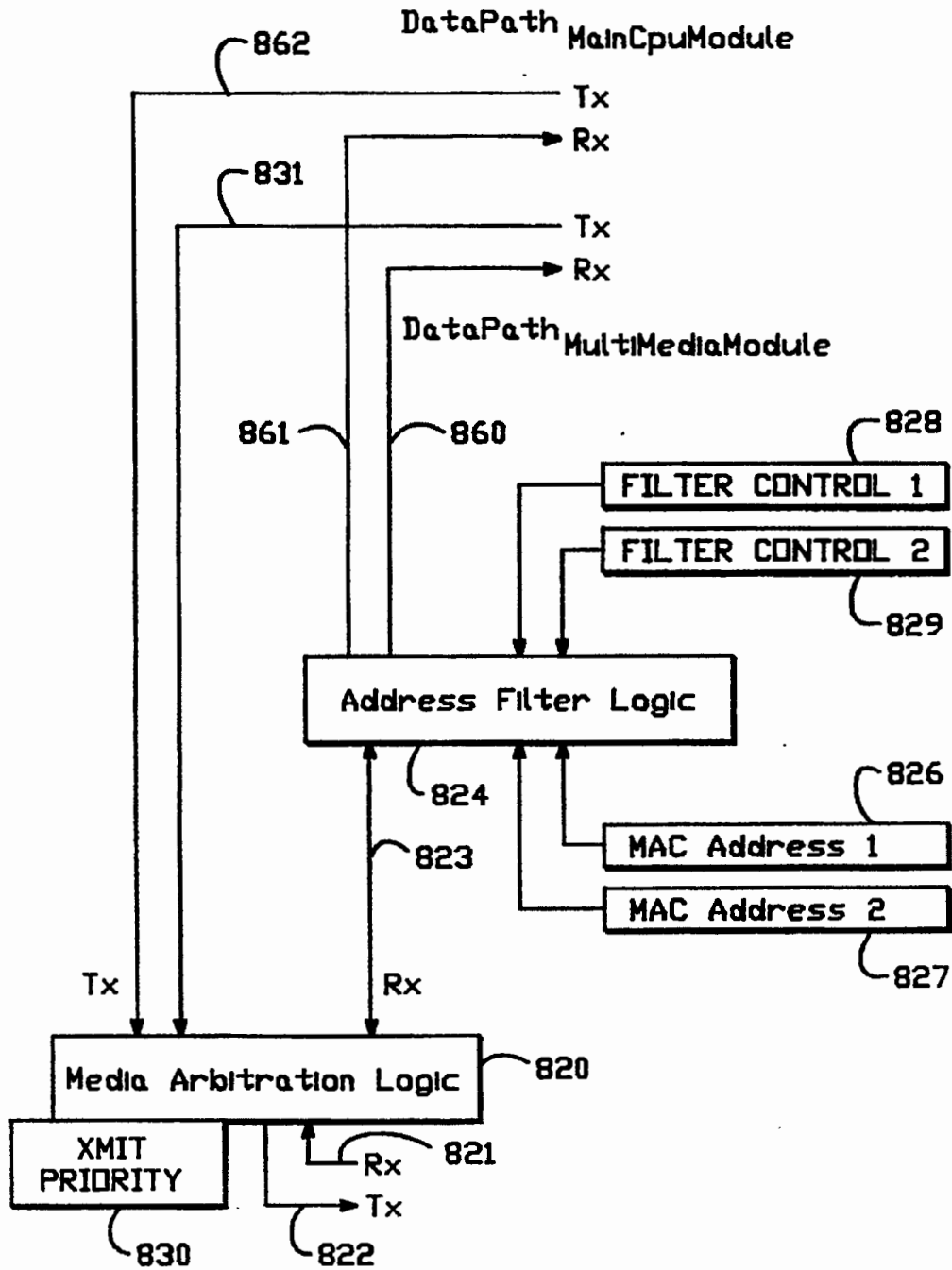


FIG. 16

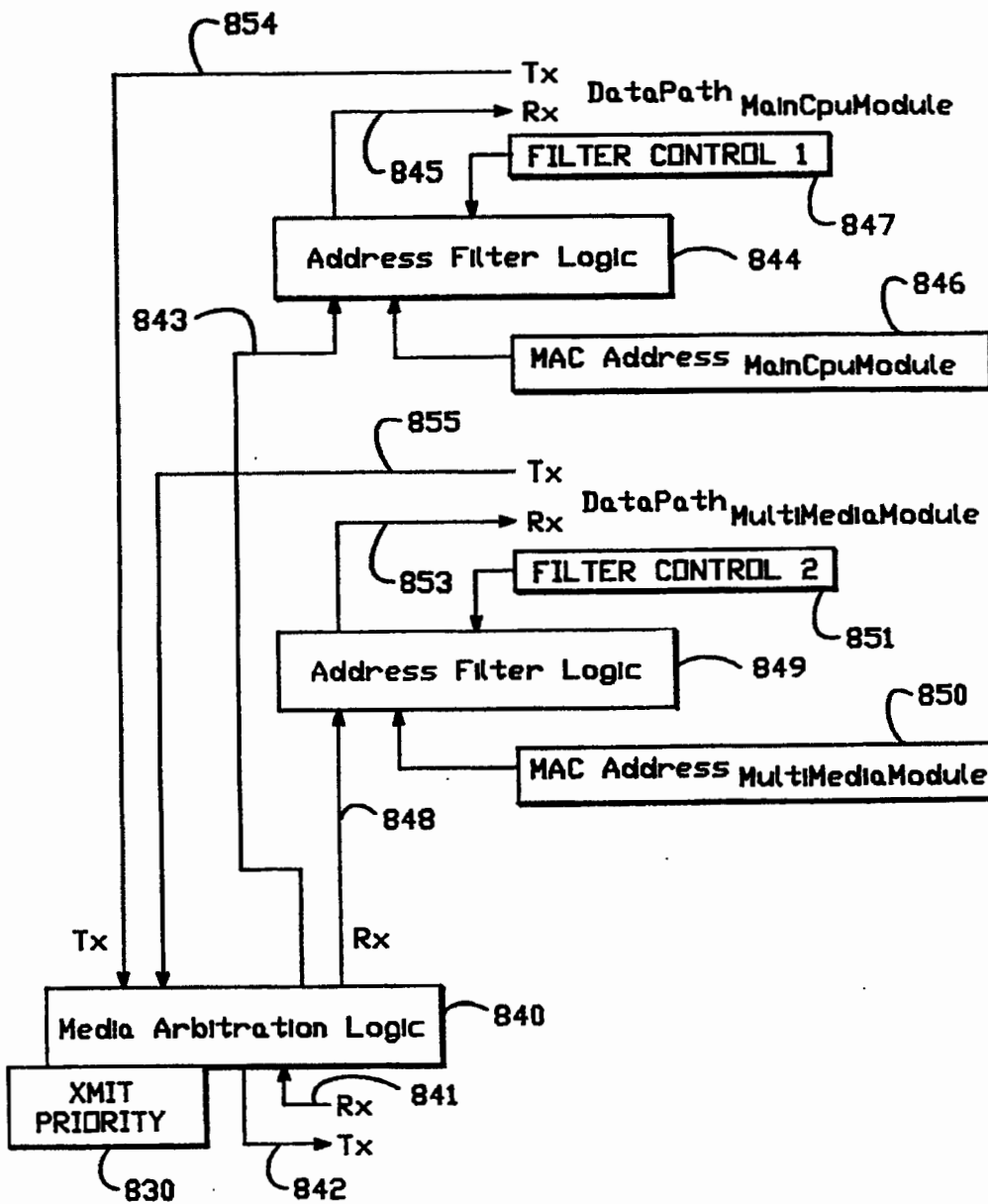


FIG. 17

MULTIFUNCTION NETWORK STATION WITH NETWORK ADDRESSES FOR FUNCTIONAL UNITS

This application is a divisional of application Ser. No. 08/098,616, filed Jul. 28, 1993.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to local area networks, and more particularly to network interfaces for multi-function computer systems.

2. Description of Related Art

Today, Local Area Networks (LANs) are connected together with computers such as Local or Remote Bridges, Routers, Bridge-Router hybrids known as BRouters, or Boundary Routers (see U.S. patent application by John Hart, Ser. No. 07/871,113, Filed Apr. 20, 1992, now abandoned, entitled SYSTEM FOR EXTENDING NETWORK RESOURCES TO REMOTE NETWORKS). All of these computers which connect LANs together are known as Intermediate Systems (IS Computers). (*Extending the IEEE 802.1 MAC Bridging Standard to Remote Bridges*, John Hart, IEEE Network, The Magazine of Computer Communications, January 1988, Vol. 2, No. 1; and *Integrating Bridges and Routers in a Large Internetwork*, Eric Benhamou, IEEE Network, The Magazine of Computer Communications, January 1988, Vol. 2, No. 1).

Attached to the LAN are personal computers (PCs), minicomputers, mainframes, printers, and other devices known as End Systems (ES Computers).

The operation of a network is usually described in terms of the OSI model (*Carrier Sense Multiple Access with Collision Detection*, IEEE Std 802.3-1985; and *Token Ring Access Method and Physical Layer Specifications*, IEEE Std 802.5-1985). This is a description of technology in terms of 7 layers, ranging from the physical layer (OSI layer 1) including the medium or wire strung between machines, to the software applications such as Claris's Filemaker Pro® Database software (OSI layer 7) (Filemaker is a registered trademark of the Claris Corporation of Santa Clara, Calif.). To allow interoperability between vendors, and simplify the design and creation of networking products, most networking products are designed in terms of these layers. For example, devices at the data link layer (OSI layer 2) almost never have the wire built in, but instead have a connector which allows some other vendors wire to be attached.

Layer 1 of the OSI model is the physical layer. This includes the transmission medium, typically the wiring infrastructure, that is run between devices, such as telephone wire, coax cable, or fiber optic cable. At this level the data being shared across a network is viewed in terms of electrical transmission signals, such as square waves, serial bit streams, or parallel bit streams.

The second layer of the OSI model is the data link layer (DLL). This is the layer that describes devices which connect to the actual wire, such as Network Adapter cards. The portion of the devices which place data frames onto the wire or cable, and which arbitrate among devices that share a single wiring scheme is called the Media Access Control (MAC) portion of the DLL. At this level data is viewed in terms of packets or frames, which contain a well defined header containing a source (originator) and destination (target) address. Data being shared is also in the packet but is

not understood by layer 2 devices in terms of what the content or meaning of the data is.

The third and fourth layers of the OSI model are the Network Layer and Transport Layer. Due to the development of many products before the standardization of the OSI model, many products blur these two layers together in terms of implementation. From the point of view of this patent, these two layers are where a decision on which DLL device to send the data to occurs if there is more than one connection in the device, in which the appropriate packet header is determined based on the DLL choices, and in which passing or forwarding data to another computer is managed. Because of the lack of standardization in the early days of networking, today there are a number of different products at these layers, usually referred to as Network Protocols or Protocol Suites. Examples are the TCP/IP protocol suite (*DDN Protocol Handbook*, Volume 2, DARPA Internet Standard, December 1985) and the XNS protocol suite (*Internet Transport Protocols*, Xerox System Integration Standard, XSI 028112, December 1981; and *MS-DOS Internal Network Driver Scheme (MINDS)*, Version 1, 3Com Corporation, January, 1984).

The fifth, sixth, and seventh layers of the OSI model are not relevant to this invention except as they benefit from the increased performance of computers which use the invention.

The DLL technology which connects a computer to the LAN is usually referred to as a Network Interface Controller (NIC). Examples of NIC devices are the Intel 586 Ethernet Controller (Intel Corporation, Santa Clara, Calif.), the IBM & National TROPIC TokenRing Controller (National Semiconductor, Santa Clara, Calif.), and the 3Com Vulcan Ethernet Controller (3Com Corporation, Santa Clara, Calif.). Each NIC is configured with a unique, single address MAC address which was assigned by the manufacturer of the NIC during the manufacturing process from a pool of network addresses assigned to the manufacturer by a standards body.

The MAC addresses used in packets at the DLL are identifiable as being either single destination addresses or group addresses. In the most common data link technologies such as Token Ring and Ethernet, this is done by setting the first address bit placed on the physical media to 1 to denote group addresses, or to 0 to denote single destination addresses. Packets in which the destination address is a single destination address are called Unicasts and are used when only a single IS/ES computer which has been assigned that MAC address is intended to receive that packet. Packets in which the destination address is a group address are called Multicasts and are used when a packet is intended to be received by all or a group of IS/ES computers.

Most NICs are designed so that the upper layers on the computer get all multicasts plus any unicasts with that ES's unique address as the destination address of the unicast. This allows the upper layers of software on an ES to only process single address packets which are targeted for that ES, and to be able to filter (block or pass) at the DLL data that is intended for another computer.

When one ES computer wishes to exchange data with another ES computer, they must first discover each others' MAC addresses. This process is called Name resolution. Typically, the first computer sends a special multicast containing its unique MAC address and the upper layer name of the second ES. This is received by all ES computers, but only the ES computer which has the correct upper layer name responds directly to the originating ES. It does so with a unicast to the first computer containing its MAC address

so that the two computers can exchange unicasts from that point on. A variant of this exchange is the concentration of the name resolution process into a name or locator server, which acts on behalf of the computers on the LAN, and responds on behalf of the end system. After identification has been established, unicasts containing each others unique MAC address are used, allowing the NIC at each device to accept directed packets for itself and ignore (block at the data link layer) directed packets intended for other computers.

ES computers with multiple protocol suites such as TCP/IP and XNS operating concurrently are constructed most often with a shared NIC, and upper layer software such as the Protocol Manager defined in the Network Device Driver Specification (NDIS) (*Network Device Driver Interface Specification, Version 2.0*, 3Com Corporation and Microsoft Corporation, 1989) requesting each protocol to look at the packet and determine if it is for that module or not, repeating the process until a protocol module identifies the packet. This methodology is referred to as Round-Robin Packet Identification (RRPI).

IS devices which connect multiple LAN's together transparently to ES computers at OSI layers 2 and above are called Bridges. Bridges can not take advantage of the ability to differentiate packets at the MAC layer, and operate in a fashion called promiscuous mode in which all packets regardless of MAC address are accepted. This is done in order to be able to learn which ES and IS devices are connected to the LAN so as to properly transfer packets to and from other LANs.

IS devices which connect multiple LAN's together by cooperating with ES computers at layers 2 to 4 are called Routers. These devices do not need to use promiscuous mode as they only forward packets to other LANs that are addressed to them by the originating ES computers. However, because of the large number of protocols and environments in which routing does not work, most routers today also have to support bridging and thus must use promiscuous mode. Computers which perform both Bridging and Routing functions are known as BRouters. Most full function router products on the market today are in fact Brouters.

Computers which have both IS and ES functionality are for the most part protocol specific and support only routed protocols because of the processing overhead of "promiscuous mode" that bridging requires. In these computers, protocol identification occurs in software as described above with NDIS, and then at routing layer 3 of each protocol further identification occurs to determine if the packet is for the ES portion of the device or the IS portion. There are several products available today which provide this integration of IS and ES functionality into single computers, such as *PhoneNet Liaison* for Macintosh computers, available through Farallon Computing, Inc., (Alameda, Calif.), and *3+Open Internet for OS/2 LAN Manager* for PC computers, available through 3Com Corporation.

All of these integrated IS/ES computers today work by sharing the layer 1 and 2 components, and then having the layer 3 and 4 software on the ES determine if the packet is destined for the ES module and thus passed to upper layer software for processing, or if the data is destined for the IS module and handled at that level. Group address packets often have to be handled by both modules.

Benefits of integrated IS/ES computers include cost savings of the equipment, and a reduction of the number of computers in the network with its resulting increase of network reliability and manageability. However, there are

also problems with this approach. First, because routing decisions are protocol specific and protocols are very large in terms of code space and complexity, most implementations support only one or perhaps two protocols. Second, software changes are problematic as they require changes to an integrated IS/ES protocol suite. Therefore much more testing is required and the updates are larger in scope. Third, most products can not perform bridging functionality because the processing cost of looking at every packet on the LAN is too great for the ES CPU to do and still be able to handle ES functions such as database management or file system management. In addition, there are occasionally software errors and defects in application software executing on ES devices unrelated to the protocol layers which can adversely affect the performance of the IS functions, and thus indirectly affect many other ES devices in the network which depend on the IS functionality.

As described above, computers have a single unique MAC address assigned to them which is used at the data link layer to identify the computer and to block out packets which are not intended for the computer. This works acceptably for simple ES computers and for IS computers which route (and do not bridge). It does not work as well for integrated IS/ES devices, or for IS devices in which bridging occurs. For both of these applications devices today must do significant processing at high layers of software. In addition, in a network with a large number of computers the amount of processing an ES computer has to do with multicasts, most of which are just ignored, can be significant.

SUMMARY OF THE INVENTION

The present invention builds DLL devices with multiple MAC addresses instead of a single MAC address, and provides multiple virtual DLL interfaces to the upper layers (3-7) in a computer. This results in a new class of multi-function computers for attachment to a network system which takes advantage of the multiple virtual DLL interfaces, to increase performance of the respective functions executed by the computer. Thus, a new network interface control apparatus and a new class of multi-function computer systems for attachments to networks are provided.

The network interface control apparatus according to the present invention includes a connector for transporting data to and from the network transmission medium, and a medium access control device coupled to the connector. The medium access control device receives and transmits frames of data through the connector, and includes a plurality of data channels to communicate with respective processing modules in a connected computer. Memory in the medium access control device stores a plurality of assigned network addresses for the plurality of data channels. Address filtering logic in the device is coupled to the plurality of data channels and to the memory, and passes and blocks frames received through the connector for the plurality of data channels in response to the respective assigned network addresses.

According to one enhancement, the memory in the medium access control device stores a plurality of additional network addresses in addition to the assigned network addresses. The address filtering logic includes circuits responsive to the additional network addresses, such as logic for blocking a particular frame on at least one of the plurality of data channels when the source and destination address of a particular frame are found in the additional addresses stored in the memory.

5

The plurality of data channels served by the media access control device may reside on a single physical interface, or in independent physical interfaces as suits the needs of a particular design. A high performance design would include independent buffering and queuing structures for each of the data channels. An alternative design may include shared buffering and queuing structures for a plurality of functional modules in the connected computer which have independent, assigned network addresses.

The network interface control apparatus for certain types of network transmission media will include a media arbitration circuit coupled between the connector and the medium access control device. This circuit supplies to the address filtering logic frames which are received through the connector from the network transmission medium, and frames which are to be transmitted to the network transmission medium from the plurality of data channels. Thus, the network interface control apparatus receives frames which are transmitted by it, so that the independent functional modules having separate network addresses, may process communications from other functions sharing the network interface control apparatus.

Alternatively, the present invention can be characterized as a network interface control apparatus which comprises a physical layer device to transport data frames to and from the network transmission medium, and a plurality of virtual data link layer modules coupled to the physical layer device. The virtual data link layer modules include a corresponding plurality of data channels to respective higher protocol layer modules in the connected computer, memory to store assigned network addresses for the plurality of virtual data link layer modules, and address filtering logic coupled to the physical layer device, the plurality of data channels and to the memory, which passes and blocks data frames received from the physical layer device for the plurality of virtual data link layer modules in response to the assigned network addresses. In one aspect of the invention, the physical layer device includes a circuit which merges the plurality of data channels of the corresponding plurality of virtual data link layer modules, into a single data path for connection to the network transmission medium.

Thus, to higher protocol layer modules in the computer, independent data link layer interfaces are presented, which share address filtering logic and physical layer circuitry, and which receive transmissions from the other higher layer modules in the computer.

As mentioned above, the present invention can also be characterized as a new class of multi-function computer system. The computer system according to this aspect of the invention includes a network interface coupled to the network transmission medium, which includes at least a first processor interface having a first assigned network address, and a second processor interface having a second assigned network address. First processing resources are coupled to the first processor interface which receive and transmit frames through the network interface. Further, second processing resources are coupled to the second processor interface which receive and transmit frames through the network interface.

The first and second processing resources may comprise a main CPU executing end system procedures, and a co-processor CPU which executes intermediate system functions. Alternatively, the first and second processing resources may comprise separate protocol suites executed by a single CPU. The first and second processor interfaces may include respective buffering and queuing structures for com-

6

munication of data between the network transmission medium and the first and second processing resources, or may share a single buffering and queuing structure.

As described above, one aspect of the invention comprises presenting the first and second processor interfaces as respective virtual data link layer interfaces to the higher layer functions executed by the computer system. Further, the system may be configured such that one of the first and second processor interfaces comprises a plurality of virtual data link layer interfaces having respective assigned network addresses.

In another aspect, the present invention presents a new class of integrated end system/intermediate system computers. This new class of IS/ES computers is based on the network interface having a plurality of assigned network addresses. Thus, the device may include a first network interface coupled to a first network transmission medium, which includes at least the first processor interface having a first assigned network address and a second processor interface having a second assigned network address. A first processor is coupled to the first processor interface, and includes network end system resources. A second processor is coupled to the second processor interface, and includes at least a second network interface coupled to a second network transmission medium. The second processor executes network intermediate system resources for transporting frames between the first and second network interfaces for transmission across the first and second network transmission media. Thus, the second processor may provide "routing adaptor" functions as described herein. Also, this system may be expanded to present any number of ports to network transmission media on the second processor, such that it executes bridging, routing, and Brouting functions. Further, these intermediate system functions are executed without interfering with the network end system resources executed on the first processor.

In yet another aspect, a new class of multi-media system has been provided which includes a multiple MAC address network interface controller providing a first virtual data link layer interface to a main CPU and a second virtual data link layer interface to a multi-media module which handles high volumes of audio and video data. The audio and video data can be transmitted through the network to or from the multi-media module based on the unique MAC address of the module, and without the overhead associated with higher level protocols used to communicate with the main processing module or other end systems in the network.

In sum, there are four basic network interface aspects to the invention. The first is to use multiple MAC addresses to a data link layer device and do address blocking or passing on this set rather than on just one address. The second aspect of the invention is to have virtual DLL's by the creation of single data link layer devices which appear as multiple separate data link layer devices to the upper layers of the computer. The third aspect of the invention is the media arbitration methodology required when a single data link layer device is emulating more than one device. The fourth aspect to the invention is the methodology to receive a packet while it is being transmitted.

Furthermore, a new class of multi-function computer system, a high performance integrated IS/ES product and a new multi-media system have been provided based on the virtual data link layer technology.

Other aspects and advantages of the present invention can be seen upon review of the figures, the detailed description, and the claims which follow.

BRIEF DESCRIPTION OF THE FIGURES

FIG. 1 is a functional block diagram of a prior art network intermediate system.

FIG. 2 is a flow chart of a software algorithm for the bridging section of the prior art intermediate system of FIG. 1.

FIG. 3 is a functional block diagram of a network intermediate system implemented according to the present invention with multiple assigned MAC addresses.

FIG. 4 is a flow chart for the software algorithm of the bridging function of the intermediate system of FIG. 3 adapted according to the present invention.

FIG. 5 is a functional block diagram of a prior art end system/intermediate system computer having a single MAC address.

FIG. 6 is a functional block diagram of an end system/intermediate system having multiple MAC addresses according to the present invention.

FIG. 7 is a functional block diagram of a simplified IS/ES computer according to the present invention.

FIG. 8 is a functional block diagram of a routing adaptor/boundary router system utilizing the present invention.

FIG. 9 is a schematic diagram of a co-processor with a network interface controller according to the present invention, adapted for connection with a host processor.

FIG. 10 is a schematic diagram of the network interface controller for use with the system of FIG. 8.

FIG. 11 is a functional block diagram of media arbitration circuitry for an AUI interface according to the present invention.

FIG. 12 is a functional block diagram of media arbitration circuitry for a BNC connector.

FIG. 13 is a functional block diagram of media arbitration circuitry for a 10BaseT connector.

FIG. 14 is a functional block diagram of an address filter implemented according to the present invention.

FIG. 15 is an overview block diagram of a multi-media system using the multiple MAC address interface controller according to the present invention.

FIG. 16 is a schematic diagram of the network interface controller used in the multi-media system of FIG. 15.

FIG. 17 is a schematic block diagram of an alternative implementation of the network interface controller in the multi-media system of FIG. 15.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

A detailed description of the present invention is provided with respect to FIGS. 1-17, in which FIGS. 1, 2, and 5 represent prior art, and FIGS. 2, 4, and 6-17 illustrate the present invention.

I. Multiple MAC Addresses

According to the present invention, a set of MAC addresses is assigned to a Data Link Layer device instead of only a single MAC address, and logic is provided to block (discard) a packet or pass a packet to the upper layers on the computer based on whether or not the source and destination MAC addresses in the packet match the set of assigned addresses. A DLL device built with this invention incorporated is improved over traditional DLL devices by being able to make decisions at layer 2 where the time spent processing

is minimal, versus at upper layers where the time spent is orders of magnitude more. Some examples of improvements that can be made using the invention include:

- 1) By assigning the computer's DLL multiple unique MAC addresses as if it consisted of multiple different computers, and building the DLL with multiple data channels to the upper layers, a single DLL can provide a virtual interface allowing various upper layer software to be written as if each software module had its own private DLL. This is discussed in more detail in the Virtual Data Link Devices section below.
- 2) By storing in the computer's DLL multiple additional MAC addresses which are multicast addresses, an ES computer can significantly reduce the processing in the upper layer software which must examine each multicast to determine whether it is relevant for it. Since most multicasts are discarded by ES computers, this use of the invention results in significant savings, especially on LANs with a large number of computers. Depending on the protocol, the DLL device can either accept all packets except those in its list of multicast MAC addresses, or accept only those in its list.
- 3) By storing in the computer's DLL multiple additional MAC addresses which were assigned to other computers on the LAN, an IS device which does bridging can have its processing overhead significantly reduced by having the DLL discard packets for which both the source and destination addresses in a packet are in the list of additional MAC addresses of the DLL. If the destination address is in the table, the packet does not have to be forwarded to another LAN. If the source address is in the table, the computer which sent the packet is "known" already to the bridge. If the source address is not in the table then the computer sending the packet is "unknown" to the bridge and its source address will be added to the list of addresses in the table by the upper layer software.

FIG. 1 shows a typical prior art design of an IS computer Router or Bridge which operates in promiscuous mode. The computer includes a network interface controller 10 which is coupled to a processor 11. The interface between the network interface controller 10 and the processor 11 includes a data path including transmit buffering and queuing structures 12, and receive buffering and queuing structures 13. The network interface controller 10 is coupled to a transmission medium 14 through a media arbitration circuit 15. A path from transmit buffering and queuing structures 12 is provided across line 19 to the media arbitration circuit 15. Also, received data is supplied on line 16 to receive buffering and queuing structures 13. The unique MAC address 18 and address filter logic 17 typically exist on the network interface controller 10, but are not used when the device is operating in the promiscuous mode.

The buffering and queuing structures 12 and 13 are coupled to the link layer interface processing modules 20 in the processor 11. The link layer interface modules 20 are coupled with protocol layer modules 21 for the bridging and routing functions. These functions 21 are coupled to other local area networks or wide area networks as represented by block 22. Also, the protocol layer modules 21 may be coupled to a network management resource block 23 executed by the processor 11.

The software is layered in this design, with processing required to differentiate routed packets, bridged packets, and network management packets. Since bridging requires checking every packet, the software executed by processor 11 must in fact process all packets on the LAN.

FIG. 2 shows the software algorithm for the bridging section of the prior art device of FIG. 1. The algorithm shown in FIG. 2 begins with a receive packet block 30. After the upper layer software receives the packet from the buffering and queuing structures, the software first determines whether the packet has a local destination (block 31). This determination is made with reference to a MAC address data base 32 which has been developed by the software. If, in block 31, it is determined that the packet has a local destination, then the software discards the packet (block 33). If, however, the packet has a remote destination, then the software forwards the packet to the remote LAN (block 34). After blocks 33 or 34, as appropriate, the end of packet processing is reached (block 35). Thus, every packet must be examined and either forwarded or discarded based on the destination MAC address. The bridge "learns" the unique MAC addresses of devices on the local LAN by examining the source address field of every packet and creating a database of them. This database is then used by the bridge to filter packets which are local. Since every packet must be examined by the software, the processor of a bridge must have sufficient capacity or else packets may be missed or "dropped" and not forwarded when they should have. These dropped packets are usually recovered by upper layer protocol software on the end stations, but they cause performance and response problems, and can lead to "thrashing", when ES devices start sending more packets to recover the lost packets, overloading the bridge even more.

Using the present invention, the performance of a Bridge or Router can be enhanced significantly. FIG. 3 shows a new design using the invention. The system includes a network interface controller 50 coupled to a host processor 51. The host processor 51 includes network management resources 52, intermediate system routing function resources 53, and intermediate system bridging function resources 54. The routing and bridging functions 53, 54 are coupled to other local area networks or wide area networks as represented by block 55. The network management resources 52 are coupled to a link layer interface module 56 for transmitting and receiving packets through buffering and queuing structures 57 and 58 to the network interface controller 50. Similarly, the intermediate system routing functions 53 are coupled to link layer interface resources 59, which are, in turn, coupled through buffering and queuing structures 60 and 61 for receiving and transmitting data through the network interface controller 50. Also, the intermediate system bridging functions 54 are coupled to a link layer interface module 62, which is, in turn, coupled through buffering and queuing structures 63 and 64 for receiving and transmitting data through the network interface controller 50.

The network interface controller 50 is coupled to a transmission medium 70 which receives and transmits data across line 71 to media arbitration circuit 72. The media arbitration circuit 72 merges the transmit data paths 73, 74 and 75 from the transmit buffering and queuing structures 57, 60, and 63 for the respective link layer interfaces 56, 59, and 62 in the host processor 51. Received data from the media arbitration circuit 72 is supplied across line 76 to address filter logic 77. The address filter logic is coupled to a memory which stores an assigned MAC address 78 for the intermediate system functions, and an assigned MAC address 79 for the network management functions, and multiple, non-unique, variable MAC addresses 80 used for additional filtering functions. Received packets are thus passed or blocked in response to the assigned addresses, 78, 79 and in response to the additional addresses 80. Thus, the

address filter is coupled across line 81 to buffering and queuing structure 58 for network management destined packets. The address filter supplies data across lines 82 and 83 to the buffering and queuing structures 61 and 64 for the routing and bridging functions respectively.

By programming the DLL's address array 80 with addresses of local ES devices, and blocking packets based on their destination address matching an entry in this array, the number of packets that the bridge software must deal with is reduced significantly. By also using the array to compare source addresses in packets, the learning function of the bridge can also be greatly improved. In addition, by using several unique MAC addresses, the network management, IS routing functions, and bridging functions of the device can be separated as well.

FIG. 4 shows the new bridging software logic taking advantage of the invention. The new algorithm shown in FIG. 4 includes a receive packet block 90. The first step for a received packet is to determine whether the packet is local (block 91). This step is carried out in response to a MAC address database 92 generated by the bridging software logic. If the packet is local, then the algorithm branches to block 93 where the packet is discarded. After the packet is discarded, the address array 80 in the network interface controller is updated (block 94). If at block 91 it was determined that the packet was destined to a remote network, then it is forwarded to the remote LAN (block 95). After blocks 94 or 95, as appropriate, the packet processing has been completed by the bridge software (block 96). Although there is an extra step (block 94) in the logic, the vastly fewer number of packets that must be processed using the invention nets a significant reduction in processor bandwidth required. In addition, the update step may be configured to happen during "deadtime" when no other CPU activity is likely to occur.

By storing source/destination address pairs in the DLL, a problem in the usage of multicast packets by many protocols suites can be overcome. For these protocols, a single multicast address is used for all group messages, and the particular upper layer usage (such as name resolution) is configured in the data portion of the packet which is not accessible by the DLL. This makes blocking of specific multicasts impossible. However, by having a list of address pairs, a DLL can be configured to block all multicast packets from all computers except the ones it cares about.

II. Virtual Data Link Devices

Because computers today are built with a single DLL and are assigned a single MAC address, the upper layers (3-7) of the computer must be more complicated when plural functions access the network. According to the invention multiple virtual DLL's are provided, each with their own unique MAC address assigned and optionally each with their own data path to the DLL. This allows a computer to be much simpler by having each major upper layer module have exclusive access to its own virtual DLL. An example of the improvement of a computer based on this is as follows.

In an integrated IS/ES device with several protocols, a unique MAC address for each protocol module and for the IS module allows a packet to be passed directly to the correct software module instead of the costly (in terms of CPU bandwidth and time spent) RRPI method mentioned in the Description of Related Art above. This technique is also very valuable in multi-processor systems, allowing upper layer software modules to execute on different CPUs and have

11

their packets transferred to them at the physical layer instead of at an upper layer of software which is again expensive in terms of CPU bandwidth and time spent.

FIG. 5 below shows a prior art system architecture of an integrated IS/ES multiprocessor device, illustrating the data flow and software modules.

The integrated IS/ES device in the prior art includes a network interface controller 100 coupled to a main processor 101. The main processor is in turn coupled to a co-processor 102.

The network interface controller 100 is coupled to a transmission medium 103 through media arbitration circuit 104. Media arbitration circuit 104 passes received data across line 105 to an address filter 106 which stores a unique MAC address 107. Packets which pass the filter 106 are supplied through buffering and queuing structures 108 to round robin protocol identification module 109.

Packets to be transmitted from the main processor 101 or co-processor 102 are passed through buffering and queuing structures 110 across path 111 to the media arbitration circuit 104.

In the embodiment shown in FIG. 5, the main processor 101 executes a plurality of processing modules 111, 112, 113 which execute independent upper layer protocols. Each includes a protocol link layer interface 114, which is coupled to a protocol intermediate system layer for routing functions or the like 115, and on to the protocol upper layers for end system functionality.

Similarly, the co-processor 102 executes a processing module 116. Processing module 116 includes the protocol upper layers for end system functionality which are coupled to a protocol intermediate system layer 117. The protocol intermediate system layer 117 is coupled to interprocessor communication module 118. This module communicates with a protocol link layer interface module 119 executed by the main processor 101.

Thus, the round robin protocol identification module 109 routes received packets to the respective protocol link layer interfaces (e.g., 119, 114) of the processing modules 116, 111, 112, 113. Packets transmitted by the respective processing modules (116, 111, 112, 113) are passed through the buffering and queuing structure 110 directly to be transmitted through the media arbitration circuit 104.

A co-processor/main processor system modified according to the present invention is illustrated in FIG. 6. As shown in FIG. 6, the integrated IS/ES system according to the present invention includes network interface controller 150 which is coupled to a main processor 151 and a co-processor 152. Main processor 151 is coupled to the network interface controller 150 through buffering and queuing structures 153 and 154. The co-processor 152 is coupled to the network interface controller 150 through buffering and queuing structures 155 and 156.

The main processor executes a plurality of processing modules 157, 158, 159 which include a protocol link layer interface 160, coupled to protocol intermediate system layer 161, which is in turn coupled to the protocol upper layers for end system functionality for the respective modules. Each of the modules 157, 158, 159 is coupled directly to the transmit buffering and queuing structure 154 and directly to the receive buffering and queuing structure 153 to establish a data channel to the network interface controller.

The co-processor 152 also executes a processing module 163. This processing module includes protocol link layer module 164, protocol intermediate system layer 165, and

12

protocol upper layers for end system functionality as before. However, by incorporating the protocol link layer interface 164 in the co-processor, and connecting the protocol link layer directly to the buffering and queuing structure 155 for transmitted frames and the buffering and queuing structure 156 for received frames, an independent data channel is established for the co-processor through a separate data path, without reliance on an interprocessor communication module.

The network interface controller 150 of FIG. 6 is coupled to a transmission medium 170 through media arbitration circuit 172. Received data is coupled through address filter 173 which stores multiple MAC addresses as described above in memory 174. Three data channels are coupled through a path that comprises the buffering and queuing structure 153 to the respective modules 157, 158, 159 of the main processor. A separate data channel is coupled through buffering and queuing structure 156 from the address filter 173 to the module 163 executed by the co-processor. Transmitted frames are supplied from the three modules of them in processor 151 through the buffering and queuing structure 154 to the media arbitration circuit 172. Also, transmitted frames are supplied from the co-processor module 163 through the buffering and queuing structure 155 to the media arbitration circuit 172.

As can be seen, the upper layer system is much simplified for the integrated IS/ES system, and the upper layer interprocessor communications and round robin protocol identification modules are no longer required.

FIG. 7 illustrates another application of the present invention, in which the integrated computer separates the end system and intermediate system functions in hardware, providing each function its own processor. Thus, as can be seen in FIG. 7, the system is coupled to a transmission medium 200 through media arbitration circuit 201. Received frames are supplied across line 202 to an address filter 203. The address filter 203 is responsive to a first assigned MAC address 204 for the end system processor and a second assigned MAC address 205 for the intermediate system processor. Received frames destined for the intermediate system are coupled from buffering and queuing structure 206 to intermediate system processor 207. Received packets destined through the end system are coupled to buffering and queuing structure 208 to the end system processor 209. Similarly, frames transmitted from the intermediate system are coupled through buffering and queuing structure 211 to the media arbitration circuit 201. Packets transmitted by the end system portion of the device 209 are coupled to buffering and queuing structure 212 to the media arbitration circuit 201.

The intermediate system 207 is coupled through at least one other network interface to other networks such as a wide area network or local area network medium as represented by block 213.

The design of prior art integrated IS/ES systems is such that the system shares hardware and a single unique MAC address for each LAN it is attached to, and the upper layer software must determine, based on the data portions of the packet, whether the packet is for the local ES portion of the system, for the IS portion, or for both. The present invention provides a hardware MAC solution which allows for two or more unique MAC addresses, and hardware level routing of packets based on the MAC destination address, with one path (208) getting only broadcast packets and packets with one unique address dedicated to the ES portion of the system, and the other path (206) getting either all packets (in

the case of IS bridging functionality) or broadcast and directed packets to a unique MAC address dedicated to IS routing functionality in the system. The IS portion is ideally a co-processor allowing CPU separation of the IS and ES functionality. This new method of integrating ES and IS functionality is particularly attractive for Routing Adapters in a Boundary Routing environment as described in co-pending U.S. patent application by John Hart, Ser. No. 07/871,113, Filed Apr. 20, 1992, entitled SYSTEM FOR EXTENDING NETWORK RESOURCES TO REMOTE NETWORKS. In the routing adaptor, the CPU and RAM requirements in terms of size, complexity and performance for the co-processor dedicated to the IS functionality are minimal and can be integrated with the new MAC hardware onto a single PC card.

As illustrated in FIG. 8, a routing adapter 241 executes functions independent of the protocol suites encapsulated in LAN frames received/transmitted across boundary LAN 223 and link 238 to which it is attached. The routing adapter functionality consists of boundary link interface 220 coupled to a co-processor 244 which executes boundary relay functions 221, and routing adapter management functions 224. In addition, a multiple MAC address network interface controller 222 such as described above, is coupled across a first interface 246 having a unique MAC address to the co-processor 244 and across a second interface 247 having a unique MAC address to the main processor 245.

The boundary link interface function 220 is positioned between a boundary link 238 and boundary relay function 221. The boundary link interface 220 in the routing adapter 241 works with its peer boundary link interface function 237 in the boundary router 240 and is responsible for transmitting and receiving frames to and from the boundary link 238. The functionality of the boundary link interface 220 is essentially identical to the boundary link interface 237 in the boundary router 240.

A boundary router, as shown in FIG. 8, includes at least one local LAN interface 230 for attachment to a local LAN 231. There is one local LAN interface for each attached LAN, as indicated in the figure. Each local LAN interface will be given a LAN address for use by the routing resources on the boundary router. Coupled to the local LAN interface is a decapsulation/encapsulation function 232, also one for each attached LAN. The decapsulation/encapsulation function 232 is coupled to router management 233 and multi-protocol relay 234 functions which are implemented for each routed protocol. Extensions to the boundary router to serve the remote network include boundary router management 235, a boundary function 236, and a boundary link interface 237. The boundary link interface 237 is connected to a boundary link 238 which provides communication with a boundary link interface 220 on the routing adapter 241.

Thus, a boundary router contains all the logic of a multiprotocol router (such as NETBuilder, available through 3COM Corporation, Santa Clara, Calif.) plus boundary functionality for the boundary links that interconnect the boundary router to the routing adapter. The additional functionality consists of boundary router management 235, boundary function 236, and the interface to the boundary link interface 237.

In the routing adaptor, the multiple MAC address network interface controller 222 includes a first interface 246 positioned between the boundary LAN 223 and boundary relay 221 on the co-processor. The interface 246 of the interface controller 222 is responsible for transmitting and receiving frames to and from the boundary LAN 223 for the routing

adaptor. The functionality of the network interface controller 222 for interface 246 includes the following:

1. handling the physical and data link protocols, etc., as defined by the boundary LAN 223;
2. transmitting frames relayed by boundary relay function 221; and
3. passing valid received LAN data frames to the boundary relay function 221 which have a destination address within a programmed set of addresses including the address of the interface 246 which provides an extended remote interface to the boundary router, or group address(es) set by routing adapter management function.

The boundary relay function 221 includes the adaptor's frame relay logic and operates independent of higher level protocol suites. The frame relay logic of a routing adapter 201 is defined by the following two rules.

1. Any frame passed from the boundary LAN 223 to the boundary relay 221 is forwarded to its boundary link interface 220 unless link 238 is not operational. In this case, it may be network management frame and it is passed to the routing adapter management function 224. This allows the routing adapter to be managed locally when the link is not operational. For instance, the routing adapter management 224 may respond to management frames which request an attempt to re-open a link, such as by attempting re-dials on dial links.
2. Any frame received from its boundary link interface 220 is forwarded to the network interface controller 222 through interface 246 unless its destination equals the routing adapter's LAN address. In this case, it is a network management frame from the boundary router management function 235 and it is passed to the routing adapter management function 224.

Routing adapter management 224 maintains local configuration information such as the LAN type of the boundary LAN 223 and the multicast destination addresses to be received.

Also, the routing adapter management 224 operates as the agent of the boundary router management function. As such, it is responsible for processing and responding to management requests, responses, etc., received from it.

Further, the routing adapter management 224 is responsible for processing and responding to management requests, responses, etc., received from end systems on the boundary LAN 223 when the boundary link 238 is not operational.

The network interface controller 222 also provides an interface 247 having a unique MAC address to the boundary LAN for the main processor 245, which executes such end system functions as suits a particular system unencumbered by the processes required to identify frames needed by the routing adaptor functions of the co-processor 244.

FIG. 9 illustrates a printed circuit board incorporating a network interface controller according to the present invention. The circuit board 700 includes a host bus connector 701, a transmission medium connector 702, and an integrated circuit transceiver 703 adapted to receive and transmit communications through the connector 702. Coupled with the transceiver 703 are a plurality of non-volatile memory cells, such as EEPROMs 704, 705, 706, and 707 storing respective MAC addresses for use by address filtering logic in the integrated circuit 703. Also, a program store 708 for optional remote initial program load (RIPL) of the operating system of the computer system the board is inserted into. The chip 703 communicates across line 709 in

15

a plurality of channels to CPU and associated logic 710 on the board. This logic 710 is coupled through an SIO/modem chip set 711 to a telecommunications link 712. Also, memory 713 is coupled to the CPU and associated logic 710. This configuration is particularly suited to the routing adapter embodiment where the CPU serves as the co-processor of FIG. 8.

FIG. 10 illustrates the network interface controller 703 used in the system of FIG. 9. This device 703 includes the media arbitration circuit 750 and address filtering logic 701 which is coupled to the MAC address registers 704 through 707. The address filter 751 supplies the received data to intermediate system modules in the CPU and associated logic 701 across line 752 and receive data to the PC processing modules across path 753. Transmit data is received from the intermediate system on line 754 and from the PC system along line 755.

As illustrated in FIG. 10, there are three MAC addresses assigned to PC processing modules. These MAC addresses are presented across line 753 as virtual data link layer devices which share a physical data path to the PC. However, this data path is divided into independent channels for communicating with data link layer processes from the point of view of the PC software.

III. Media Arbitration

Because wiring distribution systems today such as Ethernet's 10BaseT were not designed to handle devices with multiple MAC addresses and multiple data paths, the present invention provides a new medium arbitration interface scheme which functions correctly with standard wiring systems.

DLLs today do not receive the same packet that they are transmitting on the media. With a single DLL this is not an issue because the upper layers of the computer know when they send a packet. However, with virtual DLLs, when one module sends a packet, other modules might be the recipients of that packet. The invention provides the ability to receive packets that are being transmitted through virtual DLLs as if they came from a different computer.

FIGS. 11, 12 and 13 illustrate three alternative media arbitration circuits according to the present invention for Ethernet type interfaces. This technology may be applied as well to other network types, such as token rings, token buses, FDDI, ISDN, etc., with appropriate modifications to serve the multiple virtual DLLs to which the arbitration circuits are connected.

FIG. 11 illustrates a media arbitration circuit which merges the data paths at the attachment unit interface AUI. The system includes a medium attachment unit transceiver 300 coupled to a medium 301 of the network. The medium attachment transceiver 300 is any one of a number of commercially available integrated circuits which provide the medium attachment unit MAU services to a particular medium 301, and generates the signals to drive a standard attachment unit interface AUI. Thus, the output signals of the transceiver 300 include a plus and minus collision pair 302, a plus and minus receive pair 303, and a plus and minus transmit pair 304.

Each of the virtual DLLs served by the media arbitration circuit presents a standard AUI interface to the media arbitration circuit. Thus, the system includes AUI 1 306, and AUI 2 307. AUI 1 306 includes transmit pair 308, receive pair 303, and collision pair 310. AUI 2 includes transmit pair 311 and receive pair 303. AUI 2 shares the collision pair 310 with AUI 1.

16

The transmit pairs 308 and 311 are merged in summing amplifier 305 to generate transmit pair 304 for the transceiver 300. Coupled with the summing amp is collision detection logic 326. The collision detection logic 326 is coupled to a collision oscillator 327. The collision detect logic 326 signals the oscillator 327 across line 328 when a collision is detected. The output of the collision oscillator 327 is supplied on line 329 (comprising a plus and minus pair as known in the art), beginning when both the transmit pairs 308 and 311 are transmitting through the summing amp 305, and ceasing when both cease transmitting.

The receive pair 303 delivers the receive signal to interfaces AUI 1 306 and AUI 2 307. The MAU transceiver 300 echoes data sent on transmit pair 304 via receive pair 303, thus providing a receive path for both receivers to receive transmissions sent by the other.

The collision pair 310 at AUI 1 and AUI 2 is generated at the output of summing amp 332. The inputs to the summing amp 332 include the output of oscillator 327 on line 329 and the collision pair 302 generated by the transceiver 300.

Thus, it can be seen that packets transmitted through AUI 1 are received by AUI 2 and vice versa. Also, collisions between transmit pairs 308 and 311 are detected in the media arbitration circuit. Finally, the summing amplifiers isolate the transceiver outputs 300 from the separate virtual DLL interfaces 306 and 307.

FIG. 12 illustrates an embodiment of media arbitration circuit adapted for connection to a BNC type Ethernet. In this system, a coaxial cable 400 meeting the BNC Ethernet standards is tapped with a connector generally 401. The connector 401 supplies ground line 402 and signal line 403. The signal line 403 is coupled to a first transceiver 404 and a second transceiver 405. Ground line 402 is coupled to the DC to DC converters 406 and 407 which are coupled respectively to transceivers 404 and 405. The DC to DC converters 406 and 407 supply -9 Volt power to the transceivers 404 and 405 as known in the art. The transceivers 404 and 405 generate the 6 outputs of an AUI interface, which are supplied through transformers 408 and 409, respectively to the separate AUI interfaces, AUI 1 410, and AUI 2 411, for the separate virtual DLLs according to the present invention.

FIG. 13 illustrates the media arbitration circuitry for a 10BaseT standard Ethernet connector using twisted pair transmission media. Thus, a 10BaseT connector 500 supplies a receive pair 501 and receives a transmit pair 502. The receive pair 501 is supplied to a first summing amp 503 and a second summing amp 504 which drive respective receive pairs 505 and 506 for medium attachment unit transceivers 507 and 508, respectively. The transceivers 507 and 508 are coupled to the data paths in the network interface controller as known in the art, may or may not provide the standard AUI type interface.

The transmit pair 502 of the system is driven by summing amp 509. The inputs to the summing amp 509 include the transmit pair 510 generated by MAU 1 507 and the transmit pair 511 generated by MAU 2 508. The summing amp includes collision detection logic 512 which is coupled to a collision oscillator 513. When both transmit pairs 510 and 511 begin transmitting, the collision oscillator is signalled across line 514 and begins generating a collision signal on line 515. When both transmit pairs 510 and 511 cease transmitting, the collision signal on line 515 is turned off.

The inputs to the summing amps 503 and 504 which drive the receive pairs 505 and 506, respectively, receive the collision signal on line 515.

17

In addition, summing amp 503 which drives the receive pair 505 for MAU 1 507, receives the transmit pair 511 from MAU 2 as input. Summing amp 504 which drives the receive pair 506 receives in addition to the receive pair 501 and the collision pair 515, and the transmit pair 510 supplied by the MAU 1 507.

Thus, it can be seen that the circuit of FIG. 13 provides the ability to receive packets transmitted by neighbor virtual DLLs, isolates the connector 500 from the separate virtual DLLs, and presents a standard interface to the address filter and transmit buffering and queuing structures.

IV. General Purpose Programmable Ethernet NIC

An Ethernet Network Interface Controller NIC based on the invention is described with reference to FIG. 14 and Tables 1, 2, 3, and 4 set out in an appendix. The NIC has a table which can hold up to 511 MAC addresses, and provides two virtual DLL interfaces to the upper layers of the computer. It has a 512x512 address pair index array, allowing any packet with a particular pair of source and destination addresses in the address table to be specifically blocked or passed for each of the two virtual DLLs. The address pair table is implemented using an inexpensive 256Kx4 external (to the NIC) RAM. The NIC provides two separate 32 bit full duplex data paths to external devices in the computer and has a common shared programming interface. The device has a number of registers for each of the two DLL interfaces (A and B) which control such actions as how a multicast should be treated, or what to do if the entry for a source destination implies the packet should be blocked, and the entry for a destination implies it should be passed.

FIG. 14 illustrates control structures in the device. Table 1 shows the algorithms described below for determining if a packet should be passed or blocked. The registers of the device are listed in Table 2, Table 3, and Table 4.

A portion of the NIC is schematically shown in FIG. 14 to illustrate the control structures. Incoming data is received on line 600 and supplied through a buffer 601. In the buffer 601, the source address is detected and supplied on line 602, and the destination address is detected and supplied on line 603. The data is supplied out of the buffer 601 to the data paths in the NIC on line 604.

The address filter includes an address list, generally 605, designed to hold a plurality of MAC addresses in addition to the assigned MAC addresses for the two data paths. The address list in the preferred embodiment includes a set of 16 32x48 bit RAMs for a total of 512 entries. Coupled with each in RAM 605 is a source index counter 606 and a destination index counter 607. The 16 RAMs 605 and counters 606, 607 are driven in parallel. A single module of the address list in address list counter logic is illustrated in the figure for simplicity. The output of each address list RAM 605 is supplied to a comparator 608 which receives as a second input the source address on line 602, and a comparator 609 which receives as a second input the destination address on line 603. The outputs of the comparators 608 and 609 are fed back as indicated for use in controlling the searching logic. Also, these outputs are supplied to an address match register 610 which stores two bit results of the matching process.

When a match is found, the value of the source index counter for a matching source is stored in the index source register 611, and the value of the destination index counter 607 is stored in index destination register 612. The five bit counter values for the 32 bit blocks are appended on a four

18

bit block number, establishing 9 bit source index and a 9 bit destination index. The source index 611 and destination index 612 are concatenated to produce an 18 bit address used to access an index array 613. The output of the index array is supplied to registers 614 for establishing filter control parameters. The filter control parameters from register 614 are supplied on line 615 to filter control logic 616. Other inputs to the filter control logic include the match result register 610, a conflicts default register 617 which is described below, and an address type register 618. The address type register indicates whether the destination address on line 603 is a unicast or a multicast address.

The NIC also includes registers for storing assigned MAC addresses for channels A and B referred to as MAC A 620 and MAC B 621. Coupled with these registers are respective comparators 622 and 623, which compare the incoming destination address on line 603 with the stored assigned MAC addresses for the respective paths A and B. The results of these matches are used to determine whether to pass the incoming packet to the respective channels as indicated on line 624 and 625. If match line 624 NOR match line 625 is asserted, as indicated by the output of NOR gate 626 in the figure, then the address filtering logic including the counters and the logic 616 are enabled.

The NIC is designed to hold 510 MAC addresses in address list RAMs 605, and to have a matched address pair lookup table 613 that contains entries for each matched source and destination pair, plus entries for default cases and cases where there is one address match but not the other. This is done by having a 512x512 index array 613. Smaller or larger NIC designs could also be done, with the most optimal implementations being an address list of 2^n-2 and an array size of $(2^n)^2$. The following constants are used when describing entries in the array. The first two are the indexes of the first and last entry in the address list, and the second two are for entries in the array to deal with default and no match cases:

```

Index_min = 0
Index_max = 509
Index_no_match = 510
Index_default = 511
    
```

The NIC has two external address ports for assigning unique MAC addresses to registers 620 and 621 during the manufacturing process. These addresses are available to the programmer as two 48 bit read-only registers:

```

MACA
MACB
    
```

The device has an internal programmable array 605 of 510x48 bit MAC addresses.

```

AddressList[Index_min .. Index_max]
    
```

There is also a set of 24 bit source match and a set of 24 bit destination match non-overflowing counters (not shown), used for statistics. These are incremented each time an address comparison succeeds, and are accessible by upper layers of the computer.

In order to speed up processing time, the AddressList array is actually implemented as 16 different 32x48 bit memory cells 605, each with two comparators 608, 609 for source and destination:

Count_{src}[Index_{src}, . . . Index_{max}]

Count_{dst}[Index_{src}, . . . Index_{max}]

As each packet is received, the source and destination addresses of the packet are each compared to each entry in the array. If a match occurs, a 9 bit index register (611 or 612) is set with the number (0 . . . 509) of the array the match was found in, otherwise the register is set to 511 to signify that no match was found. The two registers for source and destination matches are:

Index_{src}

Index_{dst}

In addition there is a single 2 bit register "AddressMatch" 610 which is used to signify whether a match occurred. Its values are:

0=No match for either address found in AddressList[]

1=source address matched only

2=destination address matched only

3=both address-matched

The destination address of each packet is examined and its type is noted in a 1 bit boolean register "MultiCast" 618, whose values are defined as follows:

FALSE (0) =unicast address

TRUE (1) =multicast address

After the source and destination index registers are loaded, they are used to access five 4 bit data cells from an external 256Kx4 bit memory chip 613. This chip 613 is addressed using the combined 18 bits of the two index registers 611, 612 (2⁹*2⁹=512*512=262144=256K). The external memory is referred to as:

MatchArray[0 . . . 511, 0 . . . 511]_A (lower two bits of each 4 bit memory cell)

MatchArray[0 . . . 511, 0 . . . 511]_B (upper two bits of each 4 bit memory cell)

The five 4 bit data cells returned are stored in ten 2-bit registers 614 whose entries correspond to locations in MatchArray and two global default registers 650. The first two are used for the cell containing an address pair match:

Match_{SD,A}=MatchArray[Index_{src},Index_{dst}]_A

Match_{SD,B}=MatchArray[Index_{src},Index_{dst}]_B

The Match_{SD,A} and Match_{SD,B} registers are defined as follows:

- 0 (00) Empty (i.e., use defaults)
- 1 (01) Undefined
- 2 (10) Pass packet

-continued

3 (11) Block packet

The next four registers are used as default control values when both source and destination address matches occur, but the Match_{SD,A} and/or Match_{SD,B} registers are "Empty". They are:

Match_{SD,Src,A}=MatchArray[Index_{src},Index_{def(src),A]}

Match_{SD,Src,B}=MatchArray[Index_{src},Index_{def(src),B]}

Match_{SD,Src,A}=MatchArray[Index_{src},Index_{def(src),A]}

Match_{SD,Src,B}=MatchArray[Index_{src},Index_{def(src),B]}

The next two registers are used for determining action when a match of the source address occurs but not the destination address:

Match_{S,A}=MatchArray[Index_{src},Index_{max}]_A

Match_{S,B}=MatchArray[Index_{src},Index_{max}]_B

The last two registers are used for a match of the destination address but not the source address:

Match_{D,A}=MatchArray[Index_{max},Index_{dst}]_A

Match_{D,B}=MatchArray[Index_{max},Index_{dst}]_B

Two programmable 2 bit registers 650 are used to determine a packer's disposition if neither the source or destination addresses in the packet are found:

Match_{src,A}

Match_{src,B}

With the exception of the first two control registers defined above (Match_{SD,A} and Match_{SD,B}), all of the Match control registers are defined as follows:

- 0 (00) Pass packet
- 1 (01) Block packet if it is a Unicast and Pass if it is a Multicast
- 2 (10) Pass packet if it is a Unicast and Block if it is a Multicast
- 3 (11) Block packet

When both source and destination addresses are matched and MatchArray[Index_{src},Index_{dst}] is not Empty, then the decision to block or pass the packet is always made based on the contents of the entry in MatchArray. If the entry for the source and destination pair is empty, then a conflicts default register 651 is used to pick which address has precedence in case the actions for each address are different. To resolve these conflicts between pass and block instructions based on source and destination addresses, four default registers 651 are used. The registers are:

fUseDst_{U,A}

fUseDst_{U,B}

fUseDst_{M,A}

fUseDst_{M,B}

65

21

The two U registers are used if the destination packet is a UniCast (fMultiCast=FALSE), and the two M registers are used if the destination packet is a MultiCast (fMultiCast=TRUE). The two A registers are for the first DLL interface, and the two B are for the second DLL interface. In this case where both addresses are matched, but the match cell is empty, the fUseDst is used to determine which address has priority. If fUseDst $_{SD}$ is TRUE, then the entry Match $_{SD,dst}$ (from MatchArray[510,Index $_{dst}$]) is used, and if fUseDst $_{SD}$ is FALSE, then the entry Match $_{SD,src}$ (from MatchArray[Index $_{src}$,510]) is used.

If only the source address matches then Match $_{s,n}$ (=MatchArray[Index $_{src}$,511] $_{,n}$) is used, and if only the destination address matches then Match $_{d,n}$ (=MatchArray[511,Index $_{dst}$] $_{,n}$) is used.

In the case where no match is found, the Match $_{xx,A}$ and Match $_{xx,B}$ default registers 650 are used to determine a packet's disposition. A summary of the registers, arrays, and constants is contained in Table 2.

BRouter Design

Using the NIC described above with reference to FIG. 14, the following is a description of a very high performance integrated IS/ES computer. Virtual DLL and data path A of the NIC and allocated to the IS portion of the computer, and virtual DLL and data path B are allocated to the ES portion of the device.

The list of MAC addresses (605) is initialized so that all entries are "Empty":

```

FOR a = Index $_{min}$  to Index $_{max}$  DO;
  AddressList[a] = 0;
  Count $_{src}$ [a] = 0;
  Count $_{dst}$ [a] = 0;
END;
    
```

The IS portion of the computer is assigned to virtual DLL A, reserve table entry 0 for it, and assign it its own unique MAC address:

```
AddressList[0]=MAC $_A$ ;
```

22

The Match Array is initialized so that all matched pair entries are configured as "Empty" for the IS module:

```

FOR s = Index $_{min}$  to Index $_{max}$  DO;
  FOR d = Index $_{min}$  to Index $_{max}$  DO;
    MatchArray[s,d] $_A$  = 0; /* Make all entries "Empty" for
                               IS */
  END;
END;
    
```

Configure the MatchArray 613 so that any packet with destination address MAC $_A$ will always be passed to the IS module, and any packets originating from the IS module (source address =MAC $_A$) will be blocked from being received by the IS:

```

MatchArray[511,0] $_A$  = 0; /* pass on data path A all packets with
                           destination address MAC $_A$  */
MatchArray[0,511] $_A$  = 3; /* block on data path A all packets with -
                           source address MAC $_A$  */
    
```

The second address entry in the table 605 is that of the ES (DLL B), and the IS module on DLL A treats that address as a local computer on the LAN, and thus blocks any packet with that MAC address as a destination. If both the source and destination addresses are in the table we do not need to learn the source address, and decide action based on the destination address:

```

Match $_{xx,A}$  = 0; /* unknown source and destination address: pass
                  always */
fUseDst $_{U,A}$  = 0;
fUseDst $_{M,A}$  = 0; /* Decide based on source address if both are in table
                  and table entry is Empty */
    
```

The ES module's MAC address is treated as a local computer on the LAN, and the array is configured so that packets from other computers to the ES module (with MAC $_B$ destination address) are blocked.

Packets from the ES module will be passed if learned. Every time the IS module learns a new address that it wants to block or pass as a result of, an array entry is made. In the cases where either the source or destination addresses are unknown, we pass the packet to the IS module for processing:

```

FOR i = 1 to 510
  MatchArray[i,511] $_A$  = 0;
  MatchArray[511,i] $_A$  = 0; /* pass packet */
END;
    
```

The IS module is initially programmed as a standard BRouter and receives all multicasts, all packets with unknown destination addresses (to forward to other LANs), and all packets with unknown source addresses (to make note of that new source address as being on the local LAN). In addition, any Unicast packets with destination address MAC $_A$ will always be passed to the IS module, and all packets originating from the IS module (source address=MAC $_A$) will be blocked from being received by the IS module.

```

MatchsrcA = 0; /* Packets with neither source nor destination
addresses in table are always passed*/
MatchArray[Indexdefault,0]A = 0;
MatchArray[Indexpair,0]A = 0; /*Pass on data path A all packets with
destination address MACA */
MatchArray[0,Indexdefault]A = 3;
MatchArray[0,Indexpair]A = 3; /*Block on data path A all packets with
source address MACA */
For i = 2 to Indexmax DO;
    MatchArray[Indexdefault,i]A = 1;
    MatchArray[i,Indexdefault]A = 1;
    END; /*Initialize tables so that the default is to block
packets in which both source and destination are in
the table unless they are multicasts */
FOR i = 2 to Indexmax DO;
    MatchArray[Indexpair,i]A = 0;
    MatchArray[i,Indexpair]A = 0;
    END; /*Initialize tables so that the default is to pass
packets in which either source or destination address
is not in the table */
fUseDst0,A = 1;
fUseDstM,A = 1; /*Decide based on default destination address entry
if both source and destination addresses are in table
but table entry is empty */

```

When an address or address pair is to be added or removed from the table, it will typically be one of the 25 following cases:

- 1) A new MAC address of a computer on the local LAN was discovered by seeing a new source address (MAC_{new}) in a packet. In this case the IS module wants to block all packets with this destination address. If the address table AddressList[] is full, an older entry will have to be removed using an aging algorithm of some kind. After that is accomplished if necessary and the location in the table determined (Index_{new}), the NIC is initialized for that entry as follows:

```

AddressList[Indexnew] = MACnew;
Countsrc[Indexnew] = 1;
Countdst[Indexnew] = 1; /*add new MAC address to table and set use
counters to show non-empty entry */
MatchArray[Indexdefault,Indexnew]A = 3; /*Block packets in which
both source and destination are in the
table and the destination is this new
address */
MatchArray[Indexpair,Indexnew]A = 0;
MatchArray[Indexnew,Indexpair]A = 0; /*Pass packets with unknown
source address and new destination
address of with unknown destination
address and this new source address*/
FOR i = Indexmin to Indexmax DO;
    MatchArray[i,Indexnew]A = 0;
    MatchArray[Indexnew,i]A = 0;
    END; /*Make all of the matched pair entries
for this address "empty" */

```

- 2) Occasionally a multicast (MAC_{mc}) will be desired to be blocked by the network administrator. After finding an entry (Index_{new}) as above, this is done as follows:

```

AddressList[Indexnew] = MACmc;
Countsrc[Indexnew] = 1;
Countdst[Indexnew] = 1; /*add new MAC address to table and set use
counters to show non-empty entry */
MatchArray[Indexdefault,Indexnew]A = 3; /*Block packets in which
both source and destination are in the table
and the destination is this multicast */

```

```

MatchArray[Index_min, Index_max]_A = 0; /*Pass packets with unknown
source address so that they can be learned
*/
For i = Index_min to Index_max DO;
  MatchArray[i, Index_max]_A = 0;
  MatchArray[Index_min, i]_A = 0;
END; /*Make all of the matched pair entries for this address
"empty" */

```

10

3) Occasionally a network administrator wishes to see all packets to or from a particular computer (MAC_{computer}) in order to do network protocol analysis. After finding the existing entry (Index_{computer}) for the computer or a new entry as above, this is done as follows:

```

AddressList[Index_computer] = MAC_computer;
Count_in[Index_computer] = 1;
Count_out[Index_computer] = 1; /*add new MAC address to table and
set use counters to show non-empty
entry */
MatchArray[Index_dest, Index_max]_A = 0;
MatchArray[Index_min, Index_dest]_A = 0;
MatchArray[Index_min, Index_max]_A = 0;
MatchArray[Index_max, Index_min]_A = 0; /*Pass packets to or from the
computer */
FOR i = Index_min to Index_max DO;
  MatchArray[i, Index_max]_A = 0;
  MatchArray[Index_min, i]_A = 0;
END; /*Make all of the matched pair entries
for this address "empty" */

```

We assign the ES portion of the computer to virtual DLL B, reserve table entry 1 for it, and assign it its own unique MAC address, with an entry in the address list for it:

AddressList[1]=MAC_B;

The Match Array is initialized so that all matched pair entries are configured as "Empty":

FOR s = Index_min to Index_max DO;

-continued

```

FOR d = Index_min to Index_max DO;
  MatchArray[s,d]_B = 0; /* Make all entries "Empty"

```

-continued

```

END;
END;
for ES */

```

35

40

The ES portion of the computer is configured to receive only unicasts with its destination address and to receive all multicasts. Any Unicast packets with destination address MAC_B will always be passed to the ES module, any Multicast packets will be passed to the ES module, and all packets originating from the ES module (source address=MAC_B) will be blocked from being received by the ES:

```

Match_min_B = 1; /* Packets with neither source nor
destination addresses in table are passed only
if they are multicasts */
MatchArray[Index_dest, 1]_B = 0;
MatchArray[Index_min, 1]_B = 0; /*Pass on data path B all packets with
destination address MAC_B */
MatchArray[1, Index_dest]_B = 3;
MatchArray[1, Index_min]_B = 3; /*Block on data path B all packets with
source address MAC_B */
For i = 2 to Index_max DO;
  MatchArray[Index_dest, i]_B = 1;
  MatchArray[Index_min, i]_B = 1;
  MatchArray[i, Index_dest]_B = 1;
  MatchArray[i, Index_min]_B = 1;
END; /*Block packets not directed at ES unless they
are Multicasts */
IUseDst_min_B = 0;
IUseDst_max_B = 1; /*Decide based on source address entry for
unicasts and on destination address for
multicasts if both source and destination
addresses are in table but table entry is
empty */

```

The special cases of a unicast from the ES to the IS (B to A) or from the IS to the ES (A to B) are handled with entries in the matched pair table:

```

MatchArray[0,1]A = 3; /* block data from A to B from going
                    back to A */
MatchArray[0,1]B = 2; /* pass data from A to B */
MatchArray[1,0]A = 2; /* block data from B to A from going
                    back to B */
MatchArray[1,0]B = 3; /* pass data from B to A */

```

V. Multi-Media Network Interface Controller

The present invention is also useful in multi-media systems, which have unique data handling problems. There are two main types of multi-media applications, stored data (i.e., those that retrieve stored images and audio for playback) and real-time (i.e., those that display an image or generate sound as it is being recorded at another location). Audio data and image data are termed multi-media data herein.

For both applications, the three main parameters relative to the network interface controller are throughput, latency, and buffer size. If a network has enough throughput, then the latency and buffer size parameters become the dominant factors.

Audio typically requires 4K to 64K bits per second depending on quality. Video typically requires 64K to 512K or more bits per second depending on quality.

Ethernet has a maximum throughput of 5 Mbps (millions of bits per second), or about half of its 10 Megabit nominal speed. Similarly, token ring has either 2 Mbps (4 Megabit speed) or 8 Mbps (16 Megabit) depending on the version of TokenRing used. Any of these media types have enough bandwidth for multi-media for some number of users greater than one. For example, with an audio-visual multi-media application using data rates of 256K bps, an Ethernet local area network (LAN) could have up to 20 simultaneous users (5 Mbps/256 Kbps=20) per LAN segment.

Since LANs are shared services, the time it takes for data to be transferred from one machine to another is not a fixed time. The data transfer time over a LAN is examined in terms of mean time to transfer ± 2 standard deviations (± 2 SD), and applications must build in a fixed time delay that is equivalent to this maximum time or else the data delays will result in unpleasant breaks in the sound or video. This delay requires buffers to be allocated of a size equal to the delay time multiplied by the data rate. By reducing the latency, stored data applications require less memory, and real-time applications can be done with minimal time delay.

There are a number of factors which add latency in a system, including transfer time from recording device (real-time) or disk drive (stored-data), software time encapsulating data, protocol routing and link level interface software time, MAC queuing time, time waiting for shared media to become free, physical media transfer time, receiving MAC transfer time, queuing time, software time spent in link level interface, software protocol processing, application time handling the data, and device driver time displaying or replaying the data.

All of these factors must be minimized for real-time multi-media to be viable. This invention is a solution for the network interface controller to significantly reduce the amount of time (latency) needed to transmit or receive multi-media data.

FIG. 15 illustrates a multi-media system 800 which takes advantage of a multiple MAC address network interface

controller 801 according to the present invention. The network interface controller 801 is coupled to a network across line 802. A first interface of the network interface controller is coupled across line 803 to a main CPU 804. A second interface of the network interface controller is coupled across line 805 to a multi-media module 806. The multi-media module also communicates with the main CPU across line 807. The multi-media module particularly serves the function of routing audio and video data across line 808 to audio/video hardware 809.

By using a separate unique MAC address and data channel for the multi-media module of a computer, video and audio data can be processed without software overhead, greatly reducing latency. Audio/video data can be sent directly without protocol headers, if the session is local, or with minimal protocol overhead if the data is routed from another network.

Two alternate implementations of the interface controller 801 are shown in FIGS. 16 and 17. In the embodiment of FIG. 16, media arbitration logic 820 is coupled to the network across a receive line 821 and a transmit line 822. Received data is supplied through the media arbitration logic 820 across line 823 to address filter logic 824. The address filter logic 824 has associated with it a first assigned MAC address 826 for the main CPU module and a second assigned MAC address 827 for the multi-media module. Data for the multi-media module is supplied on line 860 and data destined for the main CPU module is supplied on line 861 to the respective recipients.

Data transmitted from the main CPU is supplied on line 862 to the media arbitration logic 820. Similarly, data transmitted from the multi-media module is supplied on line 831 to the media arbitration logic 820. Transmit priority logic 830 is provided with the arbitration circuit, which favors transmissions by the multi-media module, as explained more fully below.

In the design shown in FIG. 16, there is a single Address Filter Logic section which, based on the destination MAC address in a received packet, passes data on one, both, or neither data path depending on the MAC address. The controller has two control registers, FilterControl 828, 829, one for each data path. Each of the two control registers have the following values defined:

0	pass no packets
1	pass only unicast packets in which the destination address matches the unique MAC address assigned to the data path
2	pass only multicasts
3	pass unicasts which have correct destination MAC address and all multicasts
4	pass all packets (promiscuous mode)

The computer would normally be initialized by setting FilterControl 828 for the Main CPU Module =3 and FilterControl 829 for the Multi-Media Module =1.

The alternate implementation shown in FIG. 17 has two duplicate single unique MAC address filter logic modules 844 and 845, one for the Main CPU module and one for the Multi-media module, respectively. This distributed address filter implementation is simpler to design and test but requires more gates to implement in the ASIC.

Thus, as can be seen in FIG. 17, the alternative version includes a media arbitration logic 840 which has a receive input on line 841 and a transmit output on line 842 coupled to the network. A first receive output is supplied on line 843 to address filter logic 844 for the main CPU. The output of

the address filter logic is supplied on line 845 to the main CPU. The address filter logic 844 is coupled to a first register 846 which stores a first unique MAC address for the main CPU module. In addition, it is coupled to a register 847 which stores the filter control parameters as described above.

A second output of the media arbitration logic 840 is supplied on line 848 to address filter logic 849 for the multi-media module. Address filter logic 849 is coupled to a register 850 which stores a second unique MAC address for the multi-media module. In addition, the address filter logic 849 is coupled to a register 851 which stores the filter control parameter for this second path. The output of the address filter logic 849 is supplied on line 853 to the multi-media module.

Transmitted data from the CPU module is supplied on line 854 to the media arbitration logic 840. Transmitted data from the multi-media module is supplied on line 855 to the media arbitration logic 840.

There are two corresponding versions of the Media Arbitration Logic which differ only in the number of receive lines to the address filter or filters. In addition to the receive on transmit and media arbitration logic described elsewhere in this application, both versions have a transmit priority circuit, 830 (in both FIGS. 16 and 17), allowing a multi-media packet to interrupt a packet from the main CPU in order to lower latency on the transmit side. This is done by checking to see the number of bytes left to be transmitted, and if greater than the number of bytes in register TransmitPriorityLength, the packet in transmission is stopped, collision is generated for the required time, and then the multi-media packet is sent. This logic is controlled with a register, TransmitPriority, which has values defined as follows:

- 0 equal priority on both data paths (i.e., no interruption of transmission ever occurs based on priority)
- 1 Main CPU module has priority over Multi-Media module
- 2 Multi-media module has priority over Main CPU module

Thus, a unique multi-media system is provided which allows sophisticated communication protocol handling in a main CPU, but significantly minimizes the latency involved in such processing for audio and visual data directed to or from a specific multi-media module. This system further demonstrates the flexibility of the multiple MAC address network interface controller design using virtual data link layer interfaces to independent processing modules with a single connection to the network according to the present invention.

VI. Conclusion

In sum, the present invention provides a data link layer device having multiple network addresses, which is capable of address blocking and passing functions based on this set of multiple network addresses, rather than a single assigned address. The device presents virtual data link layers to upper level-software responsive to the respective assigned MAC addresses in the data link layer device. This allows upper layer software to take advantage of the multiple MAC addresses, by simplifying the processing. Furthermore, there is much reduced frame traffic in the main processor of the computer because of the enhanced filtering techniques available to the multiple MAC data link layer devices.

Accordingly, a new network interface controller has been provided taking advantage of the multiple MAC address invention, and a high performance multi-function computer system utilizing the virtual data link layers, and integrated end system/integrated system computer have been provided which greatly advance over the prior art.

The foregoing description of preferred embodiments of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously, many modifications and variations will be apparent to practitioners skilled in this art. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention for various embodiments and with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalents.

TABLE 1

```

FOR n = A,B DO;
  IF AddressMatch = 3 THEN DO;
    IF MatchSD,n = 2 THEN DO;
      {pass packet on data path n};
    END;
    ELSEIF MatchSD,n = 3 THEN DO
      {block packet on data path n};
    END;
  ELSE DO;
    IF fMulticast THEN;
      IF fUseDstM,n THEN;
        IF MatchSD,src,n = 0 OR MatchSD,src,n = 1
          THEN {pass packet on data path n};
          ELSE {block packet on data path n};
        ELSE;
          IF MatchSD,dest,n = 0 OR MatchSD,dest,n = 1
            THEN {pass packet on data path n};
            ELSE {block packet on data path n}
          ELSE
            IF fUseDstU,n THEN;
              IF MatchSD,src,n = 0 OR MatchSD,src,n = 2
                THEN {pass packet on data path n};
                ELSE {block packet on data path n};
              ELSE;
                IF MatchSD,dest,n = 0 OR MatchSD,dest,n = 2
                  THEN {pass packet on data path n};
                  ELSE {block packet on data path n};
                END;
              END;
            ELSEIF AddressMatch = 1 THEN DO;
              IF (MatchD,n = 0) OR
                (fMulticast AND (MatchD,n = 1)) OR
                (NOT(fMulticast) AND (MatchD,n = 2))
                THEN {pass packet on data path n};
                ELSE {block packet on data path n};
              END;
            ELSEIF Address Match = 2 THEN DO;
              IF (MatchD,n = 0) OR
                (fMulticast AND (MatchD,n = 1)) OR
                (NOT(fMulticast) AND (MatchD,n = 2))
                THEN {pass packet on data path n};
                ELSE {block packet on data path n};
              END;
            ELSEIF AddressMatch = 0 THEN DO;
              IF (Matchxx,n = 0) OR
                (fMulticast AND (Matchxx,n = 1)) OR
                (NOT(fMulticast) AND (Matchxx,n = 2))
                THEN {pass packet on data path n};
                ELSE {block packet on data path n};
              END;
            END;
          END;
        END;
      END;
    END;
  END;

```

TABLE 2

Constants	
Index _{src}	= 0
Index _{dst}	= 509
Index _{defsrc}	= 510
Index _{defdst}	= 511

TABLE 3

AddressList[0 .. 509]	Array for 48 bit MAC addresses
Count _{src} [0 .. 511]	24 bit counter for each MAC address in AddressList[], non overflowing, incremented every time a source address matches. Entry Count _{src} [510] is also incremented for every packet address match, and Entry Count _{src} [511] is incremented whenever no match is found.
Count _{dst} [0 .. 511]	24 bit counter for each MAC address in AddressList[], non overflowing, incremented every time a destination address matches. Entry Count _{dst} [501] is also incremented for every packet address match, and Entry Count _{dst} [511] is incremented whenever no match is found.
MatchArray[0..511,0..511] _A	512X512X2 bit array for virtual DLL A, with values interpreted as defined below for "Match..." registers.
MatchArray[0..511,0..511] _B	512X512X2 bit array for virtual DLL B, with values interpreted as defined below for "Match..." registers.
fUseDst _{U,A}	Flag for determining which default values to use for a Unicast when both source and destination addresses are matched in AddressList[], but the MatchArray[i,j] entry is "empty". See also the descriptions for Match _{SD,Src,A} and Match _{SD,Src,B} .
fUseDst _{U,B}	Same as fUseDst _{U,A} but for virtual DLL B.
fUseDst _{M,A}	Same as fUseDst _{U,A} but for multicasts.
fUseDst _{M,B}	Same as fUseDst _{M,A} but for virtual DLL B.
Match _{src,A}	Control register to handle case where neither source nor destination address are found in AddressList[]. Values are defined as: 0(00) Pass packet 1(01) Block packet if Unicast and Pass if Multicast 2(10) Pass packet if Unicast and Block if Multicast 3(11) Block packet
Match _{src,B}	Same as Match _{src,A} but for virtual DLL B.

MAC _A	External unique MAC address for Virtual DLL A
MAC _B	External unique MAC address for Virtual DLL B
fMultiCast	Values: 0 Unicast 1 Multicast
Index _{src}	Value from 0 to 511 denoting the index in AddressList[] of the address which matches the source address in the packet. A value of 510 is never returned, and a value of 511 means that no match occurred.
Index _{dst}	Same as Index _{src} but for destination address
AddressMatch	2 bit register loaded after packet processing, with values defined as: 0(00) No match for either address found in AddressList[]

-continued

5	Match _{SD,A}	1(01) source address matched only 2(10) destination address matched only 3(11) both address matched Two bit register loaded from MatchArray[Index _{src} ,Index _{dst}] _A . Values returned are defined with values: 0(00) Empty (i.e., use defaults) 1(01) Undefined 2(10) Pass packet 3(11) Block packet
10	Match _{SD,B} Match _{SD,Src,A}	Same as Match _{SD,A} , but for virtual DLL B. Two bit register loaded from MatchArray[Index _{src} ,Index _{dst}] _A . Used when AddressMatch = 3 (matched pair found), Match _{SD,A} = 0 ("empty"), and fUseDst _{U,A} = 0 (if Unicast) or fUseDst _{M,A} = 0 (if Multicast) (Use Source Address Default). Values returned are defined as: 0(00) Pass packet 1(01) Block if Unicast and Pass if Multicast 2(10) Pass packet if Unicast and Block if Multicast 3(11) Block packet
15	Match _{SD,Src,B}	Two bit register loaded from MatchArray[Index _{src} ,Index _{dst}] _B . Defined the same as Match _{SD,Src,A} but with fUseDst _{U,A} = 1 (if Unicast) or fUseDst _{M,A} = 1 (if Multicast).
20	Match _{SD,Dst,A}	Two bit register loaded from MatchArray[Index _{src} ,Index _{dst}] _B . Defined the same as Match _{SD,Src,A} but for virtual DLL B.
25	Match _{SD,Dst,B}	Two bit register loaded from MatchArray[Index _{src} ,Index _{dst}] _B . Defined the same as Match _{SD,Src,A} but for virtual DLL B.
30	Match _{src,A} Match _{src,B}	Two bit register loaded from MatchArray[Index _{src} ,Index _{dst}] _A . Used when source address is matched in AddressList[] but not destination address. Values are defined the same as for Match _{SD,Src,A} .
35	Match _{src,B} Match _{dst,A} Match _{dst,B}	Two bit register loaded from MatchArray[Index _{src} ,Index _{dst}] _B . Same as Match _{src,A} but for virtual DLL B. Two bit register loaded from MatchArray[Index _{src} ,Index _{dst}] _A . Same as for Match _{src,A} but for the case where destination address matches but not source address. Two bit register loaded from MatchArray[Index _{src} ,Index _{dst}] _B . Same as Match _{dst,A} but for virtual DLL B.
40	Match _{dst,A}	Two bit register loaded from MatchArray[Index _{src} ,Index _{dst}] _B . Same as Match _{dst,A} but for virtual DLL B.

What is claimed is:

1. A computer system to be connected with a network transmission medium, the computer system comprising:
 - a network interface coupled to the network transmission medium, including at least a first processor interface having a first assigned network address and a second processor interface having a second assigned network address;
 - first processing resources coupled to the first processor interface which receive and transmit frames through the network interface; and
 - second processing resources coupled to the second processor interface which receive and transmit frames through the network interface;
 wherein the network interface includes:
 - a connector to transport data to and from the network transmission medium; and
 - a medium access control device, coupled to the connector, to receive and transmit frames of data through the connector, the medium access control device including a first data channel to the first processor interface and a second data channel to the second processor interface, memory to store the first and second assigned network addresses, and address filtering logic, coupled to the first and second data

channels and to the memory, which passes and blocks frames received through the connector for the first and second data channels in response to the assigned network addresses.

2. The system of claim 1, wherein the first and second processor interfaces include respective buffering and queuing structures for communication of data between the network transmission medium and the first and second processing resources.

3. The system of claim 1, wherein the memory further stores a plurality of additional network addresses in addition to the first and second assigned network addresses, and the address filtering logic includes circuits responsive to the plurality of additional network addresses.

4. The system of claim 3, wherein frames include source and destination addresses, and the circuits responsive to the plurality of additional network addresses include logic for blocking a particular frame on at least one of the first and second data channels when the plurality of additional network addresses includes the source and destination addresses of the particular frame.

5. The system of claim 1, wherein the first and second processor interfaces comprise respective virtual data link layer interfaces.

6. The system of claim 1, further including at least one additional network interface coupled to the first processing resources, and wherein the first processing resources include network intermediate system functions, and the second processing resources include network management functions.

7. The system of claim 1, further including at least one additional network interface coupled to the first processing resources, and wherein the first processing resources include network intermediate system functions, and the second processing resources include network end system functions.

8. A computer system to be connected with a plurality of network transmission media, the computer system comprising:

a shared network interface including a connector coupled to a first network transmission medium, and including at least a first processor interface having a first assigned network address and a second processor interface having a second assigned network address;

a first processor coupled to the first processor interface, the first processor including network end system resources; and

a second processor coupled to the second processor interface, the second processor including a second network interface coupled to a second network transmission medium and intermediate system resources for transporting frames between the shared and the second network interfaces for transmission across the first and second network transmission media;

wherein the shared network interface includes:

a medium access control device, coupled to the connector, to receive and transmit frames of data through the connector, the medium access control device including a first data channel to the first processor interface and a second data channel to the second processor interface, memory to store the first and second assigned network addresses, and address filtering logic, coupled to the first and second data channels and to the memory, which passes and blocks frames received through the connector for the first and second data channels in response to the assigned network addresses.

9. The apparatus of claim 8, wherein the intermediate system resources include routing adaptor functions.

10. The apparatus of claim 8, wherein the intermediate system resources include bridge functions.

11. The apparatus of claim 8, wherein the intermediate system resources include router functions.

12. The apparatus of claim 8, wherein the intermediate system resources include bridge and router functions.

13. The system of claim 8, wherein the first and second processor interfaces include respective buffering and queuing structures for communication of data between the network transmission medium and the first and second processing resources.

14. The system of claim 8, wherein the shared network interface includes address filtering logic which passes or blocks received transmissions for the first processor interface in response to the first assigned network address and which passes or blocks received transmissions for the second processor interface in response to the second assigned network address.

15. The system of claim 8, wherein the memory further stores a plurality of additional network addresses in addition to the first and second assigned network addresses, and the address filtering logic includes circuits responsive to the plurality of additional network addresses.

16. The system of claim 15, wherein frames include source and destination addresses, and the circuits responsive to the plurality of additional network addresses include logic for blocking a particular frame on at least one of the first and second data channels when the plurality of additional network addresses includes the source and destination addresses of the particular frame.

17. A computer system to be connected with a network transmission medium, the computer system comprising:

a shared network interface including a connector coupled to the network transmission medium, and including at least a first processor interface having a first assigned network address and a second processor interface having a second assigned network address;

a data processing unit coupled to the first processor interface which receives and transmits frames through the shared network interface for end system processes; and

a multi-media unit coupled to the second processor interface which receives and transmits frames of multi-media data through the shared network interface for multi-media system processes;

wherein the shared network interface includes:

a medium access control device, coupled to the connector, to receive and transmit frames of data through the connector, the medium access control device including a first data channel to the first processor interface and a second data channel to the second processor interface, memory to store the first and second assigned network addresses, and address filtering logic, coupled to the first and second data channels and to the memory, which passes and blocks frames received through the connector for the first and second data channels in response to the assigned network addresses.

18. The system of claim 17, wherein the shared network interface includes transmit priority logic favoring the multi-media unit.

19. The system of claim 17, wherein the first and second processor interfaces include respective buffering and queuing structures for communication of data between the network transmission medium and the data processing unit and the multi-media unit.

20. The system of claim 17, wherein the shared network interface includes address filtering logic which passes or

35

received transmissions for the first processor interface in response to the first assigned network address and passes or blocks received transmissions for the second processor interface in response to the second assigned network address.

A computer system to be connected with a network transmission medium, the computer system comprising:

shared network interface including a connector coupled to the network transmission medium, and including at least a first processor interface having a first assigned network address and a second processor interface having a second assigned network address;

data processing unit coupled to the first processor interface which receives and transmits frames through the shared network interface for end system processes;

and routing adaptor unit, having a first port coupled to the second processor interface which receives and transmits frames of data through the shared network interface, and a second port adapted for a communication link to a remote system, and including resources to forward frames received through the first port having the second assigned network address to the second port and to forward frames received through the second port and not having the second assigned network address to the first port;

wherein the shared network interface includes:

36

a medium access control device, coupled to the connector, to receive and transmit frames of data through the connector, the medium access control device including a first data channel to the first processor interface and a second data channel to the second processor interface, memory to store the first and second assigned network addresses, and address filtering logic, coupled to the first and second data channels and to the memory, which passes and blocks frames received through the connector for the first and second data channels in response to the assigned network addresses.

22. The system of claim 21, wherein the first and second processor interfaces include respective buffering and queuing structures for communication of data between the network transmission medium and the data processing unit and between the network transmission medium and the routing adaptor unit.

23. The system of claim 21, wherein the shared network interface includes address filtering logic which passes or blocks received transmissions for the first processor interface in response to the first assigned network address and which passes or blocks received transmissions for the second processor interface in response to the second assigned network address.

* * * * *



US005802054A

United States Patent [19]
Bellenger

[11] **Patent Number:** 5,802,054
[45] **Date of Patent:** Sep. 1, 1998

[54] **ATOMIC NETWORK SWITCH WITH INTEGRATED CIRCUIT SWITCH NODES**

[75] **Inventor:** Donald M. Bellenger, Los Altos Hills, Calif.

[73] **Assignee:** 3Com Corporation, Santa Clara, Calif.

[21] **Appl. No.:** 698,745

[22] **Filed:** Aug. 16, 1996

[51] **Int. Cl. 6** H04L 12/66

[52] **U.S. Cl.** 370/401

[58] **Field of Search** 370/351, 400, 370/401, 402, 407, 408, 422

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,947,390	8/1990	Sheehy	370/401
5,047,917	9/1991	Athas et al.	364/200
5,166,931	11/1992	Riddle	370/401
5,321,695	6/1994	Faulk, Jr.	370/401
5,390,173	2/1995	Spinney et al.	370/401
5,477,547	12/1995	Sugiyama	370/401
5,610,905	3/1997	Murthy et al.	370/401
5,657,314	8/1997	McClure et al.	370/401

OTHER PUBLICATIONS

ATOMIC: A Low-Cost, Very High-Speed, Local Communication Architecture, Danny Cohen, Gregory Finn, Robert Felderman, Annette DeSchon, USC/Information Sciences Institute, 1993 International Conference on Parallel Processing.

The Use of Message-Based Multicomputer Components to Construct Gigabit Networks, by D. Cohen, G. Finn, R. Felderman and A. DeSchon, University of Southern California/Information Sciences Institute.

ATOMIC: A High-Speed Local Communication Architecture, by R. Felderman, A. DeSchon, D. Cohen, G. Finn, USC/Information Sciences Institute, *Journal of High Speed Networks* 1 (1994) pp. 1-28, IOS Press.

ATOMIC: A Local Communication Network Created Through Repeated Application of Multicomputing Components, by D. Cohen, G. Finn, R. Felderman, A. DeSchon.

An Integration of Network Communication and Workstation Architecture, by Gregory G. Finn, USC/Information Sciences Institute, Published Oct. 1991, ACM Computer Communication Review.

(List continued on next page.)

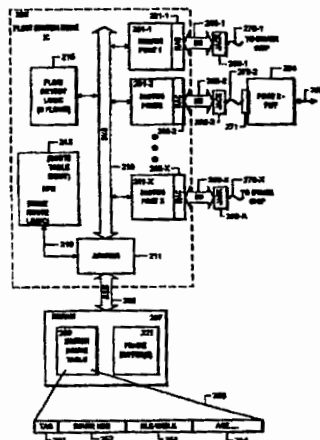
Primary Examiner—Ajit Patel

Attorney, Agent, or Firm—Mark A. Haynes; Kent R. Richardson; Wilson, Sonsini, Goodrich & Rosati

[57] **ABSTRACT**

An atomic type switch mesh is combined with standard local area network links, such as high speed Ethernet, and a bridge-like protocol to provide a high performance scalable network switch. The network switch comprises a plurality of switch nodes, a first set of communication links which are coupled between switch nodes internal to the network switch, and a second set of communication links which comprise network links from switch nodes on the border of the network switch to systems external to the network switch. The respective switch nodes include a set of ports (having more than two members) which are connected to respective communication links in either the first or second set of communication links. Each port in the set comprises a medium access control (MAC) logic unit for a connectionless network protocol, preferably high speed Ethernet. The switch nodes also include a route table memory which has a set of accessible memory locations that store switch route data specifying routes through the plurality of switch nodes within the boundaries of the network switch. Flow detect logic is coupled with the set of ports on the switch node, which monitors frames received by the set of ports and generates an identifying tag for use in accessing the route table memory. Finally, the switch node includes node route logic which is coupled with the flow detect logic, the route table memory and the set of ports. The node route logic monitors frames received by the set of ports to route a received frame for transmission out a port in the set of ports.

56 Claims, 6 Drawing Sheets



OTHER PUBLICATIONS

ATOMIC: A Low-Cost, Very-High-Speed LAN, by D. Cohen, G. Finn, R. Felderman, A. DeSchoen.
The Design of the Caltech Mosaic C Multicomputer, C. Seitz, N. Boden, J. Seizovic, and W. Su, Computer Science 256-80, California Institute of Technology.
802.3z Higher Speed Task Force Objectives (Gigabit Ethernet), Apr., 1996.

Netstation Architecture Multi-Gigabit Workstation Network Fabric, G. Finn, P. Mockapetris, USC/Information Sciences Institute.

A Zero-Pass End-to-End Checksum Mechanism for IPv6¹, G. Finn, S. Hotz, C. Rogers, USC/Information Sciences Institute, Dec., 1995.

Network Backplane, G. Finn, USC/Information Sciences Institute, Apr., 1994.

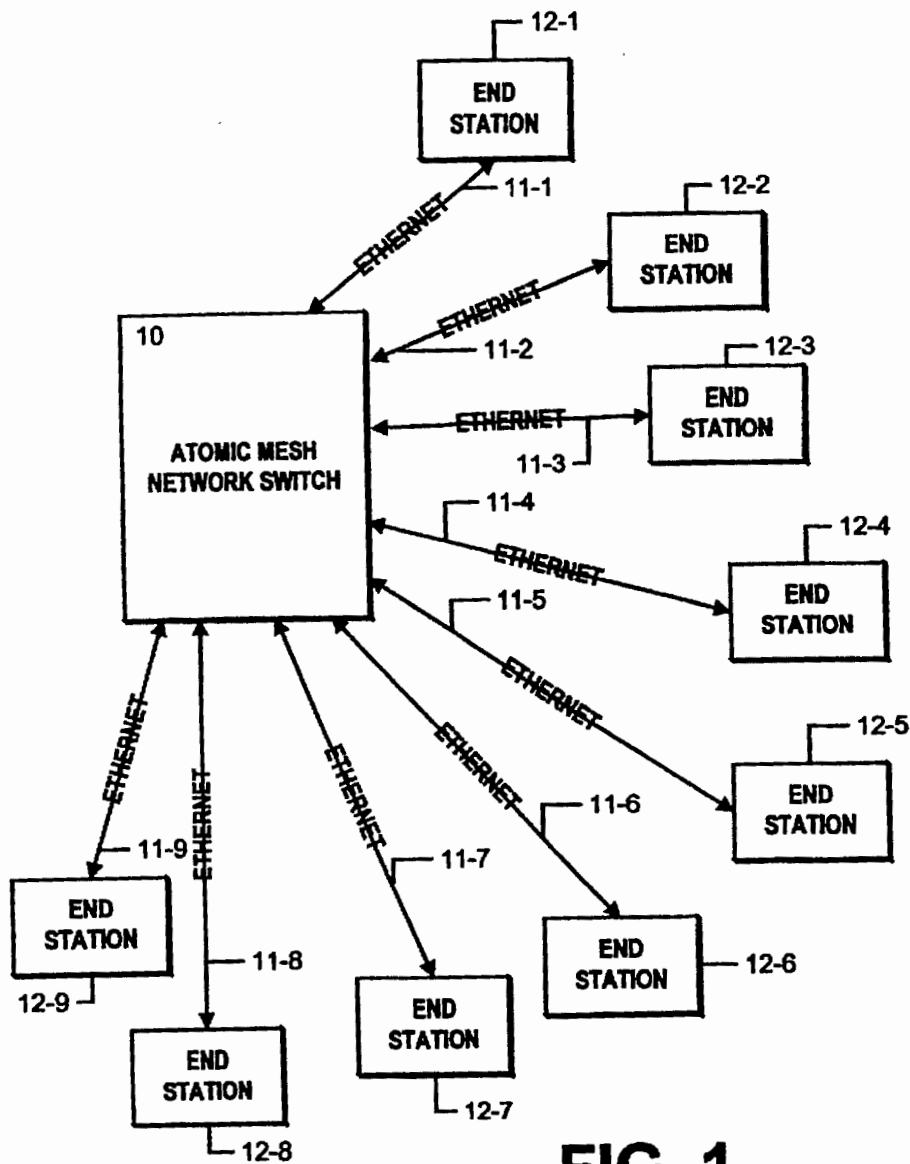


FIG. 1

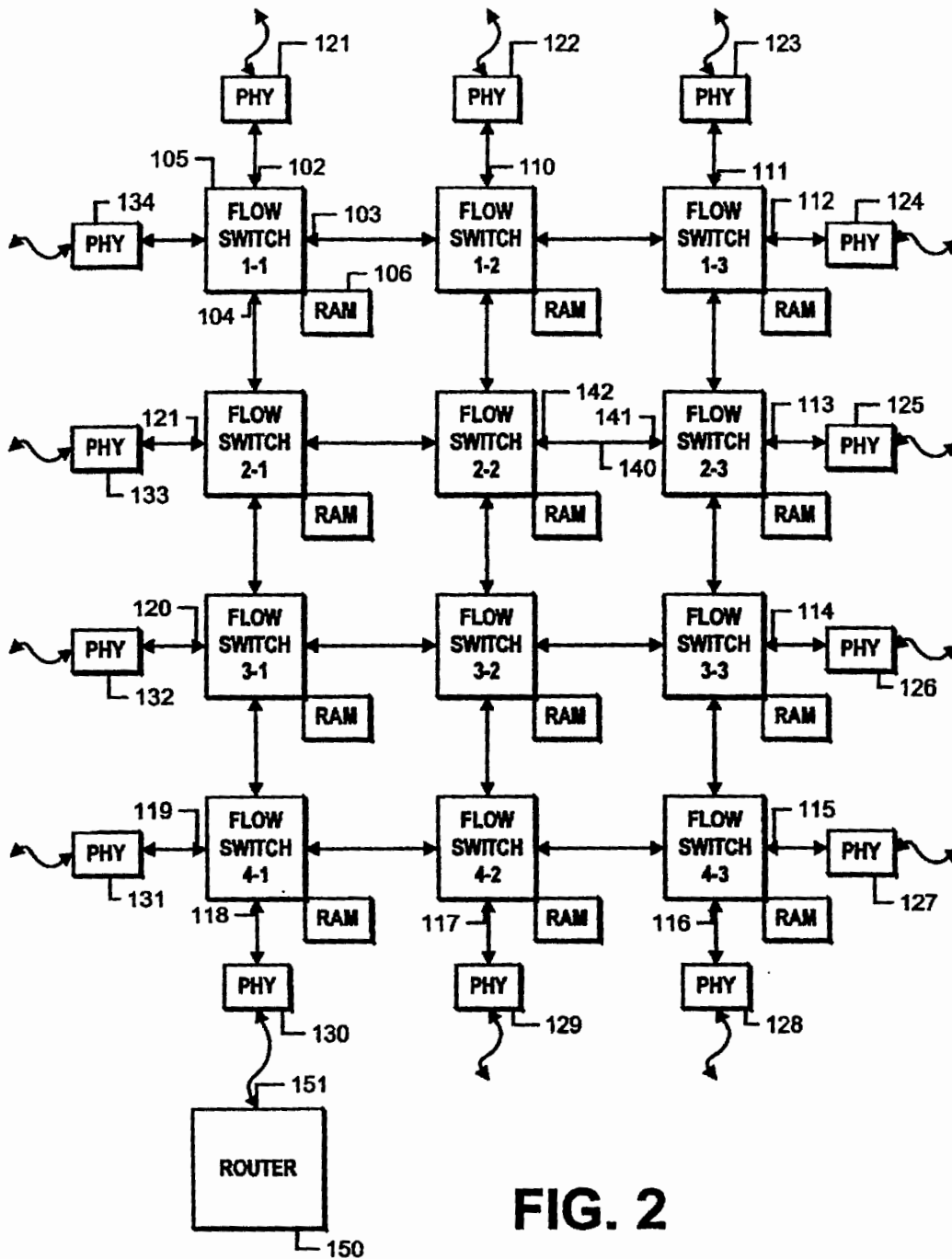


FIG. 2

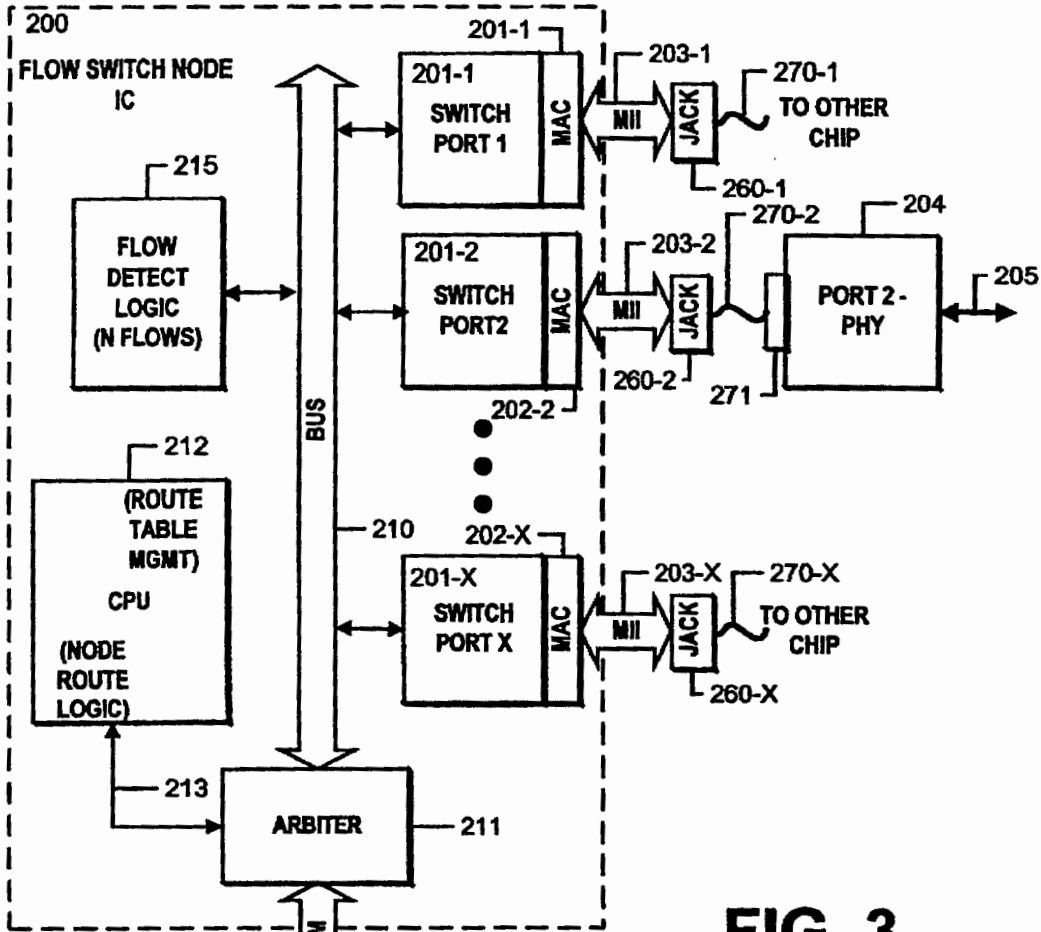


FIG. 3

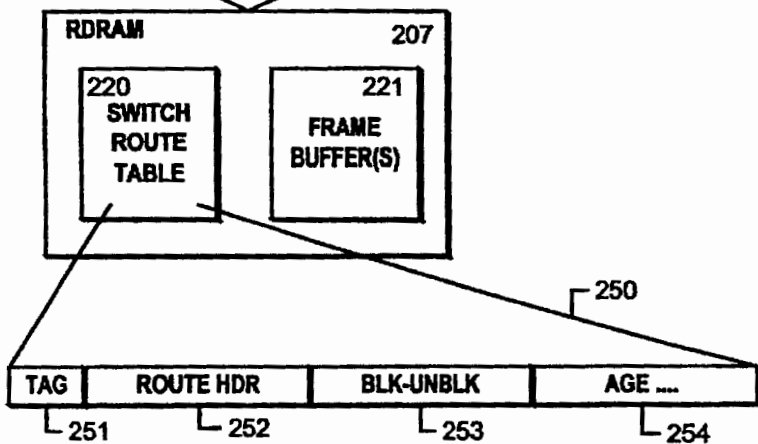
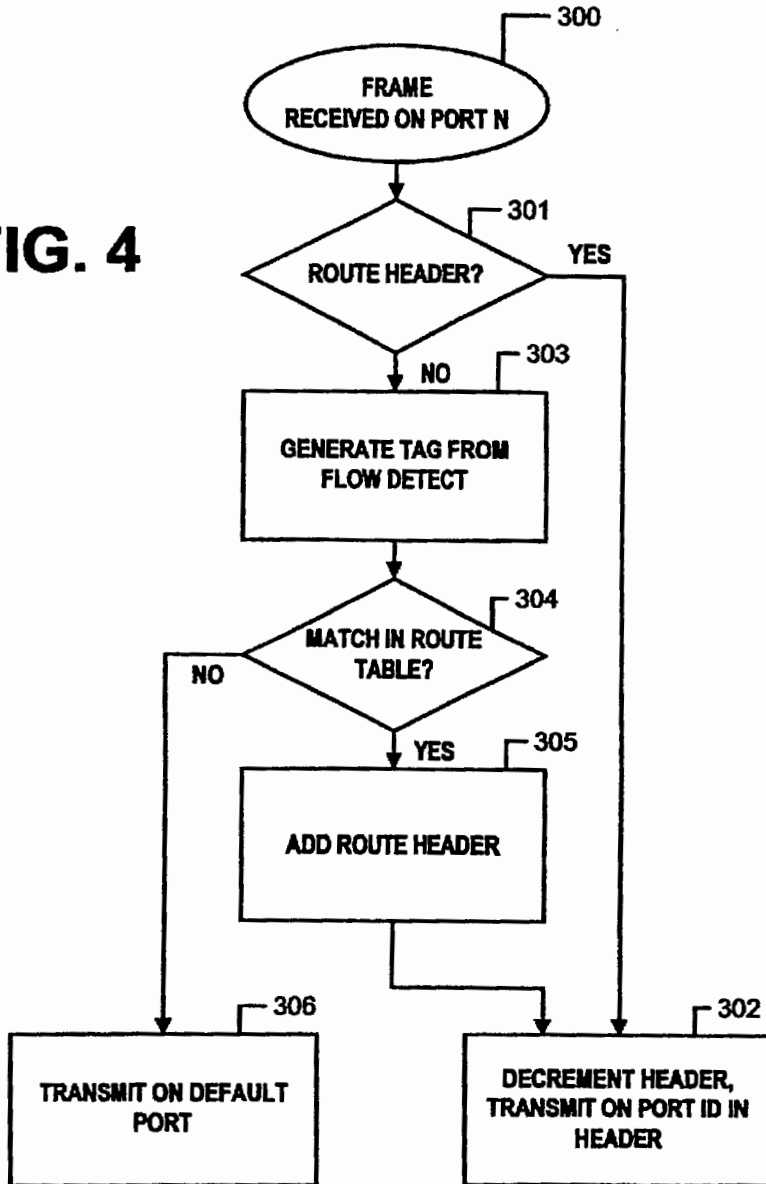
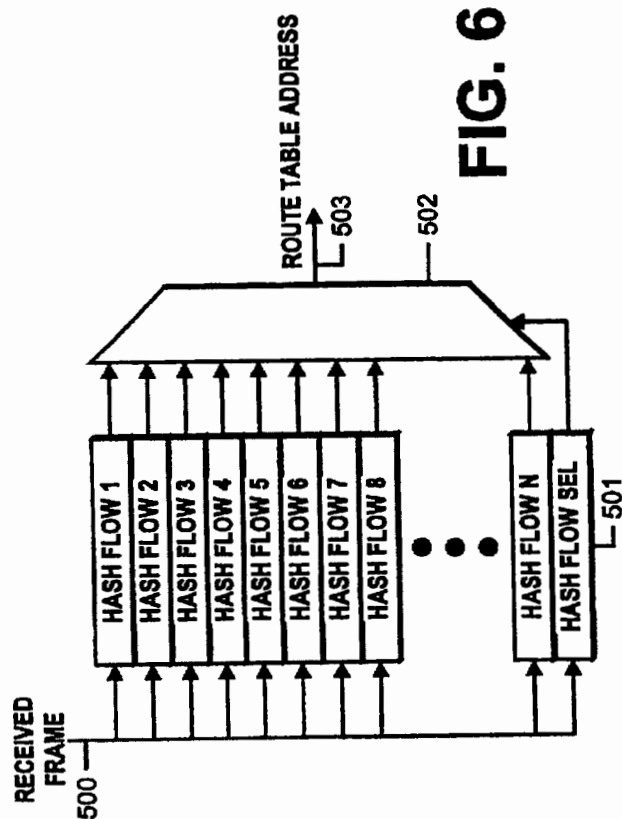
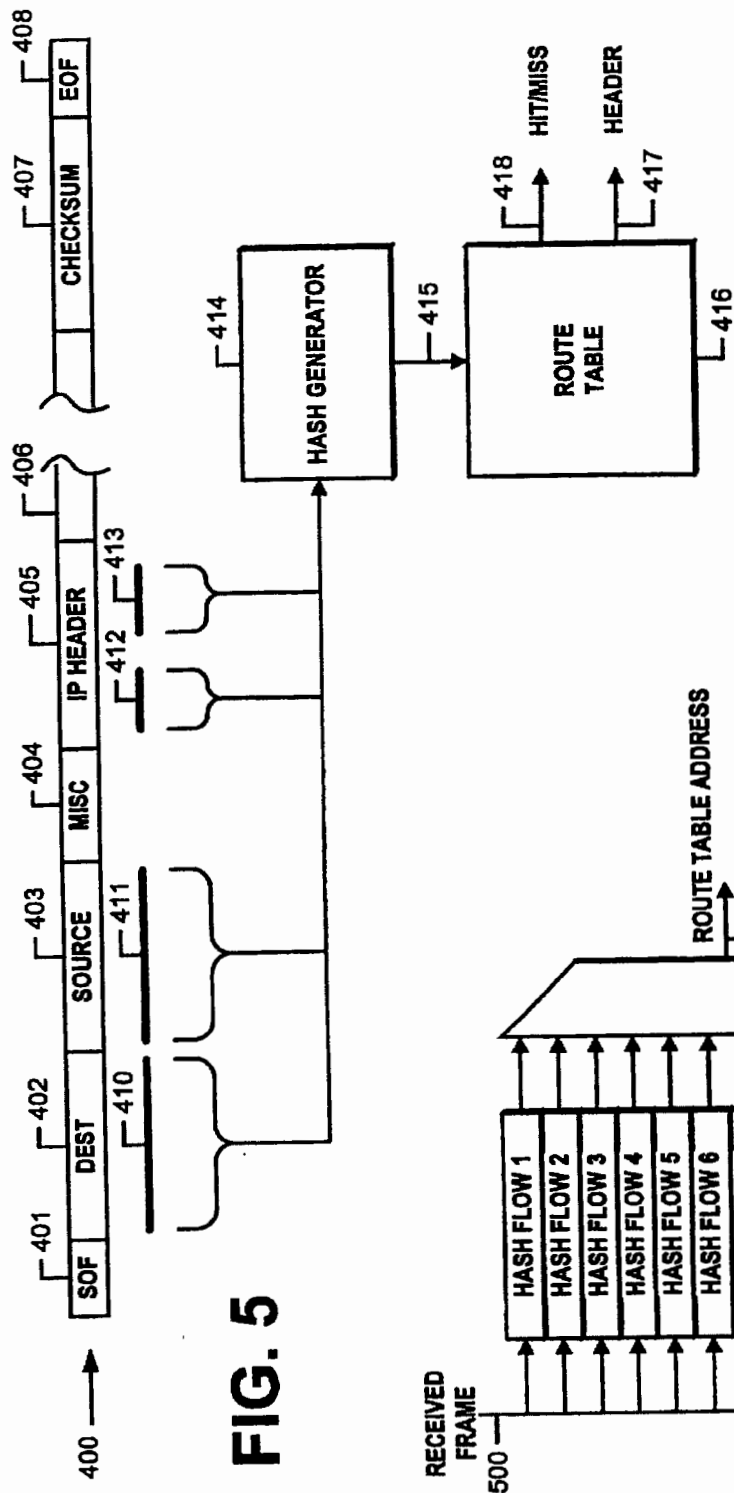


FIG. 4





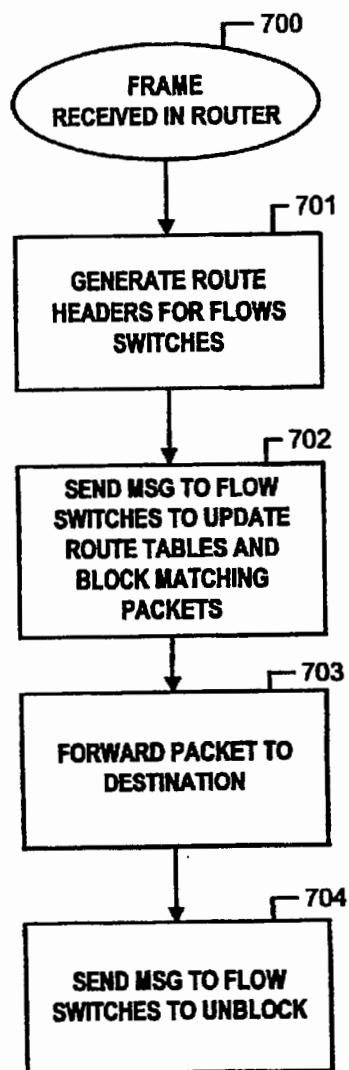


FIG. 7

**ATOMIC NETWORK SWITCH WITH
INTEGRATED CIRCUIT SWITCH NODES**
BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to the field of network intermediate devices, and more particularly to high-performance switches for routing data in computer networks.

2. Description of Related Art

Network intermediate systems for interconnecting networks include various classes of devices, including bridges, routers and switches. Systems for the interconnection of multiple networks encounter a variety of problems, including the diversity of network protocols executed in the networks to be interconnected, the high bandwidth required in order to handle the convergence of data from the interconnected networks at one place, and the complexity of the systems being designed to handle these problems. As the bandwidth of local area network protocols increases, with the development of so-called asynchronous transfer mode ("ATM"), 100 megabit per second Ethernet standards, and proposals for gigabit per second Ethernet standards, the problems encountered at network intermediate systems are being multiplied.

One technique which has been the subject of significant research for increasing the throughput of networks is known as the so-called atomic LAN. The atomic LAN is described for example in Cohen, et al., "ATOMIC: A Low-Cost, Very High-Speed, Local Communication Architecture", 1993 International Conference on Parallel Processing. There is a significant amount of published information about the atomic LAN technology. Felderman, et al. "ATOMIC: A High-Speed Local Communication Architecture", *Journal of High Speed Networks*, Vol. 1, 1994, pp. 1-28; Cohen, et al., "ATOMIC: A Local Communication Network Created Through Repeated Application of Multicomputing Components", DARPA Contract No. DABT63-91-C-001, Oct. 1, 1992; Cohen et al., "The Use of Message-Based Multicomputer Components to Construct Gigabyte Networks"; DARPA Contract No. DABT63-91-C-001, published Jun. 1, 1992; Finn, "An Integration of Network Communications with Workstation Architecture", *ACM, A Computer Communication Review*, October 1991; Cohen et al., "ATOMIC: Low-cost, Very-High-Speed LAN", DARPA Contract No. DABT63-91-C-001 (publication date unknown, downloaded from Internet on or about May 10, 1996).

The atomic LAN is built by repeating simple four port switch integrated circuits in the end stations, based on the well known Mosaic architecture created at the California Institute of Technology. These integrated circuits at the end stations are interconnected in a mesh arrangement to produce a large pool of bandwidth that can cross many ports. The links that interconnect the switches run at 500 megabits per second. Frames are routed among the end stations of the network using a differential source route code adapted for the mesh. One or more end stations in the mesh act "address consultants" to map the mesh and calculate source route codes. All of the links are self timed, and depend on acknowledged signal protocols to coordinate flow across the links to prevent congestion. The routing method for navigating through the mesh, known as "worm hole" routing is designed to reduce the buffering requirements at each node.

The atomic LAN has not achieved commercial application to a significant degree, with an exception possibly in

connection with a supercomputer known as Paragon from Intel Corporation of Santa Clara, Calif. Basically it has been only a research demonstration project. Critical limitations of the design include the fact that it is based on grossly non-standard elements which make commercial use impractical. For example, there is no way to interface the switch chips taught according to the atomic LAN project with standard workstations. Each workstation needs a special interface chip to become part of the mesh in order to participate in the LAN. Nonetheless, the ATOMIC LAN project has demonstrated a high throughput and readily extendable architecture for communicating data.

Typical switches and routers in the prior art are based on an architecture requiring a "backplane" having electrical characteristics that are superior to any of the incoming links to be switched. For example, 3Com Corporation of Santa Clara, Calif., produces a product known as NetBuilder2, having a core bus backplane defined which runs at 800 megabits per second. This backplane moves traffic among various local area network external ports.

There are several problems with the backplane approach typical of prior art intermediate systems. First, the backplane must be defined fast enough to handle the largest load that might occur in the intermediate system. Furthermore, the customer must pay for worst case backplane design, regardless of the customer's actual need for the worst case system. Second, the backplane itself is just another communication link. This communication link must be completely supported as a backplane for the network intermediate system, involving intricate and expensive design. The lower volumes for specialized backplane link further increases the cost of network intermediate systems based on the backplane architecture.

In light of the ever increasing complexity and bandwidth requirements of network intermediate systems in commercial settings, it is desirable to apply the atomic LAN principles in practical, easy to implement, and extendable network intermediate systems.

SUMMARY OF THE INVENTION

According to the present invention, the fine scalability of an atomic type LAN mesh, is combined with standard local area network links, such as high speed Ethernet, and a standard routing protocol to provide a high performance scalable network switch. The need for the special purpose backplane bus is removed according to this architecture, while providing scalability, high performance, and simplicity of design.

Accordingly, the present invention can be characterized as a network switch that comprises a plurality of switch nodes arranged in a mesh, a first set of internal communication links which are coupled between switch nodes internal to the network switch, and a second set of external communication links which comprise network links from switch nodes on the border of the network switch to systems external to the network switch. The respective switch nodes include a set of ports (having more than two members) which are connected to respective communication links in one of the first or second sets of communication links. The ports in the set of ports include respective medium access control (MAC) units for transmission and reception of data frames according to a network protocol, preferably a connectionless protocol like high speed Ethernet, and are connectable to a port on another network switch node inside the mesh across an internal communication link, or to a network communication medium outside the mesh which constitutes, or is coupled with, an external communication link.

The switch nodes also include resources to execute a routing process for frames inside the mesh. These resources include a route table memory which has a set of accessible memory locations that store switch route data specifying routes through the plurality of switch nodes inside the mesh of the network switch for specific flows of data frames, or for data frames having specific destination addresses. Flow detect logic is coupled with the set of ports on the switch node, which monitors frames received by the set of ports and generates an identifying tag for use in accessing the route table memory. Example tags consist of a destination address at one of the data link layer or the network layer, a portion of the destination address, or hash values based on one or more fields in control segments of the frame. The tags preferably act as flow signatures to associate a frame with a sequence of frames traversing the switch. For example, when a large file is transferred, a sequence of frames is generated which constitutes a flow of data to a single destination, and frames in the sequence have a single identifying tag. Finally, the switch node includes node route logic which is coupled with the flow detect logic, the route table memory and the set of ports. The node route logic monitors frames received by the set of ports to route a received frame for transmission out a port in the set of ports.

The node route logic determines whether the received frame includes a switch route field that indicates a port in the set of ports to which the frame should be directed for transmission. If the received frame includes a switch route field, that field is updated according to a source route type protocol, and the frame is forwarded with the updated switch route field out the indicated port. If the received frame does not include a switch route field, such as would normally be the case for a frame entering the network switch at a switch node on the border of the network switch, then the identifying tag generated by the flow detect logic is used to access the route table memory. Switch route data is retrieved from the route table memory, if an entry exists for the identifying tag of the current frame. This data is used to generate a switch route field for the frame, and to direct the frame out a port indicated by the data.

The node route logic on the respective switch node also includes logic that forwards a received frame for transmission on a default port in the set of ports, when the route table memory does not include switch route data for the identifying tag. The default port is coupled to a route leading to a processor in the system at which switch route data is generated, such as a multi-protocol network router either internal or external to the network switch. Thus, the node route logic further includes logic to receive switch route data from a remote system for a particular identifying tag. This switch route data is stored in the route table memory in association with the particular identifying tag. When a new entry is made in a switch route table, frames having the particular identifying tag are blocked, with or without buffering, until notification is received that it is clear to forward frames having the particular identifying tag. This blocking technique allows the remote system to which a frame was directed for routing, to forward the frame to its destination, prior to other frames in the same flow sequence being routed to that destination. This preserves the order of transmission of frames in a particular flow. The node route logic begins forwarding frames according to the switch route data stored in the route table memory for a particular tag after it receives notification from the remote system that it is clear to forward frames.

The term frame is used herein, unless stated otherwise, in a generic sense as a unit of data transferred according to a

network protocol, intending to include data units called frames, packets, cells, strings, or other names which may have more specific meaning in other contexts.

In the preferred system, all the ports on the switch node execute a single local area network protocol. Preferably this protocol is an Ethernet protocol like the carrier sense, multiple access with collision detection CSMA/CD protocol of the widely used Ethernet standard and variants of it. More preferably, the protocol is specified for operation at 100 megabits per second or higher, more preferably at the emerging one gigabit per second Ethernet standard protocol. For example, half duplex and full duplex "Gigabit" Ethernet (IEEE802.3z) or 100 Megabit Ethernet (802.3u) are used in preferred embodiments.

Flow control between the nodes is handled according to the standard LAN protocol of the ports, such as the Ethernet protocol. Thus, management of the frame flow through the switch is conducted on a frame by frame basis with the format of the frame inside the switch essentially unaltered from the format entering or exiting the switch, with well understood and easily implemented technology.

According to another aspect of the present invention, the flow detect logic on the respective switch nodes comprises logic which computes a plurality of hash values in response to respective sets of control fields in a received frame. The respective sets of control fields correlate with different network frame formats which might be encountered in the network. Logic is also included which determines a particular network frame format for a received frame, and selects one of the plurality of hash values as the identifying tag in response to the particular network frame format that has been detected. The hash values preferably comprise cyclic redundancy codes which are generated with hardware CRC generators. In this manner, the identifying tag for an incoming frame is generated very quickly, allowing for cut through of frames in a switch node so that a transmission of a frame on an outgoing port can begin before the complete frame has been received at the incoming port.

The present invention can also be characterized as individual switch nodes for use in a network switch in the configuration described above. In another aspect, the network switch node comprises an integrated circuit on which the plurality of ports, the flow control logic, and the flow detect logic are incorporated, and interconnected by an embedded high speed bus. A system including any two or more of such integrated circuits combined together to form a mesh, provide a network switch. According to another aspect of the invention, the ports on the integrated circuits are coupled with standard jack connectors, or other standard connector interfaces, allowing users of switch circuits including a plurality of integrated circuits to connect them together using cables in any desired configuration. Thus, a very flexible switch architecture is provided which can be configured for individual installations very easily.

A high performance network switch is provided according to the present invention based on a switch node made with an integrated circuit having 3 or more LAN ports. A frame is routed amongst the nodes in the switch without moving across any intermediate non-LAN bus (excluding the memory interface in each of the nodes used for the frame buffers). A route decision is made in each node based on a switch route header attached to the LAN frame, or on the Ethernet address contained within the frame, or directed to a default route if no route is stored in the route table and the Ethernet address is unknown. The flow control amongst the nodes in the switch is handled based on standard LAN

control signals. In the preferred system, the standard LAN interface amongst the nodes is 100 megabit per second or higher Ethernet, and more preferably the emerging 1 gigabit per second Ethernet protocol.

Other aspects and advantages of the present invention can be seen upon review of the drawings, the detailed description and the claims which follow.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a simplified diagram of a network including an atomic network switch according to the present invention, interconnecting a plurality of standard Ethernet links.

FIG. 2 is a block diagram of a network switch based on a mesh of switch nodes according to the present invention.

FIG. 3 is a block diagram of a switch node according to the present invention.

FIG. 4 is a flow chart illustrating the process executed by the node route logic in the switch node of FIG. 3.

FIG. 5 is a diagram illustrating the process of generating identifying tags based on cyclic redundancy code hash generators for the flow detect logic of the system of FIG. 3.

FIG. 6 is a simplified block diagram of the flow detect logic for multiple parallel flows for use in the system of FIG. 3.

FIG. 7 is a flow chart illustrating the process executed in a router or other network route processor for frames received from the network switch, which do not have entries in the route tables of the network switch.

DETAILED DESCRIPTION

A detailed description of embodiments of the present invention is provided with reference to FIGS. 1 through 7, where FIG. 1 illustrates the context in which the present invention is utilized. In FIG. 1, an atomic network switch 10 according to the present invention is connected by standard Ethernet links 11-1 through 11-9 to a plurality of end stations 12-1 through 12-9. The number of end stations and Ethernet links shown in FIG. 1 is arbitrary. A larger or smaller number of links could be connected to a single atomic switch 10 according to the present invention, as described in detail below. Furthermore, the connections 11-1 through 11-9 from the atomic switch to the respective end stations are all standard network connections, preferably CSMA/CD protocol links, such as the standard full duplex fast Ethernet (IEEE802.3u) specified for 100 megabits per second each way, or the emerging standard full duplex, 1 gigabit per second Ethernet protocol. In the preferred system, all links 11-1 through 11-9 operate according to the same network protocol. However, alternative systems accommodate multiple network protocols on the external ports of switch 10.

The end stations 12-1 through 12-9 may be personal computers, high performance workstations, multimedia appliances, printers, network intermediate systems coupled to further networks, or other data processing devices as understood in the art.

According to one embodiment of the present invention one of the end stations, such as end station 12-1 includes resources to manage the configuration of the atomic network switch 10, such as initializing route tables, maintaining the route tables, and providing other functions. Thus, end station 12-1 may include resources to act as a multi-protocol router, such as the NetBuilder2 manufactured by 3Com Corporation of Santa Clara, Calif.

FIG. 2 illustrates the internal architecture of the atomic network switch 10 shown in FIG. 1. The atomic network

switch 10 is comprised of a plurality of switch nodes arranged in rows and columns in FIG. 2. The switch nodes are labeled in the drawing by column and row numbers. Thus, the switch node in the upper left hand corner is node 1-1. The switch node at row 1, column 2 is node 1-2, and so on throughout the mesh. In a preferred embodiment, each switch node includes an integrated circuit, such as integrated circuit 105 in node 1-1, coupled to a memory chip, such as chip 106 in node 1-1. Each of the nodes includes four ports. Thus, node 1-1 includes port 101, port 102, port 103, and port 104.

The boundary of the network switch in FIG. 2 comprises the nodes 101 and 102 of node 1-1, port 110 of node 1-2, port 111 of node 1-3, 112 of node 1-3, port 113 of node 2-3, port 114 of node 3-3, port 115 of node 4-3, port 116 of node 4-3, port 117 of node 4-2, port 118 of node 4-1, port 119 of node 4-1, port 120 of node 3-1, and port 121 of node 2-1. Each of the ports 110-121, 101 and 102 on the boundary of the switch is connected to through a physical layer device, 121-134 to respective physical communication media, such as fiberoptic cables, twisted pair cables, wireless links, such as radio frequency or infrared channels, or other media specified according to standard local area network physical layer specifications. The connection between switch nodes, such as the connection 140 between port 141 on node 2-3 and port 142 on node 2-2, consist of medium independent interface connections which are defined for connection between MAC logic on a port, and medium dependent components for a port. However, these medium independent connections are connected from MAC logic to MAC logic directly. Preferably all the links between the ports in the network switch execute the same network protocol as the ports on the boundary of the switch. However, alternative systems support multiple protocol types at the boundary.

Management of the configuration of the network switch is accomplished in a router 150 which is connected across link 151 to the physical layer device 130 on the network switch.

The memory chips, such as chip 106 at node 1-1, in the network switch are used to store route tables, and as frame buffers used in routing of frames amongst the nodes of the switch.

In operation, the network switch receives and transmits standard LAN frames on physical interfaces 121-134. Preferably, the LAN interconnections comprise CSMA/CD LANs, such as 100 Megabit Ethernet (IEEE802.3 u), or 1 gigabit Ethernet. When a standard frame enters the switch at one physical interface, it is directed out of the switch through another physical interface as indicated by the address data carried by the frame itself. The individual nodes in the switch include a switch routing feature. Each individual node selects a port on which to transmit a received frame based upon the contents of the header of the incoming frame.

There are two internal modes for routing frames inside the switch. In the base mode, each node routes frames using a switch route header attached to the beginning of the regular LAN frame. The switch route header in one example consists of a series of bytes, each byte specifying one or more hops of the route. The top two bits in one byte specify a direction, in the next bits specify the distance. As a frame moves through each node, the header is updated until it reaches the target. Before a frame leaves the mesh, all the switch route bytes are stripped, and the frame has the same format as it had when it entered the mesh or, if required, a format adapted to the network protocol of the exit port.

The nodes of the switch, at least nodes on the boundary of the switch, also have a look up mode. When a frame

enters the switch, with no source route header, the Ethernet addresses, or other fields of the control header of the frame are utilized access the route table. In preferred systems, a CRC-like checksum generator is run over the header of the frame, or over selected fields in the header. At the end of the header, the checksum, or the low order bits of the checksum, are used as a hash code to access a route table stored in the memory associated with the node. Other look up techniques could be utilized for accessing the route table in the memory. For example, the destination address of the incoming frame could be used directly as an address in the table.

If there is an entry in the route table corresponding to the header of the frame, then the switch route data from the table is used to create a switch route header. The header is attached to the frame, and the frame is transmitted at the appropriate port. If no entry is found in the route table, then the frame is routed to a default address, such as the address of a multiprotocol router associated with the switch. The multiprotocol router at the default address also performs management functions such as reporting status, initializing the network, broadcast functions, and managing node route tables. Routing the frame to a default address alternatively involves attachment of a switch route header to direct the frame to the default address, or simply forwarding the frame at a default port in the local node, such that the next node in the mesh to receive the frame also looks it up in its own route table to determine whether the frame is recognized. Either way, the frame reaches the default address and is handled appropriately.

Flow control of the frames in the mesh, and at the boundary of the mesh, is based on the network protocol of the links, such as Ethernet. Therefore, in the preferred Ethernet example, if a port is not available in a target node due to a busy link, a collision on the link, or lack of memory space at the target node, the frame will be refused with a jam signal or a busy signal on the link. The sending node buffers the frame, and retries the transmission later, according to the backoff and retry rules of the protocol or other flow control techniques of the protocol.

The standard higher-speed Ethernet protocols include both half duplex and full duplex embodiments. The 100 Megabit per second Ethernet, defined by IEEE802.3u, clause 31 "MAC Control," defines a frame-based flow control scheme for the full duplex embodiment. Flow control slows down the aggregate rate of packets that a particular port is sending. The method used revolves around control frames distinguished by a unique multicast address and a length/type field in the packet. When a MAC port controller detects that it has received a control frame, the opcode in the control frame is sensed, and transmission of packets is controlled based on the opcode. In existing specifications, a single opcode PAUSE is defined. Thus, in response to the PAUSE opcode, transmission of packets is either enabled or disabled depending on the current state in a Xon/Xoff type mechanism. Thus, this full duplex mode does not depend on the shared media, collision detect techniques of the classic CSMA/CD protocols.

All the proposed standards in the Ethernet family basically use the standard 802.3/Ethernet frame format, conforming to the 802.2 logical link control layer interface, and the 802 functional requirement document with the possible exception of Hamming distance. Also, the minimum and maximum frame size as specified by the current 802.3 standard and by the half or full duplex operational modes is different in the higher rate standards. Thus, the half and full duplex embodiments of the 100 Megabit per second and Gigabit per second Ethernet standards are often referred to

as CSMA/CD protocols, even though they may not fit completely within the classic CSMA/CD definition.

FIG. 3 is a simplified block diagram of a single node in the network switch according to the present invention. The node consists of an integrated circuit 200 comprising ports 201-1, 201-2, . . . 201-X. Each port includes the frame buffer and port management logic normally associated with standard bridges. Also, coupled to each of the ports, is a medium access control MAC unit 202-1, 202-2, . . . 202-X. The MAC units 202-1 to 202-X are coupled to medium independent interfaces MII 203-1, 203-2, . . . 203-X.

In the embodiment of FIG. 3, each of the medium independent interfaces is connected to a connector jack 260-1, 260-2, 260-X. The connector jacks comprise a standard connector to which a cable 270-1, 270-2, 270-X is easily connected by the user. The cable may comprise a coaxial cable for medium independent interfaces based on serial data, or ribbon cables for wider data buses. A variety of mechanical jack configurations can be used as known in the art. For example, coaxial stubs can be mounted on printed circuit boards adjacent each port of the integrated circuits. A short coaxial cable is then connected from stub-to-stub in order to arrange the plurality of integrated circuit chips in a mesh that suits the particular installation. Also, standard ribbon connector jacks can be surface mounted on printed wiring boards adjacent to the integrated circuit. The ribbon cables are connected into the ribbon connector jacks in order to establish the inter-connection.

In alternatives, each of the switches is mounted on a daughter board, with jacks designed to be connected to a mother board in which the data is routed according to the needs of the particular application. In alternative systems, the jacks 260-1 through 260-X are not included, and the medium independent interfaces are routed in the printed wiring board in a hard-wired configuration, designed for a particular installation.

Medium independent interfaces allow for communication by means of the jacks 260-1 to 260-X and cables 270-1 to 270-X, or otherwise, directly with other MAC units on other switch integrated circuits, or to physical layer devices for connection to actual communication media. For example, the MII 203-1 in FIG. 2 is connected directly to a port on another node in the switch. The MII 203-2 in FIG. 2 is connected to a physical layer device 204 for port 2 through jack 271. The physical layer device 204 is connected to a physical transmission medium 205 for the LAN being utilized. The MII 203-X in FIG. 2 is coupled directly to another chip within the switch mesh.

According to one embodiment of the present invention, integrated circuit 200 includes a memory interface 206 for connection directly to an external memory, such as a Rambus dynamic random access memory RDRAM 207. The RDRAM 207 is utilized to store the switch route table 220, and for frame buffers 221 utilized during the routing of frames through the node.

The internal architecture of the integrated circuit 200 can take on a variety of formats. In one preferred embodiment, the internal architecture is based on a standard bus architecture specified for operation at 1 Gigabit per second, or higher. In one example, a 64 bit-wide bus 210 operating at 100 Megahertz is used, providing 6.4 Gigabits per second as a theoretical maximum. Even higher data rates are achievable with faster clocks. The integrated circuit of FIG. 3 includes bus 210 which is connected to a memory arbiter unit 211. Arbiter unit 211 connects the bus 210 to a CPU processor 212 across line 213. The processor 212 is utilized

to execute the route logic for the node. Each of the switch ports 201-1 to 201-X is coupled to the bus 210, and thereby through the arbiter 211 to the CPU 212 and the memory interface 206. Also, flow detect logic 215 is coupled to the bus 210 for the purpose of monitoring the frame received in the node to detect flows, and to generate identifying tags for the purpose of accessing the switch route table in the RDRAM 207. The arbiter 211 provides for arbitration amongst the ports, the flow detect logic, the memory, and the CPU for access to the bus, and other management necessary to accomplish the high speed transfer data from the ports to the frame buffers and back out the port.

A representative location 250 of the switch route table is shown. The location 250 includes a field 251 for the identifying tag, a field 252 for the route header, a field 253 for a block-unblock control bit, and a field 254 or fields for information used in the management of the route table, such as the age of the entry. The tag field 251 may be associated with a location by one or more of using the tag or a portion of the tag in the address, by storing all or part of the actual tag data in the addressed location, or by using other memory tag techniques.

The route header in the preferred embodiment consists of a sequence of route bytes. The first field in a route byte includes information identifying a direction, which corresponds to a particular port on the node, and a second field in the byte includes a count indicating the number of steps through the switch from node to node which should be executed in the direction indicated by the first field. For example, an eight bit route byte in a switch having nodes with four ports, includes a two bit direction field, and a six bit count field, specifying up to 63 hops in one of four directions. A sequence of route bytes is used to specify a route through the switch. Thus, the switch route header uses source routing techniques within the switch for the purposes of managing flow frames through the switch. The source route approach may, for example, in a 4 port node include a field for hops to right, hops to the left, hops up and hops down. The first field may carry information indicating left 4 hops, followed by a field indicating down 2 hops, followed by a field indicating left one hop to exit the switch. Thus, a frame would be transmitted out the left and in the right port of 3 nodes, in the right and out the down port of 1 node, in the top and out the down of 1 node, and in the top and out the left of the last node on the boundary of the switch. A standard Ethernet frame format takes over for transmission through the network outside the switches. As the size of the mesh grows, and the bandwidth handled by the mesh increases, more sophisticated routing techniques are available because of the flexible technology utilized. For larger switches, more than one route exists for frames entering one node and leaving on another node. Thus, the switch can be configured to minimize the number of frames which are blocked in passage through the switch, while maintaining optimum utilization of the bandwidth available through the switch.

The block-unblock field 253 is used during the updating of the switch route table by the host CPU 212 to block routing of frames corresponding to new entries, until it is assured that the first frame in the flow to which the entry corresponds, arrives at its destination before the node begins forwarding following frames in the flow to the destination using the route header, in order to preserve the order of transmission of the frames. The age field 254 is used also by the CPU 212 for the purpose of managing the contents of the route table. Thus, entries which have not been utilized for a certain amount of time are deleted, or used according to

least-recently-used techniques for the purposes of finding locations for new entries. Other control fields (not shown) include a field for storing a count of the number of packets forwarded by the node using this route, a drop/keep field to indicate packets that will be dropped during overflow conditions, a priority "high/low" field for quality of service algorithms, and additional fields reserved for future use, to be defined according to a particular embodiment.

The frame buffer 221 is preferably large enough to hold several frames of the standard LAN format. Thus, a standard Ethernet frame may comprise 1500 bytes. Preferably, the frame buffer 221 is large enough to hold at least one frame for each of the ports on the flow switch.

The flow switch 200 includes more than 2 ports, and preferably 4 or more ports. All the ports are either connected through the media independent interfaces 203-1 through 203-X directly to other chips in the mesh, or to physical layer devices for connection to external communication media.

The router or other management node for the switch may communicate with each of the nodes 200 using well-known management protocols, such as SNMP (simple network management protocol), enhancements of SNMP, or the like. Thus, the RDRAM 207 associated with each node also stores statistics and control data used by the management process in controlling the switch node.

Although in FIG. 3, the RDRAM 207 is shown off the chip 200, alternative embodiments incorporate memory into the switch integrated circuit 200, for more integrated design, smaller footprint for the switch, and other classic purposes for higher integration designs.

The CPU 212 executes the node route logic for the node. A simplified flow chart of the node route process executed by CPU 211 is shown in FIG. 4.

The process begins with the receipt of the frame on a particular port (step 300). The CPU first determines whether the frame carries a route header (step 301). This process is executed in parallel with the transferring of the frame being received to the frame buffer of the node. If the frame carries a route header, then the CPU updates the header by decrementing the hop count, or otherwise updating the information to account for a traversed leg of the route according to the particular switch route technique utilized. The CPU transmits the frame (with updated header) on the port identified by the header (step 302). If at step 301, no switch route header was detected, the flow detect logic is accessed to determine a tag for the frame (step 303). The tag is utilized by the CPU to access entries in the route table (step 304). If a match is found in the route table, then a route header is generated for the frame (step 305). Then, the header is updated (if required), and the frame is transmitted on the port identified by the data in the table (step 302). If at step 304, no match was found in the route table, then the frame is transmitted on a default port (step 306). An alternative technique to transmitting the frame on a default port, is to add a default route header to the frame, and transmit the frame according to the information in the default route header. In this manner, subsequent nodes in the switch will not be required to perform the look-up operation for the purposes of routing the frame. However, it may be desirable to have each node look up the frame in its own route table, in order to insure that if any node already has data useful in forwarding the frame, then that frame will be forwarded appropriately without requiring processing resources of the management process at the default address.

FIG. 5 illustrates the technique executed by the flow detect logic in generating an identifying tag for the frame

being received. FIG. 5 includes the format of a standard Ethernet (802.3) style frame 400. The frame includes a start of frame delimitator SOF in field 401. A destination address is carried in field 402. A source address is carried in field 403, and miscellaneous control information is carried in additional fields 404. A network layer header, such as an Internet protocol header in this example, is found in field 405. Other style network layer headers could be used depending on the particular frame format. The data field of variable length is found at section 406 of the frame. The end of the frame includes a CRC-type checksum field 407 and an end-of-frame delimitator 408. The flow detect logic runs a CRC-type hash algorithm over selected fields in the control header of the frame to generate a pseudo-random tag. Thus, the field 410, the field 411, the field 412, and the field 413 are selected for input into a CRC hash generator 414. The tag generated by the hash generator 414 is supplied on line 415 for use in accessing the route table 416. The route table either supplies a route header on line 417, or indicates a miss on line 418. In this way, the route management software executed by the CPU can make the appropriate decisions.

The embodiment of FIG. 5 selects a particular set of fields within the frame for the purpose of generating the pseudo-random tag. The particular set of fields is selected to correspond to one standard frame format encountered in the network. However, a variety of frame formats may be transmitted within a single Ethernet style of network, although in this example, a CRC-type hash generator is utilized, relying on typical CRC-type algorithms, referred to as polynomial arithmetic, modulo II. This type of arithmetic is also referred to as "binary arithmetic with no carry" or serial shift exclusive-OR feedback. However, a variety of pseudo-random number generation techniques can be utilized, other than CRC-like algorithms. The two primary aspects needed for a suitable pseudo-random hash code are width and chaos, where width is the number of bits in the hash code, which is critical to prevent errors caused by the occurrence of packets which are unrelated but nonetheless result in the same hash being generated, and chaos is based on the ability to produce a number in the hash register that is unrelated to previous values.

Also, according to the present invention, the parsing of the frames incoming for the purposes of producing an address to the look-up table can take other approaches. This parsing can be referred to as circuit identification, because it is intended to generate a number that is unique to the particular path of the incoming frame.

The circuit identification method depends on verifying a match on specific fields of numbers in the incoming frame. There are two common table look-up methods, referred to as binary search and hash coding. The key characteristic of binary search is that the time to locate an entry is proportional to the log base 2 of the number of entries in the table. This look-up time is independent of the number of bits in the comparison, and the time to locate a number is relatively precisely known.

A second, more preferred, method of look-up is based on hash coding. In this technique, a subset of address field or other control fields of the frame are used as a short address to look into the circuit table. If the circuit table contains a match to the rest of the address field, then the circuit has been found. If the table contains a null value, then the address is known not to exist in the table. The hash method has several disadvantages. It requires a mostly empty table to be efficient. The time to find a circuit cannot be guaranteed. The distribution of duplicates may not be uniform, depending on the details of which fields are selected for the initial address generation.

The address degeneracy problem of the hash coding technique is reduced by processing the initial address fragment through a polynomial shift register. This translates the initial address to a uniformly-distributed random number. A typical example of random number generation is the CRC algorithm mentioned above. In a preferred hashing technique, the hardware on the flow switch includes at least a template register, pseudo-random number generation logic and a pseudo-random result register. The template register is loaded to specify bytes of a subject frame to be included in the hash code. The template specifies all protocol-dependent fields for a particular protocol. The fields are not distinguished beyond whether they are included in the hash or not. As the frame is processed, each byte of the initial header is either included in the hash function or it is ignored, based on the template. A hash function is generated based on the incoming packet and the template. The pseudo-random number generator is seeded by the input hash bits selected by the template. The change of a single bit in the input stream should cause a completely unrelated random number to be generated. Most common algorithms for generating pseudo-random numbers are linear-congruential, and polynomial shift methods known in the art. Of course, other pseudo-random number generation techniques are available.

A first field of the pseudo-random number is used as an address for the look-up table. The number of bits in this field depends on the dimensions of the look-up table. For example, if the circuit table has 64,000 possible entries, and the hash number is eight bytes long, the first two bytes are used as an address. The other six bytes are stored as a key in the hash table. If the key in the hash table matches the key in the hash code, then the circuit is identified. The additional bytes in the table for the addressed entry specify the route to be applied. The length of the pseudo-random hash code is critical, to account for the probability that two unrelated frames will result in the same hash number being generated. The required length depends on the size of the routing tables, and the rate of turnover of routes.

The problem with a pure hash code circuit identification technique is that there is a chance of randomly misrouting a packet. The problem arises when you are generating random numbers out of a larger set. There is a chance that two different input patterns will produce the same hash code. Typically, a hash code will be loaded into a table with a known route. Then a second, different, packet will appear that reduces to the same hash code as the one already in the table. The second packet will be falsely identified as having a known route, and will be sent to the wrong address. The exact mechanism of this error can be understood by the well-known statistics of the "birthday problem." The "birthday problem" answers the question, "What is the probability that two people in a group will have the same birthday?" It turns out that the number of people in a group required for there to be a likelihood of two people having the same birthday is quite small. For example, there is a 50% chance that two people out of a group of 23 will have the same birthday.

The probability of a switching error depends on the number of circuits active. For example, if there are no circuits active, then there is no chance that an invalid circuit will be confused with another circuit, since there are no valid circuits. As each circuit is added to the table, it decreases the remaining available space for other numbers by approximately $(\frac{1}{2})^{\text{bits}}$, where "bits" is the number of bits in the hash code. If the hash code is 32 bits long, then each entry into the circuit table will reduce the remaining code space by $(\frac{1}{2})^{32}$, which is equal to 2.32×10^{-10} . The cumulative prob-

ability of not making an error in the circuit table is equal to the product of the individual entry errors up to the size of the table. This is $(1 - (1/2^{32})) \cdot (1 - (1/2^{32})) \cdot (1 - (1/2^{32})) \dots (1 - (1/2^{32}))$, where n is the number of entries in the table. In the case of a 32-bit hash code, and an 8,000-entry circuit table, the probability of making an error in the table would be about 0.7%. With a 64,000-entry circuit table, the probability of an error would be about 39%.

Using a 32-bit hash code and some typical-sized circuit tables indicates that the conventional wisdom is correct. That is, there will be routing errors if only a 32-bit hash code is used. However, if the number of bits in the hash code is increased and probability is recalculated for typical-sized circuit tables, we find that the probability of error quickly approaches zero for hash codes just slightly longer than 32 bits. For example, an 8,000-entry table with a 40-bit hash code will reduce the error rate to 0.003%. A 48-bit hash code will reduce the error to 0.000012%. These calculations show that a pure hash code look-up table can be used if the length of the hash code is longer than 32 bits for typical-size tables.

As a further example, consider the case of a 64-bit hash code. Assuming an 8,000-entry table, the probability of making an error is $2 \cdot 10^{-12}$. Even if the table is completely replaced with new entries every 24 hours, it would take over one billion years for an error to occur. Using a 64-bit hash code with a 64,000-entry table would give a probability of error of 10^{-10} . Assuming the table turned over every day, it would take about 28 million years for an error to occur. An error might occur sooner, but the rate would be negligible. In all cases, there is no realistic chance of making an error based on this routing technique within the lifetime of typical networking equipment.

In a preferred embodiment, filtering mechanisms are implemented on the flow switch integrated circuit, and multiple filters operate in parallel. The circuit look-up table is implemented with external memory much larger than the number of circuits expected to be simultaneously active. This means that the hash pointer generated either points to a valid key or a miss is assumed. There is no linear search for matching key. When a circuit is not found in the table, the packet is routed to a default address. Normally, this default address directs the packet to a stored program router. The router will then parse the packet using standard methods, and then communicate with the flow switch circuit to update the circuit table with the correct entry. All subsequent packets are directly routed by the switch element without further assistance from the router.

Example template organizations for the bridging embodiment, the IP routing embodiment, and the IPX routing embodiment are set forth below.

Example for bridging:

Basic ethernet packet:	Preamble 64 bits are discarded
DestinationAddress:	bytes 1-6 Used
SourceAddress:	bytes 7-12 Used
Packet Type:	bytes 13-14 are ignored (802.3 length)
Data bytes:	15 upto 60 are ignored
CRC:	Last 4 bytes are ignored

The template register is 8 bytes long. Each bit specifies one byte of the header. The first bit corresponds to byte 1 of the DestinationAddress.

The template for bridging is FF-F0-00-00 00-00-00-00

The selector is: Always TRUE. Hierarchy=1 (default to bridging)

Example for IP:

Preamble 64 bits are discarded		
Destination	bytes 1-6	optional
Source	bytes 7-12	optional
Packet type	bytes 13-14	Ignore (802.3 length)
byte 15:	IP byte 1	= version length = optional
byte 16:	IP byte 2	= service type = Ignore
17-18:	IP 3-4	= length = Ignore
19-22:	IP 5-8	= Ignore
23	IP 9	= TTL = optional
24	IP 10	= Proto = optional
25-26	IP 11-12	= Hdr checksum = Ignore
27-30	IP 13-16	= Source IP address = Used
31-34	IP 17-20	= Destination IP address = Used
35-	IP 21-	= Ignore

Assume that optional fields are included in the pseudo-random hash code.

The template would then be: FF-F2-03-03 FC-00-00-00

The selector is: Bytes 13-15=080045, Hierarchy=2

Example for IPX in an Ethernet frame:

Preamble 64 bits are discarded		
Destination	bytes 1-6	Optional
Source	bytes 7-12	Optional
Type	bytes 13-14	Optional (Selector = #137)
byte	IPX	
15-16	1-2	Checksum Ignore
17-18	3-4	Length Ignore
19	5	Hop count Optional
20	6	Type Optional (Selector = 2 or 4)
21-24	7-10	Dest Net Use
25-30	11-16	Dest Host Use
31-32	17-18	Dest Socket Ignore
33-36	19-22	Src Net Use
37-42	23-28	Src Host Use
43-	29-	Ignore
Template (with optional fields):		FF-FC-3F-FC FF-C0-00-00
Selector:	Bytes 13-14 = #137, Hierarchy = 2	

The examples shown are representative, and may not correspond to what would actually be required for any particular application. There are many protocol pattern possibilities. Some combinations may not be resolvable with the hierarchy described in these three examples.

In the embodiment in which there are a number of filters operating in parallel, the flow detect logic includes the template register discussed above, a second register loaded with a template for detecting the specific protocol type represented by the template register. This feeds combinational logic that provides a boolean function, returning a true or false condition based on a string compare of a section of the frame to determine the protocol. A third register is loaded with a hierarchy number, which is used to arbitrate among similar protocols, which might simultaneously appear to be true based on the second protocol detect register. A fourth register is optional, and contains a memory start address which triggers the operation of the filter.

The multiple instantiations of the filters operate in parallel. The filters can be reprogrammed on the fly to support the exact types of traffic encountered. Furthermore, the filters may operate in a pipeline mode along a series of switching nodes. Each protocol returns its hierarchy number when that filter detects the protocol pattern contained in the template. For example, bridging protocol may be defined as true for hierarchy 1 for all frames, if no stronger filter fires, such as an IP or IPX filter, then the bridging filter will be selected as the default.

Thus, the flow detect logic in a preferred system executes a plurality of hash flow analyses in parallel as illustrated by

FIG. 6. Thus in FIG. 6, a received frame is supplied on line 500 in parallel to hash flow logic 1 through hash flow logic N, each flow corresponding to a particular frame format. Also, the received frame is supplied to a hash flow "select" 501 which is used for selecting one of the N flows. The output of flows 1 through N are supplied through multiplexer 502 in FIG. 6, which is controlled by the output of the select flow 501. The output of the select flow 501 causes selection of a single flow on line 503, which is used for accessing the route table by the CPU.

Thus a preferred embodiment of the present invention uses a routing technique base on flow signatures. Individual frames of data move from one of the Ethernet ports to a shared buffer memory at the node. As the data is being moved from the input port to the buffer, a series of hash codes is computed for various sections of the input data stream. Which bits are or are not included in each hash calculation is determined by a stored vector in a vector register corresponding to that calculation. For example, in the most common case of an IP packet, the hash function starts at the 96th bit to find the "0800" code following the link-layer source address, it then includes the "45" code, 32 bits of IP source, 32 bits of IP destination, skips to protocol ID 8 bits, and then at byte 20 takes the source port 16 bits and the destination port 16 bits. The result is a 64 bit random number identifying this particular IP flow.

The hash code is looked up in or used to access a local memory. If the code is found, it means that this flow type has been analyzed previously, and the node will know to apply the same routing as applied to the rest of the flow. If there is no entry corresponding to this hash code, it means that the flow has not been seen lately, and the node will route the frame to a default destination. A least recently used algorithm, or other cache replacement scheme, is used to age flow entries out of the local tables.

In practice, many filters operate simultaneously. For example, filters may be defined for basic bridging, IP routing, sub-variants, AppleTalk, and so on. The actual limit to the number of filters will be determined by the available space on the ASIC. The logic of the filters is basically the same for all the filters. The actual function of each filter is defined by a vector register specifying which bits are detected.

A second feature is the use of multi-level filters. In the common case simultaneously supporting bridging, IP, and IPX; about ten filters operate in parallel. An additional level of coding is used to select which of the other filters is to be used as the relevant hash code. This second level filter would detect whether the flow was IP or IPX for example.

In the case where the flow is not recognized, it is passed to the default route. As the packet passes along the default route, additional nodes may examine the packet and detect its flow type based on different filters or on a different set of flow signatures (hash table entries) stored. This method of cascading filters and tables allows for the total size and speed of the mesh to be expanded by adding nodes. Ultimately, if a packet can not be routed by any of the nodes along the default route, the packet will arrive at the final default router, typically a NetBuilder2. The default router will analyze the packet using standard parsing methods to determine its correct destination. A flow signature will be installed in an appropriate node, or nodes, of the mesh so that subsequent flows of the same signature can be routed autonomously without further intervention.

A flow effectively defines a "circuit" or a "connection"; however, in standard Ethernet design, packets are treated

individually without any regard to a connection. Typically a router will analyze every single packet as if it had never seen it before, even though the router might have just processed thousands of identical packets. This is obviously a huge waste of routing resources. The automation of this flow analysis with multiple levels of parallel and cascaded hashing algorithms combined with a default router is believed to be a significant improvement over existing routing methods.

Flow based switching is also critical to ensuring quality of service guarantees for different classes of traffic.

FIG. 7 is a flow chart illustrating the process executed in the router or other management node, whenever a frame is received which does not have a switch route header. Thus, the process of FIG. 7 begins at step 700 where a frame is received in the router, such as the router 150 in FIG. 2. The router applies the multiprotocol routing techniques to determine the destination of the frame. Based on the destination, and other information about the flows within the switch, switch route headers are generated for nodes in the switch (step 701). Thus, a different route header is generated for each node in the switch mesh, and correlated with the tag which would be generated according to the received frame at each node. Next, a message is sent to the nodes in the switch to update the route tables with the new route headers, and to block frames which match the tag of the frame being routed (block 702).

After step 702, the frame is forwarded from the router to its destination (step 703). After the frame has been forwarded to its destination, the router sends a message to all of the nodes in the switch to unblock frames which have a matching tag (step 704). This blocking and unblocking protocol is used to preserve the order in which frames are transmitted through the switch, by making sure that the first frame of a single flow arrives at its destination ahead of following frames.

Logic in the nodes for the purpose of accomplishing the blocking and unblocking operation take a variety of formats. In one example, the entry at each location in the route table includes a field which indicates whether the flow is blocked or not. When an entry is first made in the route table, the blocking field is set. Only after a special instruction is received to unblock the location, is the blocking field cleared, and use of the location allowed at the switch node.

Accordingly, in the preferred system the atomic network switch according to the present invention is based on repeated use of a simple 4-port switch integrated circuit. The integrated circuits are interconnected to create a mesh with a large pool of bandwidth across many ports. The links that interconnect the integrated circuits run according to a LAN protocol, at preferably 100 megabits per second or higher, such as a gigabit per second. Individual ports act as autonomous routers between the boundaries of the switch according to the switch route protocol which is layered on top of the standard frame format. The overall bandwidth of the switch can be arbitrarily increased by adding more atomic nodes to the switch. Using a well-understood and simple interface based on standard Ethernet LAN protocols, vastly simplifies the implementation of each node in the switch, because each is able to rely on well understood MAC logic units and port structures, rather than proprietary complex systems of prior atomic LANS. Furthermore, any node of any switch can be connected to a physical layer device that connects to an Ethernet medium, or can be disconnected from the Ethernet medium and connected to another node switch to readily expand and change the topology of the switch. The fine granularity and scalability of the mesh

architecture, combined with the ability to optimize the topology of the switch for a particular environment allow implementation of a high bandwidth, low cost network switch.

A high bandwidth and very flexible network switch is achievable according to the present invention with a simple, scalable, low-cost architecture.

The foregoing description of a preferred embodiment of the invention has been presented for purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously, many modifications and variations will be apparent to practitioners skilled in this art. It is intended that the scope of the invention be defined by the following claims and their equivalents.

What is claimed is:

1. For a network switch including a mesh of interconnected network switch nodes, a network switch node comprising:

a set of ports having more than two members, and the ports in the set including respective medium access control units for transmission and reception of data frames according to a network protocol, the ports in the set of ports being connectable to a port on another network switch node inside the mesh, or to a network communication medium outside the mesh; and

node route logic, coupled with the set of ports, which monitors frames received by the set of ports to route a received frame for transmission according to the network protocol to a selected port in the set of ports, including logic to select the selected port according to rules for navigating through the mesh inside to the network switch, and wherein the node route logic forwards the received frame for transmission to a default location of a multiprotocol router resource associated with the switch when the node route logic cannot otherwise determine a route for the received frame.

2. The network switch node of claim 1, wherein the network protocol comprises a connectionless protocol.

3. The network switch node of claim 1, wherein the network protocol comprises an Ethernet protocol.

4. The network switch node of claim 1, wherein the network protocol comprises an Ethernet, full duplex protocol.

5. The network switch node of claim 1, wherein ports in the set of ports include medium independent interfaces for the network protocol.

6. The network switch node of claim 1, further including: route table memory, coupled with the node route logic, having a set of accessible locations for storing switch route data;

flow detect logic, coupled with the set of ports, which monitors frames received by the set of ports and generates an identifying tag for use in accessing the route table memory;

wherein the node route logic includes logic which determines whether the received frame includes a switch route field indicating a port in the set of ports, and if the received frame includes a switch route field, updates the switch route field, and forwards the received frame with the updated switch route field to the port indicated by the switch route field, and if the received frame does not include a switch route field, accesses the route table memory using the identifying tag generated in the flow detect logic to retrieve switch route data indicating a

port in the set of ports, adds a switch route field to the received frame, and forwards the received frame with the switch route field to the port indicated by the switch route data.

7. The network switch node of claim 6, wherein the default location includes a default port and wherein the node route logic forwards the received frame for transmission on the default port in the set of ports when the switch route table does not include switch route data for the identifying tag.

8. The network switch node of claim 7, wherein the default port is coupled to a route to a multi-protocol, network route processor at which switch route data is generated.

9. The network switch node of claim 6, including logic to receive switch route data from a remote system for a particular identifying tag, to store the switch route data in the route table memory in association with the particular identifying tag, and to block frames having the particular identifying tag until notification is received that it is clear to forward frames having the particular identifying tag, and after notification is received that it is clear to forward frames having the particular identifying tag, forward frames having the particular tag according to the switch route data.

10. The network switch node of claim 6, wherein the default location includes a default port and wherein the node route logic forwards the received frame for transmission on the default port in the set of ports when the route table memory does not include switch route data for the identifying tag; and further including:

logic to receive switch route data from a remote system for a particular identifying tag, to store the switch route data in the route table memory in association with the particular identifying tag, and to block frames having the particular identifying tag until notification is received that it is clear to forward frames having the particular identifying tag, and after notification is received that it is clear to forward frames having the particular identifying tag, forward frames having the particular tag according to the switch route data.

11. The network switch node of claim 10, wherein the default port is coupled to a route to a multi-protocol, network route processor at which switch route data is generated.

12. The network switch node of claim 6, wherein the flow detect logic comprises:

logic which computes a plurality of hash values in response to respective sets of control fields in a received frame, where the respective sets of control fields correlate with respective network frame formats; and

logic which determines a particular network frame format for a received frame, and selects one of the plurality of hash values as the identifying tag in response to the particular network frame format.

13. The network switch node of claim 12, wherein the hash values comprise pseudo-random codes.

14. The network switch node of claim 6, wherein the flow detect logic comprises:

logic which computes a hash value in response to a set of control fields in a received frame, where the set of control fields correlates with a network frame format, and applies the hash value as the identifying tag.

15. The network switch node of claim 14, wherein the hash value comprises a pseudo-random code.

16. The network switch node of claim 1, wherein the network protocol comprises an Ethernet protocol, specified for operation at 100 Megabits per second.

19

17. The network switch node of claim 16, wherein the Ethernet protocol comprises a full duplex protocol.

18. The network switch node of claim 1, wherein said set of ports and said node route logic comprise elements of a single integrated circuit.

19. The network switch node of claim 18, wherein ports in the set of ports include medium independent interfaces for the network protocol, and the network protocol comprises an Ethernet protocol, specified for operation at 100 Megabits per second or higher.

20. The network switch node of claim 19, wherein the Ethernet protocol comprises a full duplex protocol.

21. The network switch node of claim 1, wherein ports of the set of ports include medium independent interfaces for the network protocol, the medium independent interfaces defining a particular bus configuration, and further including connectors coupled to the medium independent interfaces adapted to receive cables configured according to the particular bus configuration.

22. An integrated circuit, comprising:

a set of ports for access to respective communication media, the set of ports having more than two members, and the ports in the set including respective medium access control logic for a network protocol;

a memory interface for connection to a route table memory having a set of accessible locations for storing switch route data;

flow detect logic, coupled with the set of ports, which monitors frames received by the set of ports and generates an identifying tag for use in accessing the route table memory; and

node route logic, coupled with the flow detect logic, the memory interface and the set of ports, which monitors frames received by the set of ports to route a received frame for transmission to a port in the set of ports, the node route logic determining whether the received frame includes a switch route field indicating a port in the set of ports, and if the received frame includes a switch route field, updates the switch route field, and forwards the received frame with the updated switch route field to the port indicated by the switch route field, and if the received frame does not include a switch route field, accesses the route table memory through the memory interface using the identifying tag generated in the flow detect logic to retrieve switch route data indicating a port in the set of ports, adds a switch route field to the received frame, and forwards the received frame with the switch route field to the port indicated by the switch route data and if the route table memory does not include switch route data for the identifying tag, then forwards the received frame to a default location of a multiprotocol router resource associated with the switch.

23. The integrated circuit of claim 22, wherein the network protocol comprises a connectionless protocol.

24. The integrated circuit of claim 22, wherein the network protocol comprises an Ethernet protocol.

25. The integrated circuit of claim 24, wherein the Ethernet protocol comprises a full duplex protocol.

26. The integrated circuit of claim 22, wherein ports in the set of ports include medium independent interfaces for the network protocol.

27. The integrated circuit of claim 22, wherein the default location includes a default port and wherein the node route logic forwards the received frame for transmission on the default port in the set of ports when the switch route table does not include switch route data for the identifying tag.

20

28. The integrated circuit of claim 27, including logic to receive switch route data from a remote system for a particular identifying tag, to store the switch route data in the route table memory in association with the particular identifying tag, and to block frames having the particular identifying tag until notification is received that it is clear to forward frames having the particular identifying tag, and after notification is received that it is clear to forward frames having the particular identifying tag, forward frames having the particular identifying tag according to the switch route data.

29. The integrated circuit of claim 22, wherein the default location includes a default port and wherein the node route logic forwards the received frame for transmission on the default port in the set of ports when the route table memory does not include switch route data for the identifying tag; and further including:

logic to receive switch route data from a remote system for a particular identifying tag, to store the switch route data in the route table memory in association with the particular identifying tag, and to block frames having the particular identifying tag until notification is received that it is clear to forward frames having the particular identifying tag, and after notification is received that it is clear to forward frames having the particular identifying tag, forward frames having the particular identifying tag according to the switch route data.

30. The integrated circuit of claim 22, wherein the flow detect logic comprises:

logic which computes a plurality of hash values in response to respective sets of control fields in a received frame, where the respective sets of control fields correlate with respective network frame formats; and

logic which determines a particular network frame format for a received frame, and selects one of the plurality of hash values as the identifying tag in response to the particular network frame format.

31. The integrated circuit of claim 30, wherein the hash values comprise pseudo-random codes.

32. The integrated circuit of claim 22, wherein the flow detect logic comprises:

logic which computes a hash value in response to set of control fields in a received frame, where the set of control fields correlates with a network frame format, and applies the hash value as the identifying tag.

33. The integrated circuit of claim 32, wherein the hash value comprises a pseudo-random code.

34. The integrated circuit of claim 22, including an embedded bus interconnecting the set of ports, the flow detect logic, the node route logic and the memory interface.

35. The integrated circuit of claim 22, wherein the network protocol comprises an Ethernet protocol, specified for operation at 100 Megabits per second or higher.

36. The integrated circuit of claim 35, wherein the Ethernet protocol comprises a full duplex protocol.

37. The integrated circuit of claim 35, including a bi-directional, embedded bus interconnecting the set of ports, the flow detect logic, the node route logic and the memory interface, the embedded bus specified for operation at 1 Gigabit per second or higher.

38. The integrated circuit of claim 22, including the route table memory on the integrated circuit.

39. A network switch, comprising:

a plurality of switch nodes;

a first set of communication links, communication links in the first set coupled between switch nodes in the plurality of switch nodes internal to the network switch;

a second set of communication links, communication links in the second set comprising network links external to the network switch;

the respective switch nodes in the plurality of switch nodes including

a set of ports connected to respective communication links in either the first set of communication links or the second set of communication links, the set of ports having more than two members, and the ports in the set including respective medium access control logic for a network protocol;

route table memory having a set of accessible locations for storing switch route data which specify routes through the plurality of switch nodes;

flow detect logic, coupled with the set of ports, which monitors frames received by the set of ports and generates an identifying tag for use in accessing the route table memory; and

node route logic, coupled with the flow detect logic, the route table memory and the set of ports, which monitors frames received by the set of ports to route a received frame for transmission to a port in the set of ports, the node route logic determining whether the received frame includes a switch route field indicating a port in the set of ports, and if the received frame includes a switch route field, updates the switch route field, and forwards the received frame with the updated switch route field to the port indicated by the switch route field, and if the received frame does not include a switch route field, accesses the route table memory using the identifying tag generated in the flow detect logic to retrieve switch route data indicating a port in the set of ports, adds a switch route field to the received frame, and forwards the received frame with the switch route field to the port indicated by the switch route data, and if the route table memory does not include switch route data corresponding to the identifying tag, then forwarding the received frame to a default location of a multiprotocol router resource associated with the switch.

40. The network switch of claim 39, wherein the network protocol for ports in the set of ports on the respective switch nodes comprises a connectionless protocol.

41. The network switch of claim 39, wherein the network protocol for ports in the set of ports on the respective switch nodes comprises an Ethernet protocol.

42. The network switch of claim 41, wherein the Ethernet protocol comprises a full duplex protocol.

43. The network switch of claim 39, wherein ports in the set of ports on the respective switch nodes include medium independent interfaces for the network protocol.

44. The network switch of claim 39, wherein the default location includes a default port and wherein the node route logic on the respective switch nodes forwards the received frame for transmission on the default port in the set of ports when the switch route table does not include switch route data for the identifying tag.

45. The network switch of claim 44, wherein the default port is coupled to a route to a multi-protocol, network route processor at which switch route data is generated.

46. The network switch of claim 39, including logic on the respective switch nodes to receive switch route data from a remote system for a particular identifying tag, to store the

switch route data in the route table memory in association with the particular identifying tag, and to block frames having the particular identifying tag until notification is received that it is clear to forward frames having the particular identifying tag, and after notification is received that it is clear to forward frames having the particular identifying tag, forward frames having the particular identifying tag according to the switch route data.

47. The network switch of claim 39, wherein the node route logic on the respective switch nodes forwards the received frame for transmission on a default port in the set of ports when the route table memory does not include switch route data for the identifying tag; and further including:

logic on the respective switch nodes to receive switch route data from a remote system for a particular identifying tag, to store the switch route data in the route table memory in association with the particular identifying tag, and to block frames having the particular identifying tag until notification is received that it is clear to forward frames having the particular identifying tag, and after notification is received that it is clear to forward frames having the particular identifying tag, forward frames having the particular identifying tag according to the switch route data.

48. The network switch of claim 47, wherein the default port is coupled to a route to a multi-protocol, network route processor at which switch route data is generated.

49. The network switch of claim 39, wherein the flow detect logic on the respective switch nodes comprises:

logic which computes a plurality of hash values in response to respective sets of control fields in a received frame, where the respective sets of control fields correlate with respective network frame formats; and

logic which determines a particular network frame format for a received frame, and selects one of the plurality of hash values as the identifying tag in response to the particular network frame format.

50. The network switch of claim 49, wherein the hash values comprise pseudo-random codes.

51. The network switch of claim 39, wherein the flow detect logic on the respective switch nodes comprises:

logic which computes a hash value in response to set of control fields in a received frame, where the set of control fields correlates with a network frame format, and applies the hash value as the identifying tag.

52. The network switch of claim 51, wherein the hash value comprises a pseudo-random code.

53. The network switch of claim 39, wherein the network protocol for ports in the set of ports on the respective switch nodes comprises an Ethernet protocol, specified for operation at 100 Megabits per second or higher.

54. The network switch of claim 53, wherein the Ethernet protocol comprises a full duplex protocol.

55. The network switch of claim 39, wherein the MAC logic for ports in the set of ports on the respective switch nodes executes the same network protocol for all ports in the set of ports.

56. The network switch of claim 39, wherein ports in the set of ports on the respective switch nodes include medium independent interfaces for the network protocol, the medium independent interfaces defining a particular bus configuration, and further include connectors coupled to the medium independent interfaces adapted to receive cables configured according to the particular bus configuration.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 5,802,054
DATED : September 1, 1998
INVENTOR(S) : Donald M. Bellenger

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

On title page, item 22 Filed:
replace "August 16, 1996"
with --August 15, 1996--.

Signed and Sealed this
Twenty-third Day of March, 1999

Attest:



Q. TODD DICKINSON

Attesting Officer

Acting Commissioner of Patents and Trademarks



US005720032A

United States Patent [19]

[11] Patent Number: 5,720,032

Picazo, Jr. et al.

[45] Date of Patent: *Feb. 17, 1998

[54] NETWORK PACKET SWITCH USING SHARED MEMORY FOR REPEATING AND BRIDGING PACKETS AT MEDIA RATE

[75] Inventors: Jose J. Picazo, Jr., San Jose; Paul Kakul Lee, Union City; Robert P. Zager, San Jose, all of Calif.

[73] Assignee: Compaq Computer Corporation

[*] Notice: The term of this patent shall not extend beyond the expiration date of Pat. No. 5,432,907.

[21] Appl. No.: 790,163

[22] Filed: Jan. 28, 1997

Related U.S. Application Data

[62] Division of Ser. No. 694,491, Aug. 7, 1996, which is a continuation of Ser. No. 498,116, Jul. 5, 1995, which is a continuation-in-part of Ser. No. 881,931, May 12, 1992, Pat. No. 5,432,907.

[51] Int. Cl. H04J 3/02

[52] U.S. Cl. 395/200.2; 395/200.02; 370/401; 370/351; 370/404

[58] Field of Search 395/200.02, 200.2, 395/200.15; 370/401-408, 351

[56] References Cited

U.S. PATENT DOCUMENTS

Re. 33,426	11/1990	Sugimoto et al.	370/402
4,627,052	12/1986	Hoare et al.	370/402
4,715,030	12/1987	Koch et al.	370/85
4,825,435	4/1989	Amundsen et al.	370/501
4,901,312	2/1990	Hui et al.	370/403
4,922,503	5/1990	Leone	370/402
4,982,400	1/1991	Ebersole	370/407
5,060,228	10/1991	Tsutsui et al.	370/402
5,088,032	2/1992	Bosak	395/200.15
5,179,554	1/1993	Lomiccka et al.	370/398
5,214,646	5/1993	Yacoby	370/402
5,251,213	10/1993	Vidlock et al.	370/403
5,264,742	11/1993	Sougen	307/465
5,276,681	1/1994	Tobagi et al.	370/229

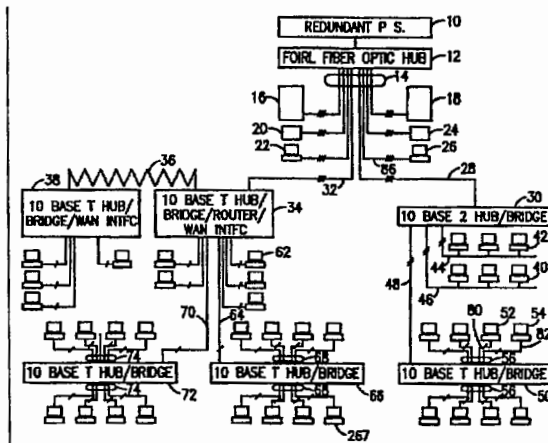
5,299,195	3/1994	Shah	370/462
5,301,303	4/1994	Abraham et al.	395/500
5,321,695	6/1994	Fank, Jr.	370/401
5,329,618	7/1994	Moati et al.	395/200.02
5,396,495	3/1995	Moorwood et al.	370/408
5,440,546	8/1995	Bianchini, Jr. et al.	370/60
5,457,681	10/1995	Gaddis et al.	370/56
5,477,547	12/1995	Sugiyama et al.	370/85
5,521,913	5/1996	Gridley	370/58.2

Primary Examiner—Christopher B. Shin
Attorney, Agent, or Firm—Jenkins & Gilchrist

[57] ABSTRACT

A hub circuit with an integrated bridge circuit carried out in software including a switch for bypassing the bridge process such that the two bridged networks effectively become one network. An in-band management process in software is disclosed which receives and executes network management commands received as data packets from the LANs coupled to the integrated hub/bridge. Also, hardware and software to implement an isolate mode where data packets which would ordinarily be transferred by the bridge process are not transferred except in-band management packets are transferred to the in-band management process regardless of which network from which they arrived. Also disclosed, a packet switching machine having shared high-speed memory with multiple ports, one port coupled to a plurality of LAN controller chips coupled to individual LAN segments and an Ethernet microprocessor that sets up and manages a receive buffer for storing received packets and transferring pointers thereto to a main processor. The main processor is coupled to another port of the memory and analyzes received packets for bridging to other LAN segments or forwarding to an SNMP agent. The main microprocessor and the Ethernet processor coordinate to manage the utilization of storage locations in the shared memory. Another port is coupled to an uplink interface to higher speed backbone media such as FDDI, ATM etc. Speeds up to media rate are achieved by only moving pointers to packets around in memory as opposed to the data of the packets itself. A double password security feature is also implemented in some embodiments to prevent accidental or intentional tampering with system configuration settings.

10 Claims, 13 Drawing Sheets



(do) to be...

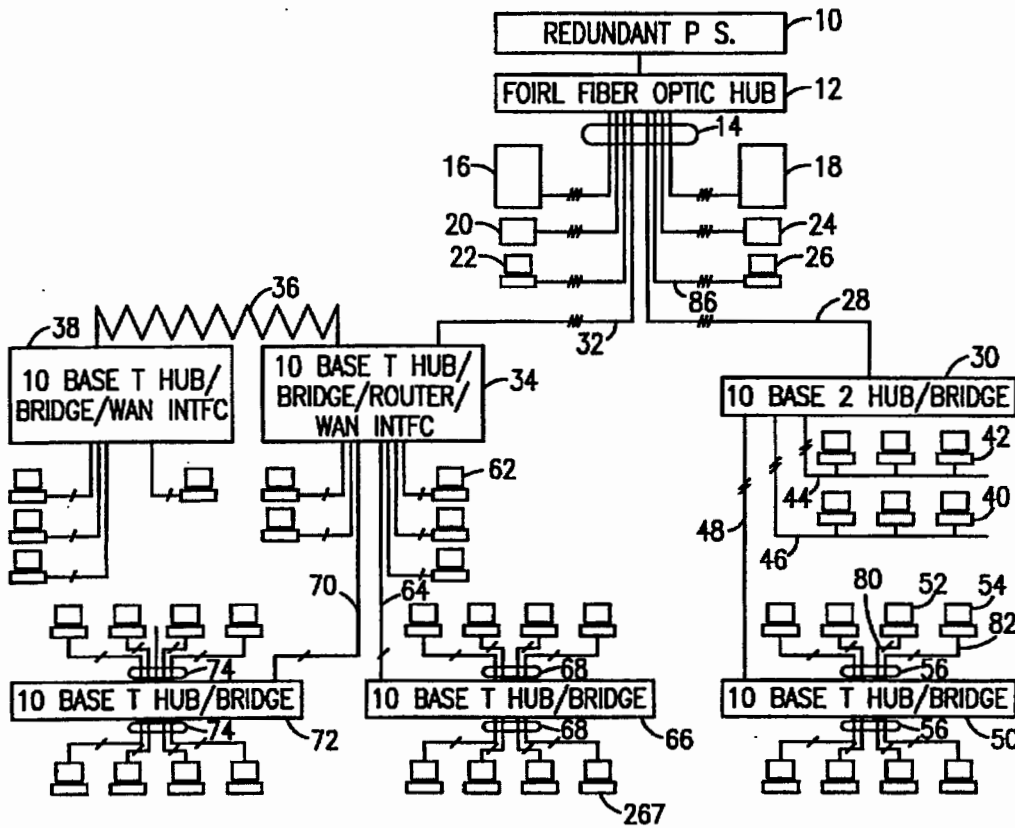


FIG. 1

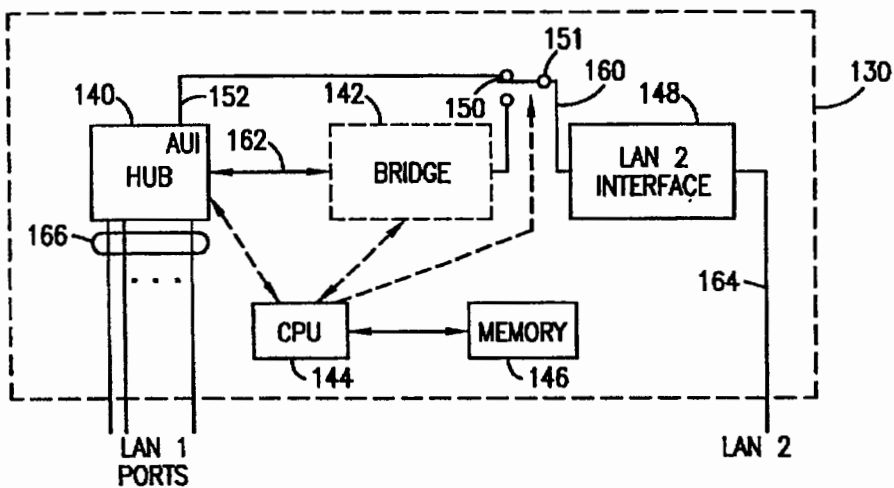


FIG. 2

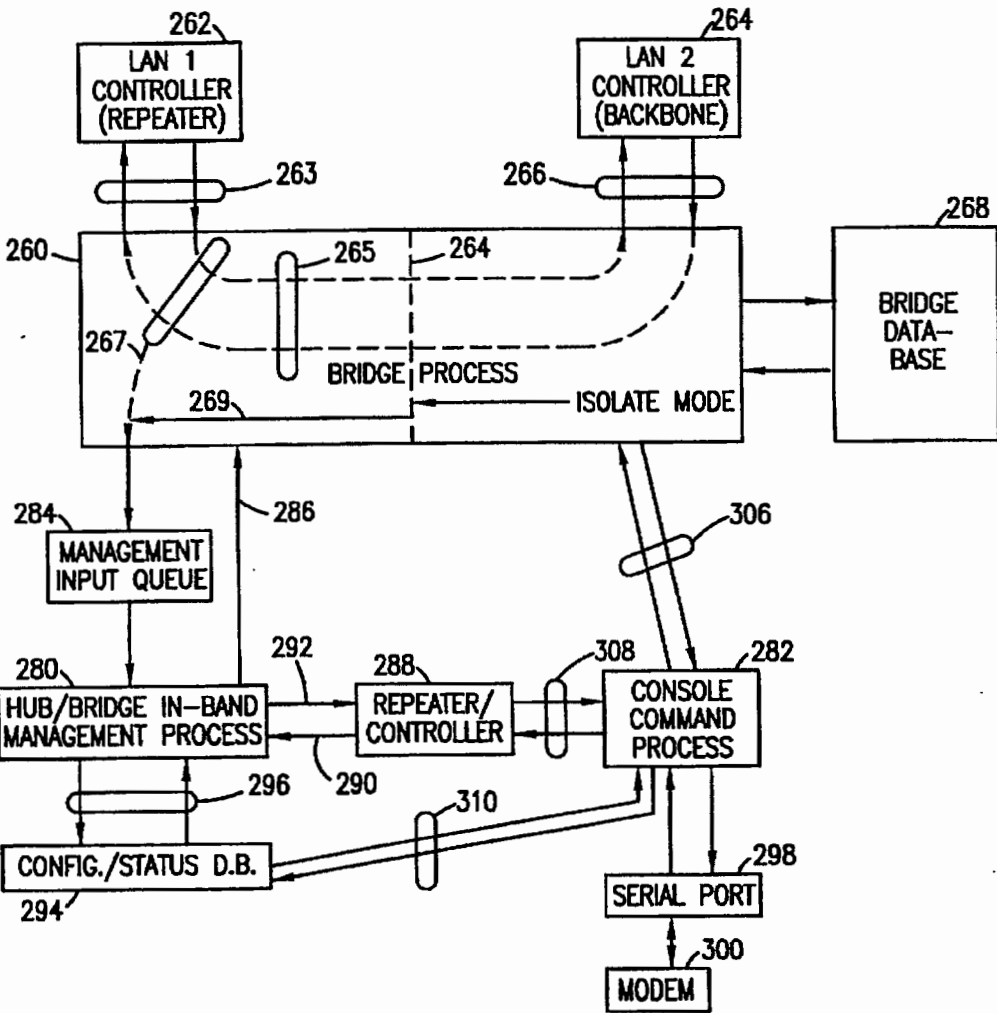


FIG. 4

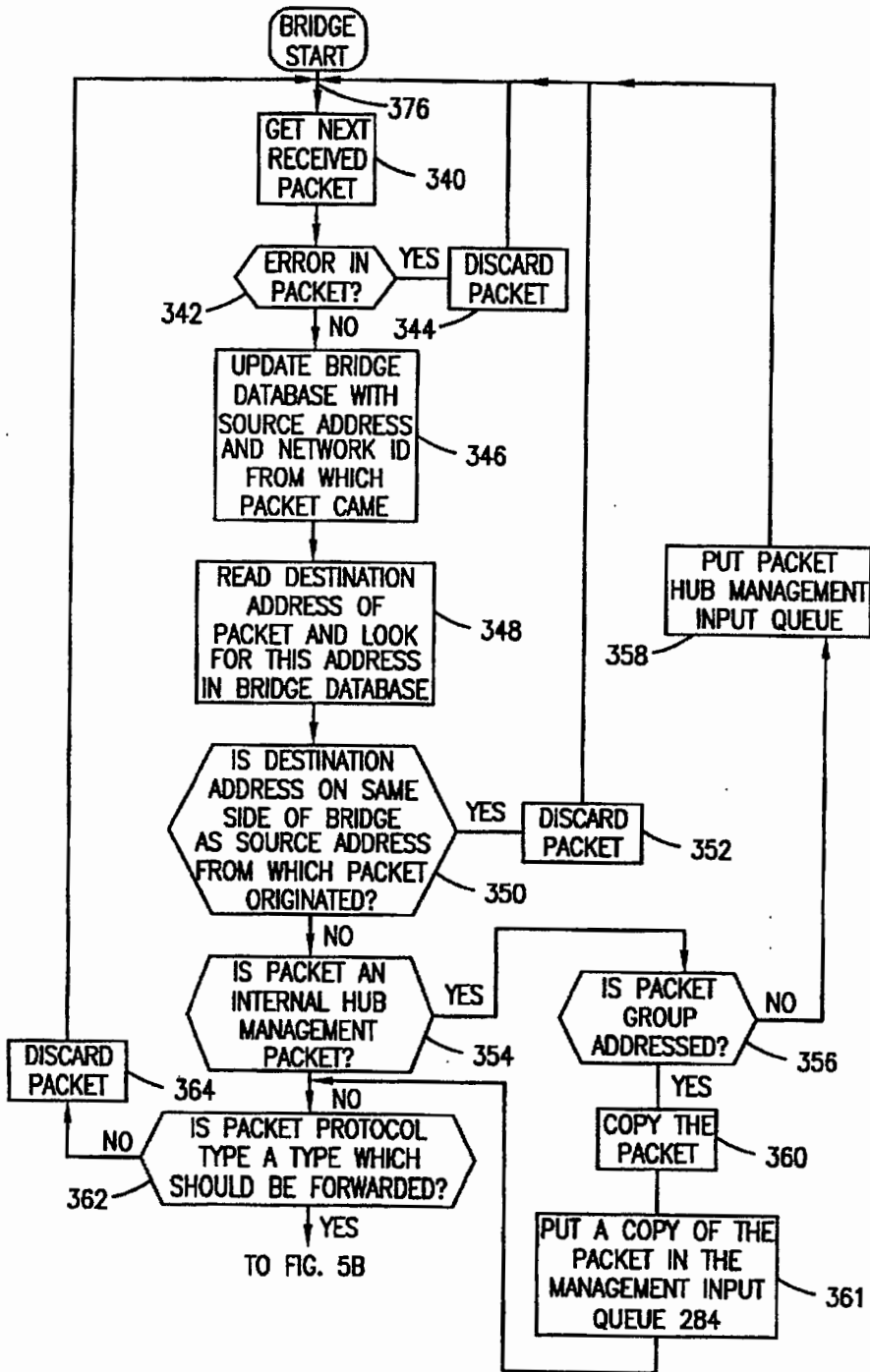


FIG. 5A

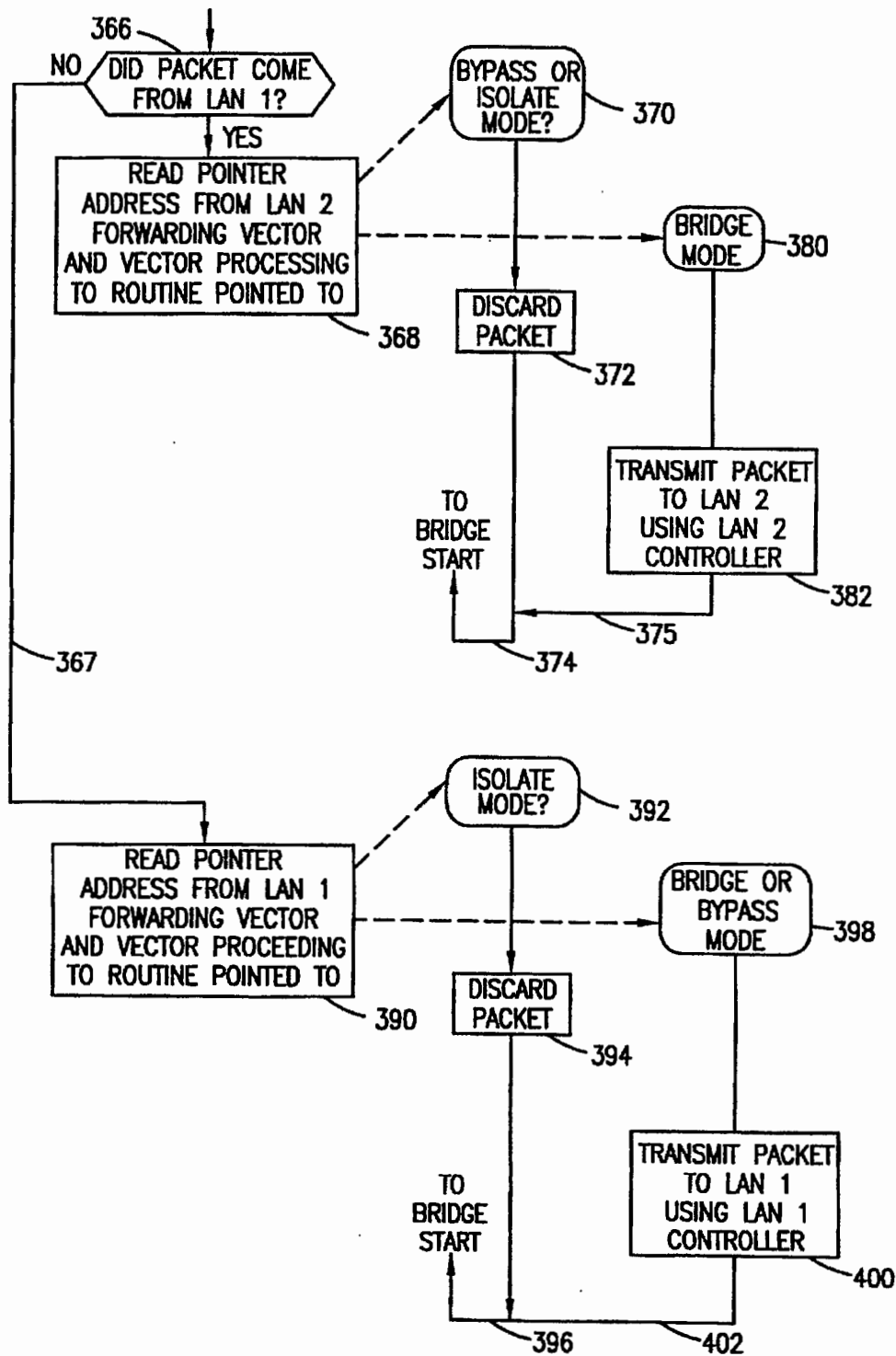


FIG. 5B

FIG. 6A

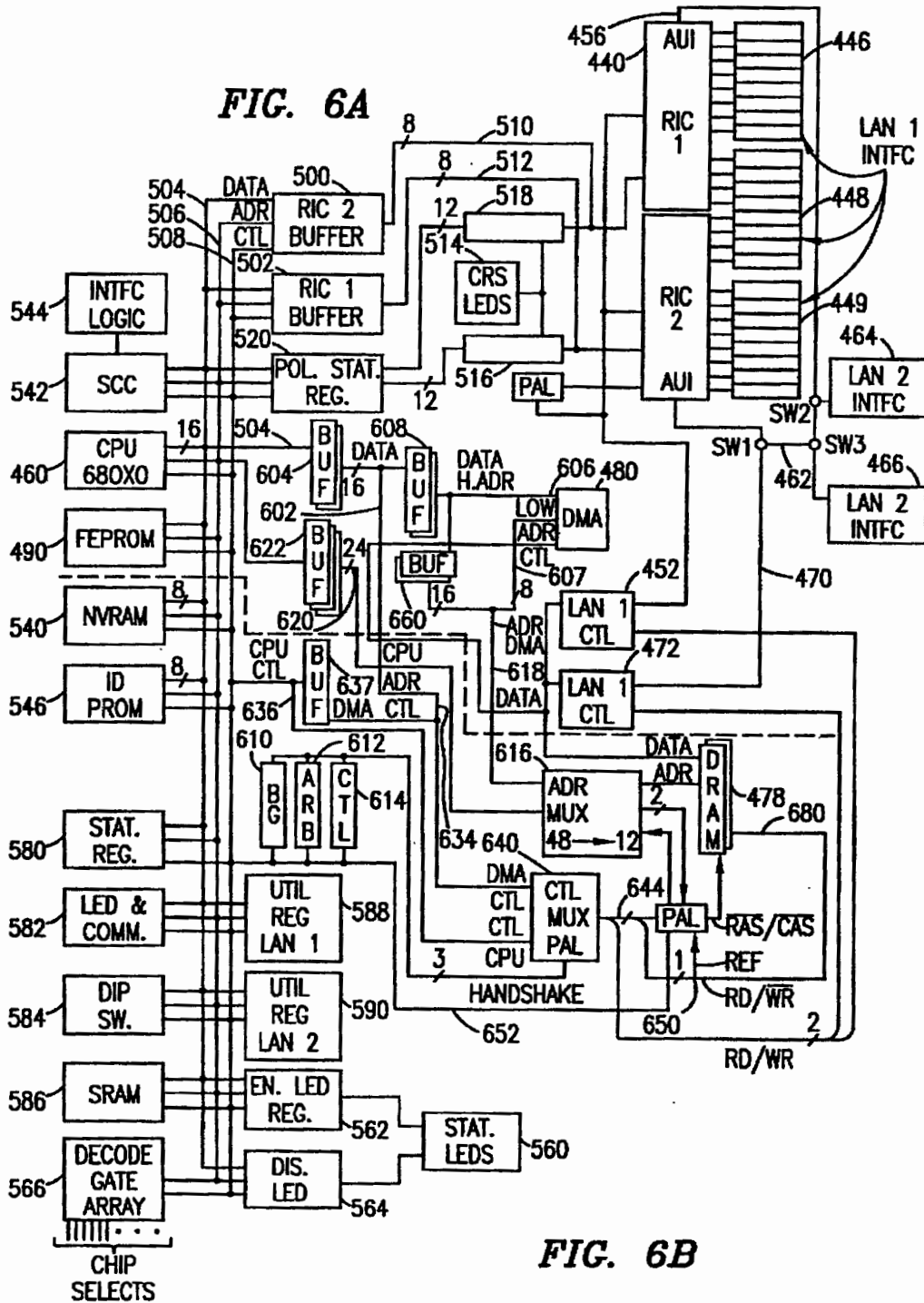


FIG. 6B

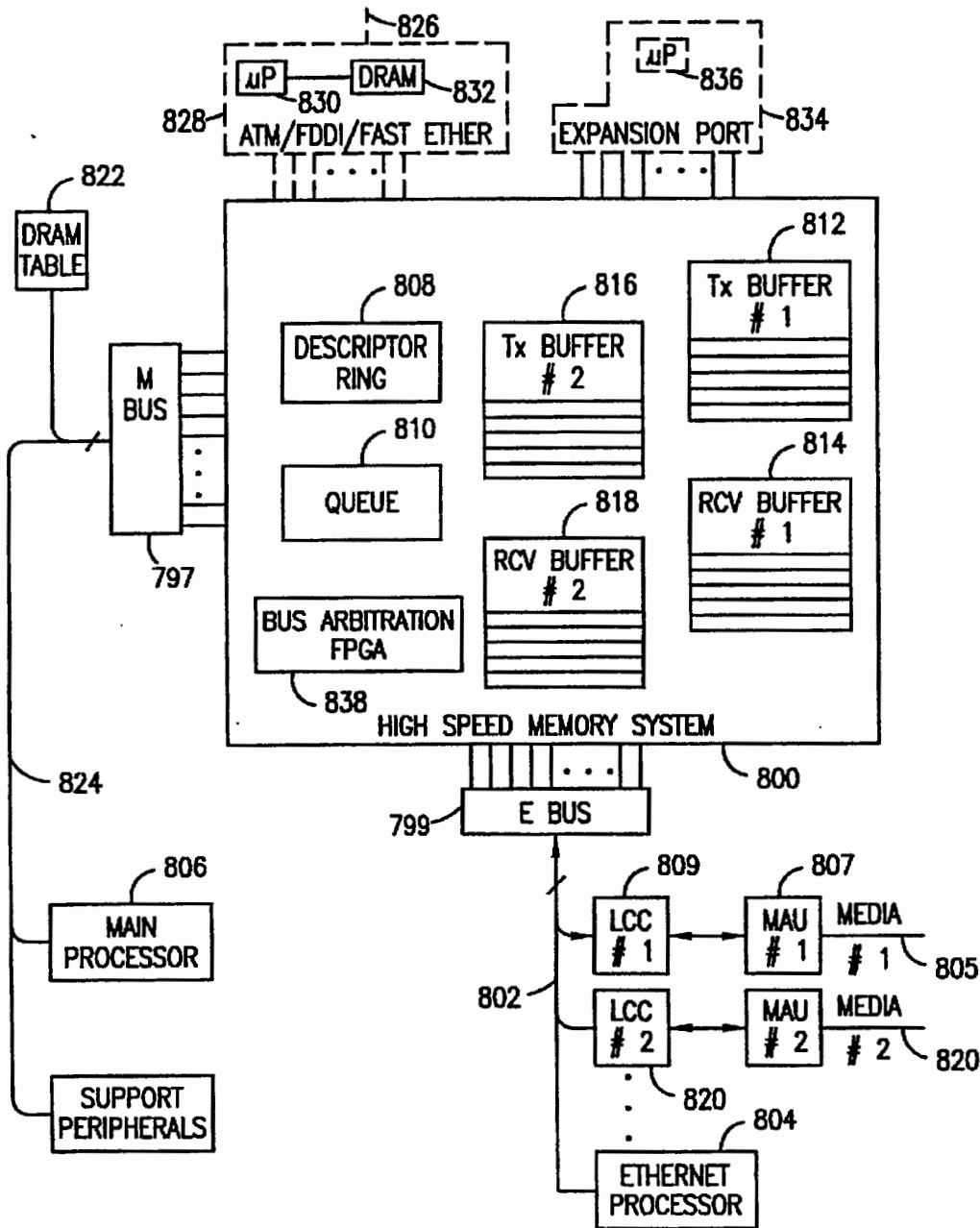


FIG. 7

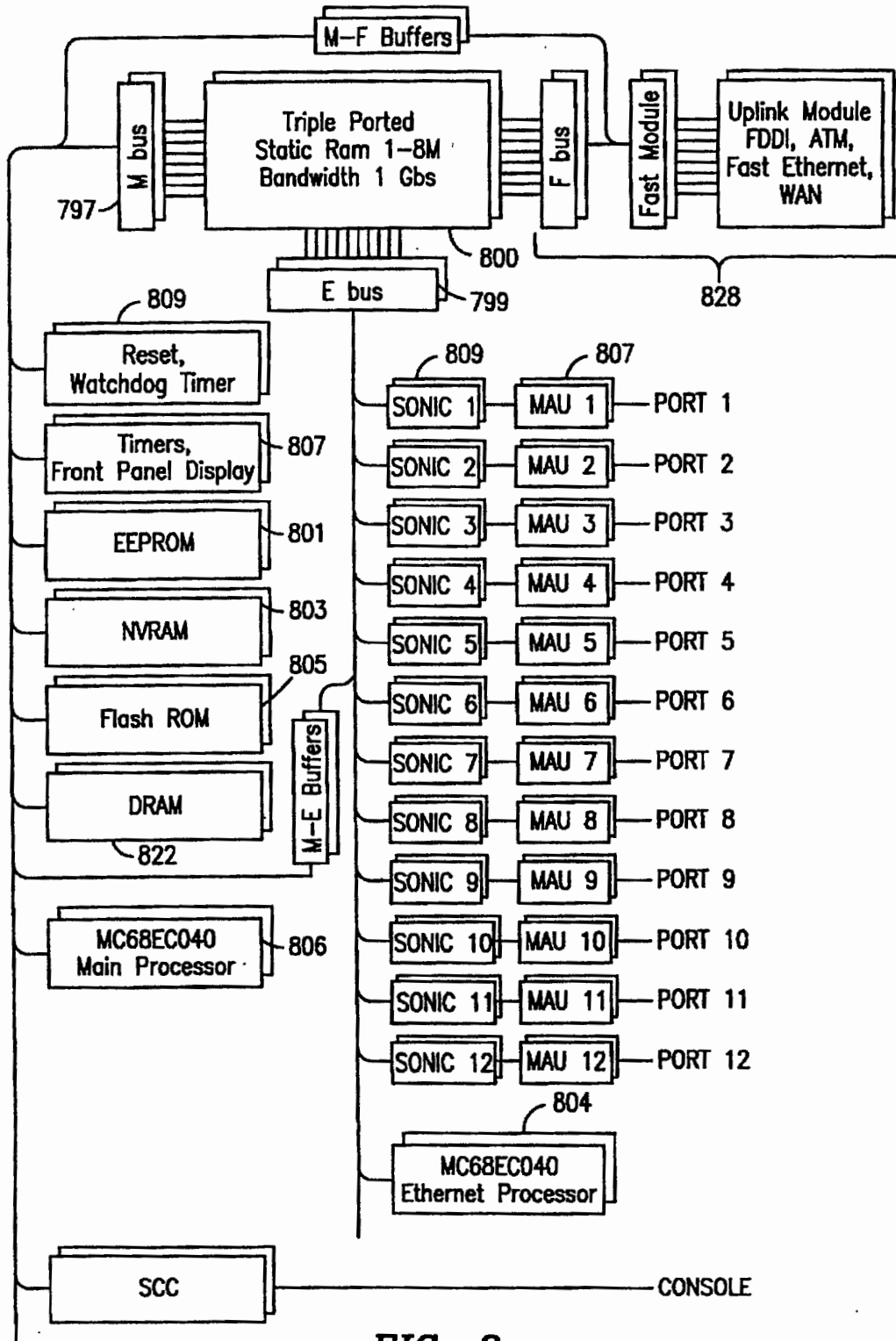


FIG. 8

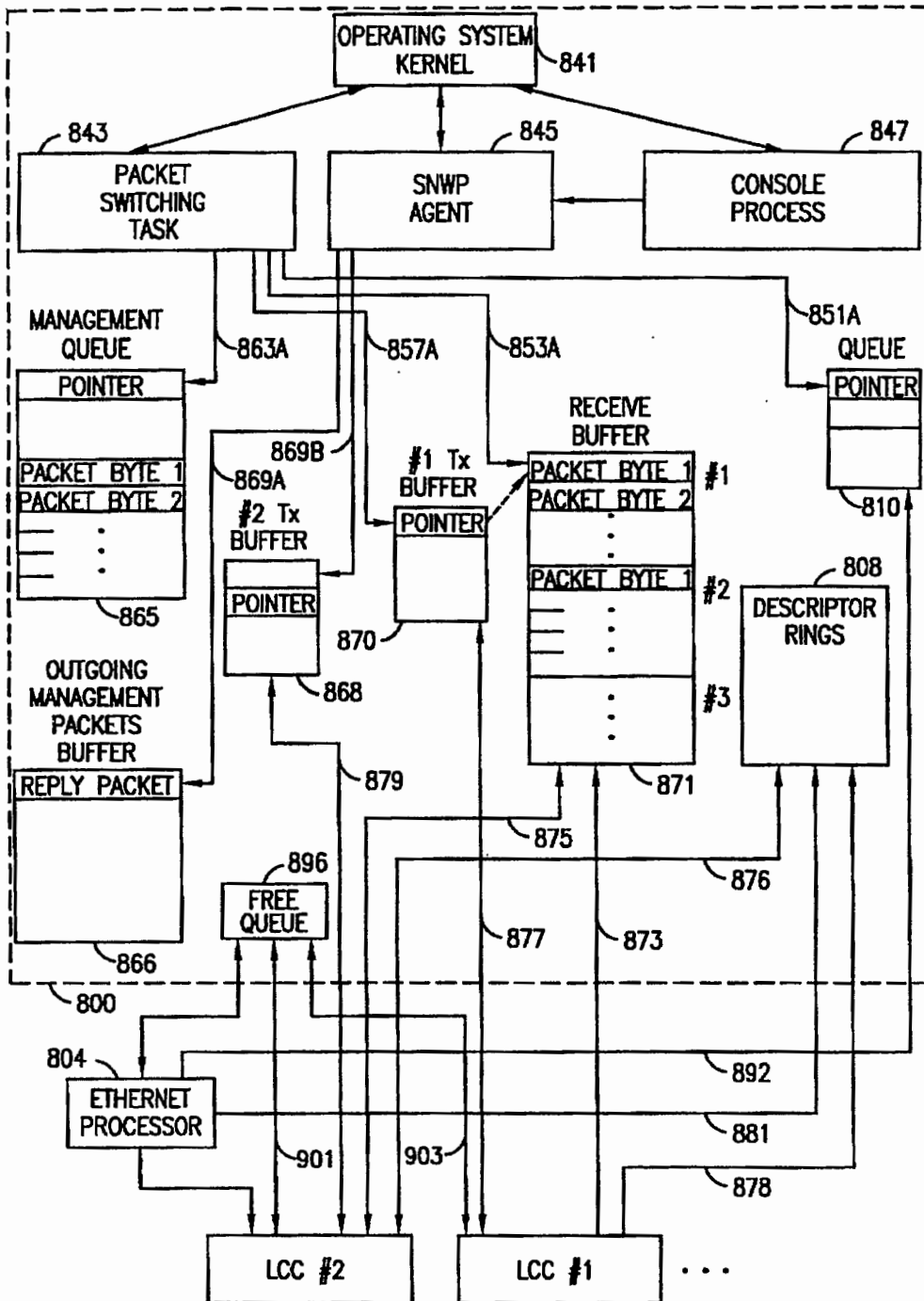


FIG. 9

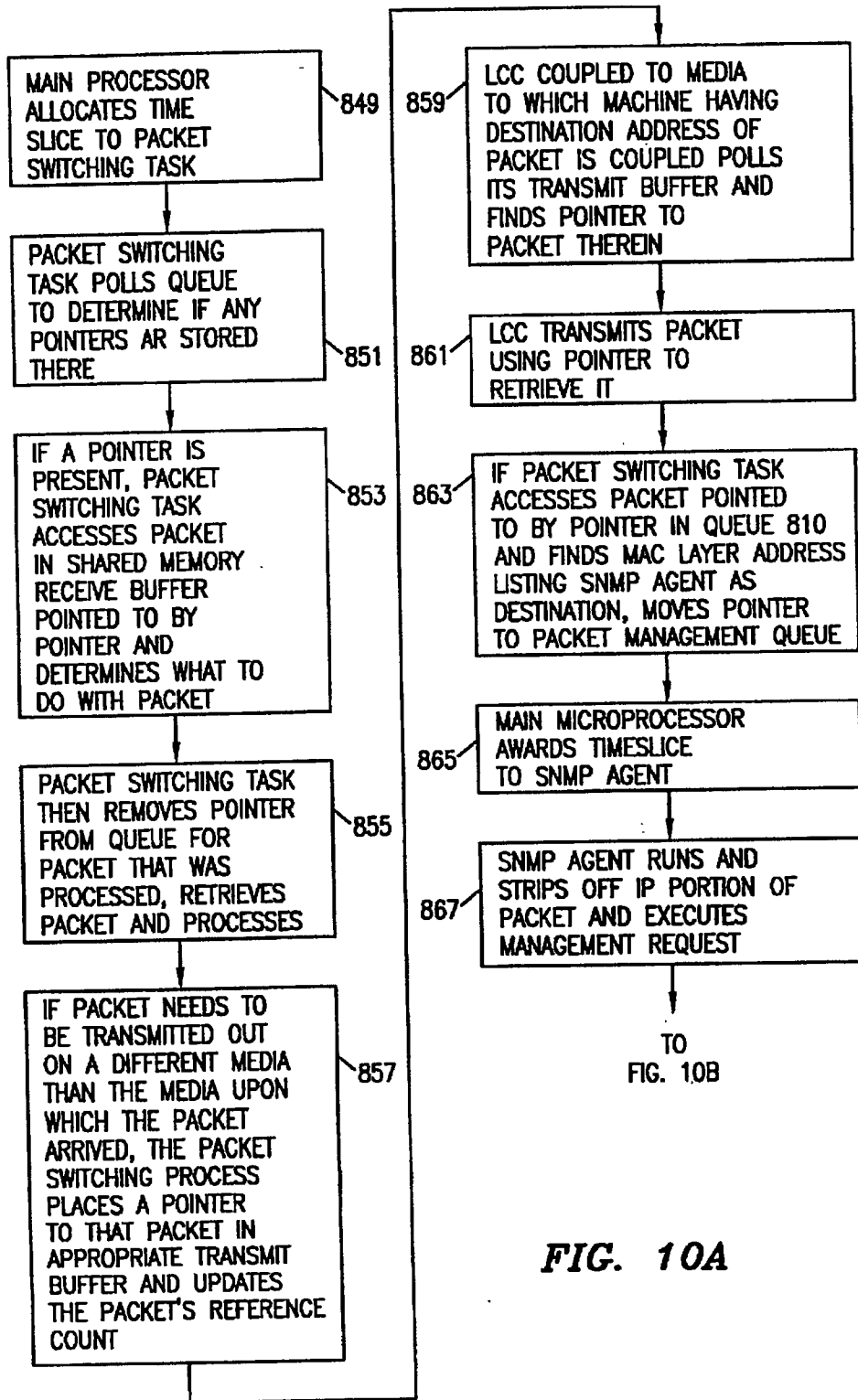


FIG. 10A

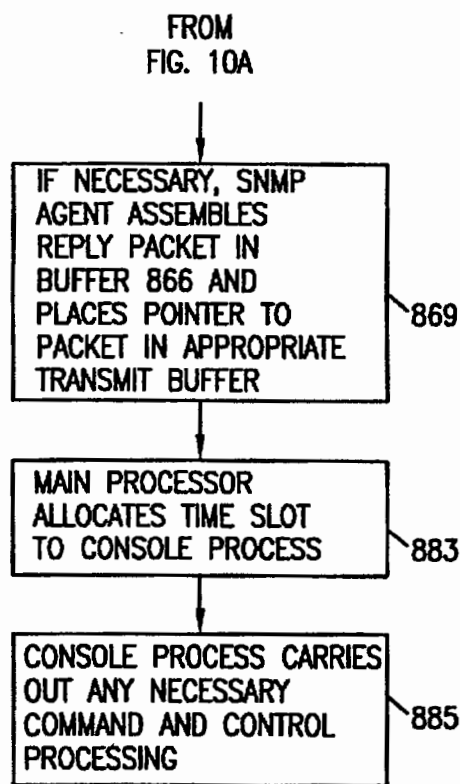


FIG. 10B

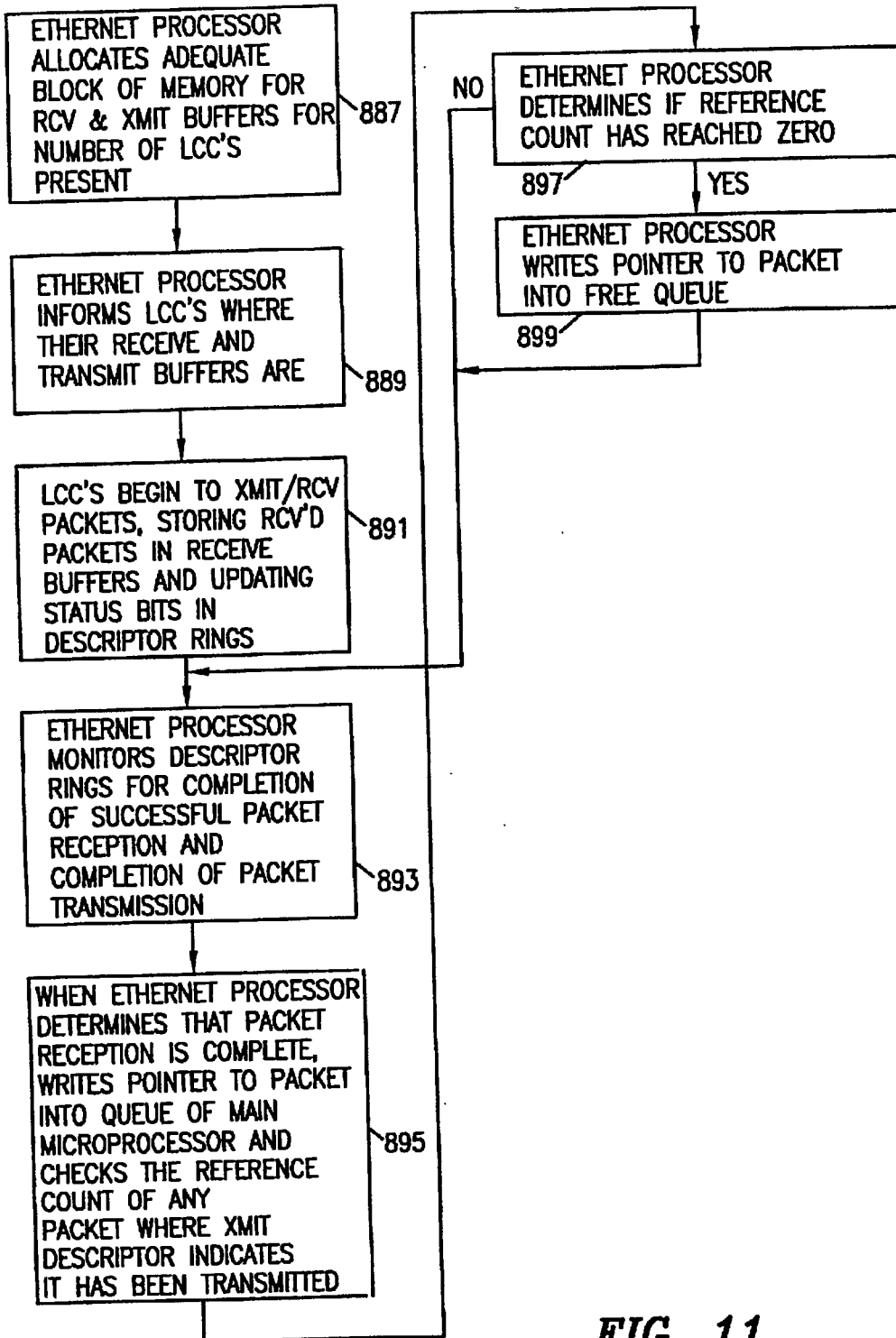


FIG. 11

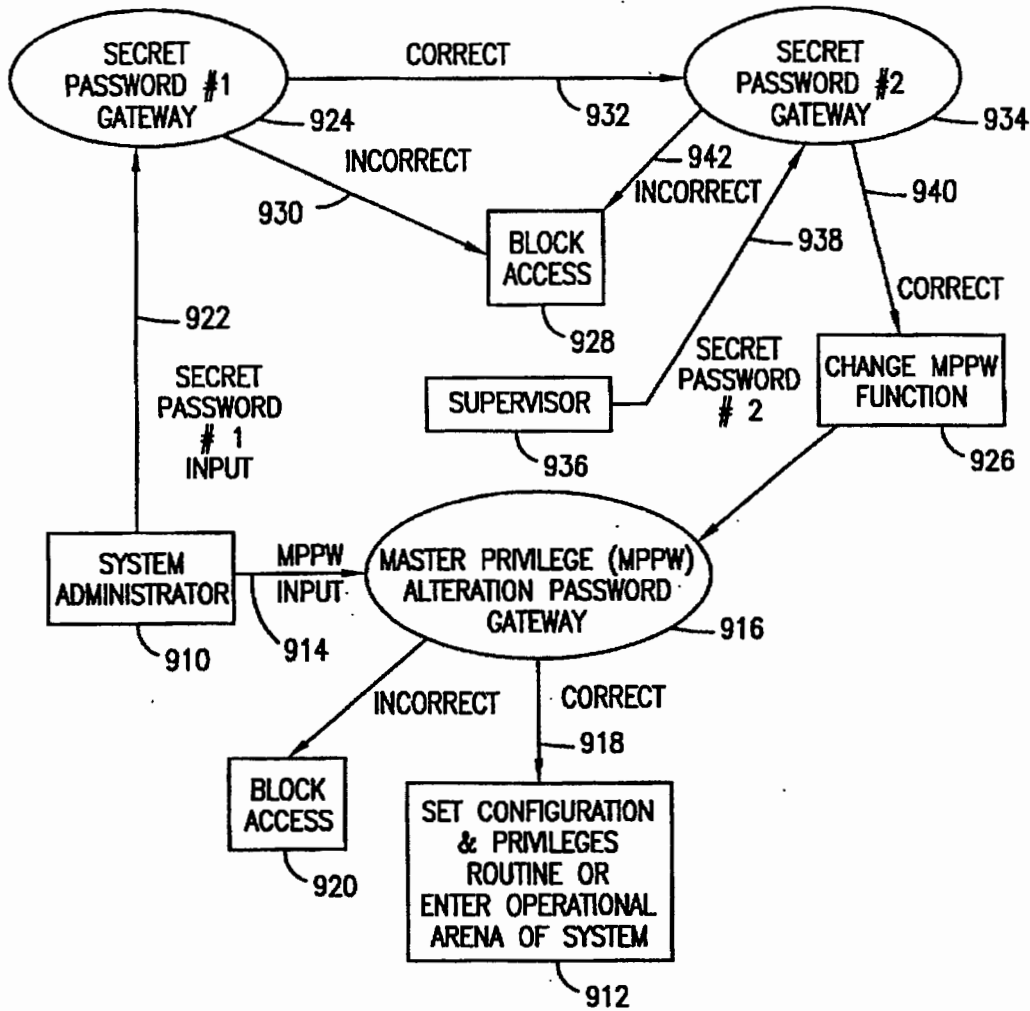


FIG. 12

**NETWORK PACKET SWITCH USING
SHARED MEMORY FOR REPEATING AND
BRIDGING PACKETS AT MEDIA RATE**

BACKGROUND OF THE INVENTION

This application is a division of application Ser. No. 08/694,491 filed Aug. 7, 1996; which is a continuation of application Ser. No. 08/498,116, filed Jul. 5, 1995; which is a CIP of application Ser. No. 07/881,931, filed May 12, 1992, now U.S. Pat. No. 5,432,907. This is a continuation-in-part of a U.S. patent application entitled, NETWORK HUB WITH INTEGRATED BRIDGE, Ser. No. 07/881,931, Filed May 12, 1992 (now allowed). The invention pertains to the field of networks for communications between computers, and, more specifically, to improvements in hubs for such networks.

Networks serve the purpose of connecting many different computers or terminals to each other, host computers, printers, file servers etc. so that expensive computing assets, programs, files and other data may be shared among many users. Communication protocols and standards for networks developed quickly to standardize the way in which data packets were sent across the data exchange media of the network. Several protocols have developed for networks including Ethernet™, Token Ring™, FOIRL and FDDI, the latter two being adapted for fiber optic physical media carrying the signals.

The physical media first used on Ethernet were thick coaxial cables, and a standard called 10Base5 was developed for assuring multi-vendor compatibility between components in thick coax, mix and match networks where network components from different vendors were used. These thick coax lines were bulky, expensive and hard to work with. Later, thinner coax Ethernet was developed, and, as an alternative to coax, unshielded twisted pair wires were used for the physical media. A vendor compatibility standard called 10BaseT developed for twisted pair media.

Networks have their own hardware and software to interface with the physical media that carry the signals, and the network software must interface with the operating system software. Computers communicate with each other using a set of rules called a protocol. A group of protocols, all related to the same model are called a protocol suite. To encourage open systems, a common model called OSI was developed by the International Standards Organization. OSI engendered a protocol suite which allows computers of all sizes and capabilities the world over to communicate using a common set of rules.

The OSI model has seven layers of software, each of which makes different functionality available to computers communicating using this model. Each layer in the model deals with specific computer-communication functions.

The Physical Layer is the lowest layer and specifies the rules for transmission of signals across the physical media. Hubs, also known as repeaters, have multiple connections to this physical media called ports. The purpose of a hub is to receive data packets from one port and repeat these packets, i.e., retransmit them on every other port connected to the hub according to whatever protocol, e.g., Ethernet, etc., which is in use.

The Data Link layer deals with transmission of data between devices on the same network. In addition to describing how a device accesses the physical media, this layer also provides some measure of error detection and control. Local Area Network (LAN) technologies such as Ethernet, Token Ring and FDDI operate at this layer. Data

link addresses are implemented at this layer, and provide each device connected to the network a unique identifier by which packets may be sent to it. Bridges, which are devices which aid in forwarding data packets from one network segment or one network to another, operate at the Data Link layer.

The Network Layer deals with transfer of data between devices on different networks. The Network Layer adds the notion of network addresses which are specific identifiers for each intermediate network between a data source and a destination. Routers, which are devices which assist in transferring data packets from one network to another, operate at the Network Layer.

The remaining layers, called the higher layers, are the Transport Layer, Session Layer, Presentation Layer and Application Layer. These layers deal with communication between message source and message destination. The transport layer manages the transfer of data from a source program to a destination program. Process addresses, which identify specific "processes", i.e., computer programs, are implemented at this layer. Gateways operate at these higher OSI layers.

Within the OSI model, the user presents data through application programs to the highest layer. This data is then passed downward through the hierarchy of layers with each layer adding addressing and/or control information. When the data reaches the physical layer, it is sent to a device.

Conversely, received data is passed up through the layers with each layer stripping address or control information.

One way to think of a protocol is a common language by which computers may communicate, but a more accurate way is as a set of rules by which data is communicated between identical OSI layers.

There are other communication protocols beside the OSI Model. These include TCP/IP, XNS, IPX, AppleTalk, DECnet and SNA. Each of these protocols has its own layer model. For example, TCP/IP collapses network functionality into only 4 layers, while AppleTalk has 6 layers.

All network media have a limitation on the maximum volume of traffic that may be carried based upon the bandwidth imposed by the physical characteristics of the media. Ethernet bandwidth is 10 Megabits/second. This acts a limit on the traffic volume and can limit the number of computers, which may be connected to a single "segment" of a network. A segment is section of a network connected to a group of machines which may communicate with each other via repeater operations without having to traverse a bridge or router. Bridges and routers are useful in that they allow connections of multiple segments such that more computers may communicate with each other than would otherwise be possible given the limited bandwidth of the media.

Each bridge and router requires certain other peripheral circuitry to support it such as LAN controllers, a CPU, a power supply, a network management process, memory to store bridge source and destination address tables and various other things like status registers etc. Likewise, repeaters require many support circuits many of which are the same support circuits needed by bridges and routers. Further, bridges, routers and repeaters or hubs require initialization to set them up for operations, and they require initial installation labor to set them up properly to operate in a particular network configuration. In addition, each type machine is subject to network management considerations, assuming an intelligent hub. An intelligent hub is one which collects statistics about traffic flow through its ports, can electronically turn ports on and off and which provides error correc-

tion and detection services. Intelligent bridges, routers and hubs supply status information upon request from network management processes and can respond to network management commands, such as shut off a particular port.

In the prior art, bridges and routers were separate circuits from hubs and this created needless duplication of many peripheral circuits which were common between hubs and bridges and which could be shared. This needless duplication cost more and provided more points of failure. For example, if the bridge power supply failed or the CPU crashed, all machines on the two network segments on either side of the bridge would be cut off from each other.

Typically, a bridge is connected to a hub by a separate local area network segment which itself requires two port interface circuits such as LAN controllers and AUI's (generic network interfaces) with appropriate port drivers adapted for the specific media used for the bridge-hub LAN segment. This bridge-hub LAN segment represents an additional expense, requires management and provides additional points of failure which could disable the network. An intelligent hub coupled to a bridge or router by a separate LAN segment then requires three different device addresses for management message traffic, and creates more possibility for a network failure in multiplying the number of points of possible failure.

Another drawback of separate bridge/router and hub circuits is that bridge/routers do not usually include a mode where the bridge/routing function can be bypassed. The ability to bypass the bridge/routing function provides flexibility in network growth as small networks do not need bridging functions until the maximum network traffic volume starts to exceed the available network bandwidth. The ability to selectively bypass the bridge/routing function gives a network designer the ability to design a small network which has a built in capacity to grow larger without adding new components and improves the ability to troubleshoot the network.

Integrated hubs and bridges existed as option cards for concentrator chassis at the time this patent application was filed. One example of such a device is the Penril 2530 concentrator card with full performance bridging although it is not currently known whether this device qualifies as prior art because the copyright date of the literature on this device is dated the same month as the filing date of the parent of this patent application. The Penril Module 2530 10baseT concentration and bridging card for the Penril 2500 series concentrator combines a hub and bridge which operates at all times on the same printed circuit board. The design of the Penril 2500 concentrators were for large networks. The 2530 card slides into a card slot on the 2500 series concentrator which can also service a plurality of such cards. The concentrator frame is believed to contain certain shared features such as power supply etc. and has a local, internal LAN segment that couples all the repeater/bridge cards together so that they can send data back and forth between them. The repeater on each card can be coupled to up to 25 machines on the network segment connected to that card and the integrated bridge continuously bridges the network segment coupled to a particular card to the internal LAN segment such that a machine coupled to a LAN segment coupled to card 1 can send a packet to a machine coupled to a LAN segment coupled to card 2 via the bridge on card 1, the internal LAN segment of the concentrator, the bridge on card 2 and the repeater on card 2. No distributed management functionality is integrated on either card 1 or 2. That management functionality is placed on a third card which resides on a different card slot. If the management card

broke, the repeaters and bridges in cards 1 and 2 could not be controlled. Likewise, if the internal LAN broke, user 1 could not send data to user 2 or vice versa.

A concentrator structures like the Penril 2500 series is designed for large networks since to connect two external network segments, two cards are needed each of which can service up to 25 user machines. If the network has only 27 users, such a concentrator represents too big and complex of a structure to be affordable and justifiable for such an application.

Another problem with concentrators such as the Penril 2500 series is their lack of "stackability". The problem is this. Suppose a particular building had 3 users on the ground floor and a group of 20 heavy users on the 4th floor or otherwise spaced away from the 3 users on the ground floor by a distance which is just under the maximum 10BaseT cable run permitted by the applicable Ethernet specification. The use of a concentrator requires that every one of the group of 20 users has his own twisted pair running from his machine back to the concentrator. The same is true for thick and thin coaxial cable installations. Such a configuration can be prohibitively expensive because a great deal of wire or coax must be used and the expense of installing all that wiring through the walls and ceilings can be large. Now suppose that the distance to the group of 20 from the concentrator is larger than the maximum allowable cable run. In such a case, the complex wiring cannot be used, and if those users must be able to share resources with the 3 users on the first floor, another concentrator must be purchased. Concentrators like the Penril are not inexpensive. Typical costs today are in the neighborhood of \$30,000 for the concentrator frame and about \$6000 for each card.

A similar problem arises in large networks in big companies who may, for example, have a branch office in another state with only 6 users. If those users must share data or resources connected to the network at the parent company, they must be on the same network as the users at the parent company. With concentrator technology, the 6 users in the branch office must be connected to the concentrator at the parent company by a wide area network (WAN) connection. The Penril concentrator 2500 series has a card module (the 2540) which implements a WAN interface, but the 6 users in the branch office must also have a concentrator to plug their WAN interface card into. Therefore, the expense of having the tiny 6 user network segment remotely located is greater than it needs to be.

Thus, a need has arisen for an apparatus which can perform the functionality of bridges or routers and hubs without the aforementioned deficiencies, and which can overcome the aforementioned difficulties with concentrator technology in smaller networks or large network will small satellite networks.

SUMMARY OF THE INVENTION

According to a broad teaching of the invention, there is disclosed herein, inter alia, a packet switching machine having shared high-speed memory with multiple ports. One port is coupled to a plurality of LAN controller chips each of which is coupled to its own media access unit and an individual LAN segment. The port coupled to the LAN controllers is also coupled to an Ethernet processor that serves to set up, manage and monitor a receive buffer having enough space to store packets received by all the LAN controller chips. The Ethernet process also sets up and manages a transmit buffer for each LAN controller chip and sets up and monitors a descriptor ring which stores status

data maintained by the LAN controller chips and pointers to the transmit and receive buffer portions of the shared memory.

When a LAN controller receives a packet, the packet is stored in the receive buffer in shared memory, and a pointer to that packet is written into the receive portion of the portion of the descriptor ring devoted to that LAN controller. The LAN controller sets a status bit in the receive portion of the portion of the descriptor ring that is devoted to that LAN controller when packet reception starts indicating that a packet is being received. After packet reception is complete and error detection has been done and the packet is deemed to be correct, the LAN controller sets another bit in the receive portion of the portion of the descriptor ring that is devoted to that LAN controller indicating that the packet has been correctly received.

The Ethernet process monitors status bits set in the descriptor ring by the LAN controller chips that indicate when a packet has been successfully received, and, when this event occurs, reads the pointer to the packet from the descriptor ring and transfers the pointer to a queue which is monitored by a main processor coupled to another port of the shared memory. The main processor is coupled to another port of the memory and monitors its queue for the presence of pointers. When a pointer to a received packet is found, the main processor accesses the packet and determines from the packet's address data what to do with the packet. If the packet is addressed to a machine coupled to the media segment of a different LAN controller than the LAN controller that received the packet, the main processor writes a pointer to the packet into the transmit buffer of the LAN controller coupled to the media segment on which the packet is to be transmitted. If the packet is a management packet, a pointer to the packet is written into a management queue which is monitored by an SNMP agent so as to forward the packet to the SNMP agent for processing. The SNMP agent and the packet switching tasks are time division multiplexed with a console process by an operating system kernel.

The main microprocessor and the Ethernet processor coordinate to manage the utilization of storage locations in the shared memory. When the main microprocessor writes a pointer to a packet into one or more transmit buffers, it also accesses a reference count in a predetermined field in the packet stored in the receive buffer and writes a number therein indicating the number of LAN controllers that are scheduled to transmit the packet. The LAN controllers also write status bits into transmit portions of the descriptor record in the portion of the descriptor ring devoted to that LAN controller. The Ethernet processor monitors the transmit portions of the descriptor ring. When the Ethernet processor determines that a status bit for a particular LAN controller indicates that the LAN controller has successfully transmitted a packet, the Ethernet processor accesses the reference count field in the packet and decrements the reference count. When the reference count reaches zero, the Ethernet processor writes a pointer to the storage location in which that packet is stored in the receive buffer into a Free Queue indicating that the storage locations currently occupied by the packet are free to be used to store other incoming packets.

Another port of the shared memory is coupled to an uplink interface to higher speed backbone media such as FDDI, ATM etc. The main microprocessor can forward packets to these interfaces by writing pointers into transmit buffers dedicated to these interfaces in the shared memory, and received packets are written into the receive buffer as if they were received by a LAN controller.

In some embodiments, another port of the shared memory is coupled to an expansion interface having another microprocessor which serves to load share with the Ethernet processor and the main processor to achieve higher speed operation.

Speeds up to media rate are achieved by only moving pointers to packets around in memory as opposed to the data of the packets itself.

A double password security feature is also implemented in some embodiments to prevent accidental or intentional tampering with system configuration settings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a typical network environment in which the teachings of the invention find utility.

FIG. 2 is a block diagram of one embodiment of the invention employing the broad concept of integration of a bridge with a hub in the same package to share circuitry and eliminate points of failure which would exist if the bridge and hub were separate circuits.

FIG. 3 is a block diagram of another embodiment of the invention with dual network two transceivers for fault tolerance.

FIG. 4 is a data flow diagram illustrating the three software processes that are executed in the preferred embodiment, to perform bridging, in-band management and out-of-band management functions.

FIGS. 5A and 5B are a flow diagram of the processing of the bridge process illustrating operation of the forwarding vectors.

FIGS. 6A and 6B are a block diagram of the circuitry of the preferred embodiment.

FIG. 7 is a block diagram illustrating an embodiment of a packet switching network hub.

FIG. 8 is a block diagram illustrating a species of machines built in accordance with an operating in accordance with the present invention.

FIG. 9 is a block diagram illustrating the process carried out according to the present invention.

FIG. 10a is a more detailed block diagram illustrating the operation of FIG. 9.

FIG. 10b is a block diagram illustrating the process carried out by the main microprocessor shown in FIG. 9.

FIG. 11 is a block diagram illustrating a flow chart of the process carried out by the Ethernet processor of FIG. 9.

FIG. 12 is a block diagram illustrating an embodiment of the present invention utilizing a dual password security arrangement.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT:

Referring to FIG. 1 there is shown a typical network installation in which the teachings of the invention find use. A redundant power supply 10 supplies a fiber optic hub 12 which has a plurality of fiber optic ports indicated generally at 14. Each of these ports is connected to a fiber optic physical data transmission media via a port driver circuit not shown. Each of the fiber optic media is indicated by a line with three slash marks through it. These media are coupled to mainframe computers 16 and 18, laser printer 20 and three personal computers 22, 24 and 26. Data transmitted to the fiber optic hub 12 by any of the computers is automatically repeated by repeater circuitry in the hub on all the other ports using the FOIRL Ethernet standard.

The fiber optic hub 12 is connected via a backbone port connection 28 to a 10Base2 hub with integrated bridge 30. The fiber optic hub also has another port serving as a backbone connection 32 to a 10BaseT hub 34 with integrated high performance bridge/router and wide area network (WAN) interface 36. The wide area network interface can span great distances. In the example shown, the wide area network interface 36 couples the 10BaseT hub 34 to another 10BaseT hub 38 with an integrated high performance bridge and wide area network interface. The hub with integrated bridge represents a significant advantage in that the presence of bridges and routers in complex, high volume networks provides segmentation of the network so as to maximize use of the media by allowing maximum traffic volume, i.e., volume at media rate, e.g., 10 megabits/second for Ethernet, on more segments of the network without violating the maximum Ethernet specification limit of 4 repeaters between devices. Since bridges and repeaters require many of the same support circuits, it is advantageous to combine a bridge and a hub into the same circuit so as to share these support circuits. Such a combined hub/bridge reduces the cost, complexity and points of failure. Such a combined circuit also eliminates the bridge to hub LAN segment where the bridge and hub are separate. This also eliminates the IP address of this segment and all management burden thereof.

Coupling of portions of a LAN by a bridge also allows the segments on opposite sides of the bridge to use different communication protocols.

Also, in some embodiments, the bridge can be a router, and any known routing or bridging process is within the teachings of the invention.

Another advantage of a combined hub and bridge is the stackability of the architecture as compared to concentrators like those manufactured by Penril. When a user is out of card slots in a concentrator, that user will have to buy an entirely new concentrator (concentrators are very expensive) even if there is only one small group of users who cannot fit into the repeater cards on the first concentrator. Another disadvantage of concentrators with bridge cards, repeater cards and management cards, is that the management is not integrated. If the management card fails, the bridge and repeater cards are not manageable.

The 10Base2 hub 30 is connected to a number of computers of which computers 40 and 42 are typical. These connections are via coaxial line segments 44 and 46. Coaxial connections are shown in FIG. 1 by lines with two slash marks through them. The 10Base2 hub 30 is also connected via a coaxial backbone connection 48 to a 10BaseT hub with integrated bridge 50.

The 10BaseT hub 50 is connected via a plurality of repeater ports 56 to a plurality of computers of which computers 52 and 54 are typical. Any data packet entering the hub 50 from any one of the ports is automatically repeated on all the other repeater ports 56. The same type of repeating operation is carried out automatically by all of hubs 12, 30, 34, 38, 66 and 72.

A 10BaseT hub uses a physical layer communication protocol which is appropriate for a twisted pair of physical media. Twisted pair connections are shown in FIG. 1 by lines with single slashes through them. A 10Base2 hub repeats data packets on its ports using a physical layer protocol appropriate to coaxial cable.

The 10BaseT hub 34 has a plurality of repeater ports connected to a plurality of computers of which device 62 is typical. Hub 34 also has a twisted pair port connection 64 to

another 10BaseT hub 66 which has an integrated bridge. Connection 64 is a backbone connection for hub 66. Hub 66 is connected to a plurality of computers of which computer 67 is typical via repeater ports 68.

Likewise, hub 34 is connected via a twisted pair port connection 70 to the backbone port of another 10BaseT hub with integrated bridge 72. The hub/bridge 72 is connected to a plurality of computers via repeater ports 74.

As an example of how the integrated hub bridge circuits in FIG. 1 work, consider the following hypothetical data exchange transactions. Suppose that computer 52 wishes to send a data packet to computer 54. In this example, the data packet would enter the 10BaseT hub/bridge 50 via twisted pair line 80 and would be automatically repeated on all the repeater ports 56 including twisted pair line 82. Computer 54 would receive the packet as would all the other computers connected to hub/bridge 50. However, the packet would have a destination address indicating device 54 was the intended recipient such that other computers connected to the hub/bridge 50 would discard the packet.

In the preceding example, the bridge function in hub/bridge 50 would examine the destination address of the packet arriving via twisted pair 80 and check a forwarding table of network addresses which contains entries for various network addresses indicating whether those addresses are on network 1 or network 2. In the bridge mode of operation for hub/bridge 50, all of the repeater ports 56 are considered to be network 1 and the backbone connection 48 is considered to be network 2. The bridging function, in the preferred embodiment, is a learning bridge which builds the forwarding table as data packets arrive at the bridge from each source address. The bridging function knows which network a packet came from, and will make an entry in its table associating each source address with the network from which it came. Assuming that computer 54 had already sent a packet somewhere else, the bridging function would know that computer 54 was connected to network 1 and therefore would not forward the packet received from computer 52 to the network 2 via backbone connection 48. However, in the situation where computer 54 had not previously sent a packet, the bridging function in hub/bridge 50 would assume that computer 54 was connected to network 2, and would forward the packet to network 2 via backbone connection 48. However, since the packet would be automatically repeated on all repeater ports 56 anyway, computer 54 would still receive the packet via its repeater port even though the packet was also forwarded to network 2. Since computer 54 would send an acknowledgment message acknowledging receipt of the packet, the bridge function in hub/bridge 50 would then make an entry in its table indicating that computer 54 was coupled to network 1. Thereafter, further packets destined for computer 54 would not be forwarded by the bridge in hub/bridge 50 on the backbone 48.

Now suppose computer 52 wishes to send a packet to computer 42. In this case, the bridge function in hub/bridge 50 would not find an entry for computer 42 and would forward the packet received from network 1 via twisted pair 80 out on the coaxial backbone connection 48.

The backbone connection 48 for hub/bridge 50 is connected to a repeater port of 10Base2 hub/bridge 30. Therefore the packet arriving on coaxial line 48 is automatically repeated on coaxial lines 44 and 46, and would therefore arrive at computer 42.

To change the hypothetical slightly, suppose computer 52 wanted to send a packet to computer 26 connected to fiber optic hub 12. In this case, the bridging functions in hub/

bridge 30 would read the destination address and may or may not find an entry for computer 26. In either eventuality, the bridge 30 would forward the packet received on coaxial line 48 out on fiber optic backbone connection 28. This backbone connection 28 is connected to one of the repeater ports of the fiber optic hub 12 and therefore would be repeated on all other repeater ports 14 thereof. In this manner the packet would be transmitted out on the repeater port connected to fiber optic media 86 and would arrive computer 26.

One of the advantages of integration of the hubs and bridges in the sample network of FIG. 1 is that it substantially reduces the cost of the network. This is, in part, because the hub/bridge integration eliminates much circuitry needed to couple each hub to a bridge with the associated LAN controllers and transceivers needed to do this. Network management traffic is also reduced because there are fewer network addresses of machines which must be addressed by network manager traffic. Typically a network manager will be coupled to one of the hub/bridges by a terminal and will address management commands to any of the network implementing circuits on the network. These commands will be forwarded as data packets to the proper hub/bridge etc. like other data packets, but will be taken off the network by the machine to which they are addressed and executed. By having fewer boxes that need to be managed and fewer addresses, this management traffic, which represents network overhead, is reduced.

FIG. 2 shows a highly simplified block diagram of an embodiment of the broad concept according to the teachings of the invention. The hub 140 and bridge process 142 are integrated in the same system and are supported by the same physical support structure and housed in the same housing. The hub 140 is connected to a plurality of individual transceiver lines shown collectively at 166. In addition, the hub and bridge functions share certain physical assets such as the CPU 144 and the memory 146. In a sense, the LAN 2 interface 148 is also shared, because in the bypass mode switch 150 is connected so as to couple an AUI port 152 of the hub 140 to LAN 2 through the LAN 2 interface 148. In bypass mode, LAN 1 and LAN 2 comprise a single local area network. Because the AUI port of the hub 140 cannot drive any physical media, the LAN 2 interface 148 is necessary to merge the machines coupled to LANs 1 and 2 into a single network even though the physical media of LAN 1 and LAN 2 may be different.

In the preferred embodiment, the bridge process 142 is a software process carried out by the central processing unit 144 using memory 146 and the bridging software routine described in flow charts given below and detailed in the source code appendix attached hereto. In other embodiments, the bridge 142 may be a hardware circuit which operates either autonomously or under the control of central processing unit 144. In either type embodiment, the hub and bridge functions will share the central processing unit and will be managed by the CPU implementing network management functions.

Bypass mode is useful for providing flexibility in network designs. It is most useful in planning for network growth where local area networks 1 and 2 may be connected together as single network when the level of network traffic is small enough such that the bandwidth limitations of the physical media do not impose a ceiling on the number of machines which may be connected. However, when the number of machines coupled to the network grows and the volume of traffic approaches 10 Megabits per second, the CPU 144 in FIG. 2 can alter the state of switch 150 such that

the AUI port 152 is no longer coupled to bus 160 directly and bridge mode becomes active. When the bridge is active, because only traffic on bus 162 which has a destination address identifying a machine connected to local area network 2 will get through to LAN 2, the number of machines effectively sharing each network is substantially cut down. Thus, the amount of traffic on each network is cut down to a level which can be easily handled by the physical media.

Referring to FIG. 3, there is shown a block diagram of another embodiment of an integrated hub/bridge with redundant network two transceivers. A repeater/controller 90 has a plurality of repeater ports 92 each of which is coupled to a hub interface circuit such as the port 1 transceiver circuit 94, the port 2 transceiver circuit 96 or the port 24 transceiver circuit 98. Each of these port transceiver circuits interfaces between the network data link layer protocol implemented by the repeater/controller 90 and the particular physical layer protocol appropriate to the physical media being used to carry the data. The physical media is represented by lines 100, 102 and 104. The physical media may be unshielded twisted pair in the case of a 10BaseT hub, coaxial cable in the case of a 10Base2 hub, or fiber optic wave guides in the case of a 10BaseF or an FOIRL hub, etc. All of the examples given above are for the Ethernet network data link layer protocol, however the teachings of the invention are applicable to any network data link layer or physical layer protocol such as Token Ring, FDDI, etc. Further, the teachings of the invention are equally applicable to any communication model such as OSI and any transport layer protocol such as TCP/IP, XNS, IPX, AppleTalk, DECnet, and SNA.

Any data received through any one of the port interface circuits such as port 1 circuit 94 is automatically repeated by the repeater/controller 90 on all of the other ports 92. In addition, any received data packet is also repeated out an AUI port 106 and is also transmitted on a network one data bus 108 coupled to a LAN 1 controller 110. The AUI port 106 is a non-media specific port which can be coupled to a transceiver circuit which is appropriate to the particular physical media to be driven. The format of data packets and the collection of signal lines and signal definitions for an AUI port is set by a national standard in accordance with the particular communication data link layer protocol being used. However, the AUI port itself can drive a 50 meter AUI transceiver cable, but cannot drive the physical media of the network without a suitable network interface transceiver.

The AUI port 106 plays an important role in implementing a novel feature of some embodiments of the invention called bypass mode. In bypass mode the bridging function is bypassed, and the backbone port is treated as just another repeater port. In FIG. 2 this is physically implemented by connecting the AUI port 106 to a software controlled switch 112 in bypass mode. Switch 112 is set in bypass mode so as to connect terminal 114 to line 116. Line 116 can be coupled to the data input/output port of either of two LAN interfaces 118 and 120 through another switch 122. Switches 112 and 122 may be software controlled in some embodiments and manually operated in other embodiments.

The function of the selected LAN interface 118 or 120 is to drive whatever physical media is used for the backbone port connection 124 to network two. This backbone port physical media may be twisted pair, coaxial cable, fiber optic waveguide, etc. The purpose of having two LAN interfaces A and B is to provide fault tolerance redundancy such that if one fails, the other may be used. Both of the switches 112 and 122 are controlled by a microprocessor 126 in the preferred embodiment. This microprocessor is shared by all of the circuitry in the integrated hub/bridge 130. Normally,

11

the microprocessor 126 will establish the position of software controlled switch 112 during an initialization phase at power-up time.

During initialization, data is written via data bus 127 to the repeater/controller 90 to set this device up for operation. The microprocessor 126 also reads data written by the user (or a front panel switch position in some embodiments) to determine whether bridge or bypass mode is desired. If bypass mode is desired, microprocessor 126 send a control signal to switch 112 so that terminal 114 is connected to line 116. If bridge mode is desired, switch 112 is controlled such that terminal 132 is coupled to line 116. Terminal 132 is coupled to a LAN 2 controller 180 which is driven by the bridge function carried out in software by microprocessor 126. LAN 2 controller is the network two interface for the integrated bridge.

The microprocessor 126 has multiple duties including: being shared by both the hub and bridge processes for initialization, on-line, in-band management and carrying out bridging duties in some embodiments although the bridge function could be carried out by separate circuitry in some embodiments. The microprocessor is only indirectly involved in the hub process since the repeater/controller 90 does the retransmission work without intervention by the microprocessor. The microprocessor can intervene in this process in executing management commands such as turning ports on or off and will report certain status data back to the network manager such as port polarity status, per-port error rate etc.. In the bridge function however, the microprocessor plays a central role in executing the software that carries out the forwarding function. In most embodiments, the in-band management process runs in background while the bridge process runs in foreground.

In the bridging mode, data packets will be forwarded from local area network 1, 140, to local area network 2, 124, where appropriate, while in the bypass mode, local area network 1 and local area network 2 will be merged and will all be considered the same local area network by the hub.

The bridging function is carried out in the embodiment of FIG. 3 as follows. When a packet arrives from local area network 1, it is repeated on network one data bus 108 and received by LAN 1 controller 110. The LAN 1 controller 110 then cooperates with a DMA controller 172 to store the data packet in a receive buffer for network one in memory 170. At initialization time, the LAN controllers 110 and 180 are informed by the microprocessor 126 of the starting and ending memory locations of receive and transmit FIFO buffers in memory 170 for each of networks one and two. In some embodiments, the receive and transmit buffers are implemented as FIFO buffers using linked lists in memory 170. In other embodiments, separate FIFO memories could be used.

Since the microprocessor 126 also uses memory 170 to store the forwarding table entries for the bridging function, the data, address and control buses of the memory 170 must be shared between the DMA controller 172 and microprocessor 126. The details of how this bus sharing is carried out are not critical to the invention and any bus arbitration scheme will suffice for practicing the invention. In the preferred embodiment, when the LAN controllers receive packets, they request the DMA controller to store them, and the DMA controller requests bus arbitration PAL (programmable array logic) for access to the bus. If bus access is not immediately granted, the local area network controllers 110 and 180 can temporarily store data packets in internal buffers. When a data packet arrives and is stored in

12

the receive buffer, an interrupt to the microprocessor is generated by the LAN controller which received the packet. This tells the microprocessor which network is the source of the packet and that the bridge process detailed below must be performed. The flow chart of FIGS. 5A and 5B below are the processing of an interrupt service routine which services the LAN controller interrupts in some embodiments.

The microprocessor 126 processes received data to be input to the bridging process by accessing memory 170 using a pointer to the received packet sent with the interrupt from the LAN controller. The microprocessor reads the destination address and consults the forwarding table. If the packet is to be forwarded, the microprocessor "deposits" the data packet in the transmit buffer corresponding to the appropriate network by rearranging the pointers on the linked list of that transmit buffer to point to the new data packet to be transmitted in sequence. The LAN controllers are continually requesting access to the memory buses through the DMA controller 172 and the arbitration PAL 196 to retrieve data packets from their respective transmit buffers. In the case of LAN controller 110, such packets are forwarded to the repeater/controller 90 via data bus 108 for repeating on all network one ports. In the case of the LAN 2 controller 180, these data packets are forwarded to the LAN 2 interface circuit 118 or 120 selected by switch 122 for transmission on the network 2 media.

The local area network controllers 110 and 180 manage pointers for their FIFO buffers so as to keep track of the addresses where the last message stored in the receive queue is located and the address of the next packet to be transmitted in the transmit queue and to keep the linked lists properly FIFO ordered.

Microprocessor 126 also establishes a management queue in the memory 170 where in-band management commands and requests are stored temporarily until a management process, running in background mode, can access and execute the desired function.

Arbitration logic 196 is used to grant access to the memory buses according to some appropriate access protocol. In some embodiments, the protocol might be first-come, first-served, while in other embodiments the access protocol may use some priority scheme as between the DMA device 172 and the microprocessor 126.

A multiplexer 186 under control of the arbitration PAL 196 selectively connects the address bus 188 of the memory 170 either to the address/control bus 190 of the DMA device 172 or the address/control bus 192 of the microprocessor 126 in accordance with a control signal on line 194. The arbitration logic 196 also generates the row address strobe and column address strobe signal (RAS*/CAS*) on line 198 so as to time division multiplex the bus 188 between 10 bits of row address and 10 bits of column address. The arbitration logic 196 is coupled to the microprocessor 126 address and control bus to receive input information by a connection not shown in FIG. 2.

Arbitration of the memory data bus is carried out through tri-state buffers 200 and 202. Tri-state buffer 200 selectively connects the data bus 204 of the DMA device to the DRAM memory data inputs 128 coupled to the LAN controller data outputs when a chip select signal on line 206 is true. Likewise, tri-state buffer 202 couples the data bus 127 of the microprocessor to the memory data inputs when a chip select signal on line 208 is true. These chip select signals are generated in some embodiments by an address decoder gate array 197 coupled to the microprocessor address bus. In other embodiment, they may be generated by arbitration/

PAL logic 196 so as to control and arbitrate access to the DRAM data inputs as between the DMA device 172 and the microprocessor 126.

As in the case of the embodiment of FIG. 2, microprocessor 126 is shared by the hub function and the bridge function. Specifically, the microprocessor sends data to the repeater/controller circuit 90 at initialization time to set the circuit up for operation, sends data to it during operation to do certain things like turn ports on or off and receives data from the repeater/controller regarding status for purposes of replying to management inquiries regarding port status. Data is sent to and received from the repeater/controller 90 via the data bus 127 using a tri-state buffer 210. This tri-state buffer receives a chip select signal on line 312 generated by address decoder 197 or arbitration/PAL control logic 196. The address decoder or arbitration logic 196 also generates a chip select signal for a tri-state buffer 214 which gates the address/control bus 192 on the microprocessor through to the address and control inputs 216 of the repeater/controller circuit 90. Once the microprocessor has been granted control of these buses, data may be sent to the repeater/controller 90 to initialize it or to cause it to carry out certain management functions, or to receive status information therefrom in some embodiments. In the preferred embodiment, status information travelling from the repeater/controller 90 to the microprocessor is sent by the repeater through the LAN 1 controller.

In some embodiments, the bus arbitration logic may be eliminated altogether and separate memory circuits may be used for all entities which need random access memory.

Referring to FIG. 4, there is shown a data flow diagram showing the data paths which exist between the three ongoing software processes of the preferred embodiment and several hardware and data structures which are involved therewith. In the preferred embodiment, a bridge process 260 is carried out in software in the foreground mode. As described above, the bridge process receives data from and sends data to a LAN 1 controller 262 via FIFO receive and transmit buffers in random access memory (not shown). This process is symbolized by arrows 263. Likewise, the bridge process sends data to and receives data from a LAN 2 controller 264 in a similar manner as symbolized by arrows 266. When the bridge process is active, i.e., when the hub/bridge is not in bypass mode, a bridge database 268 in random access memory is consulted for each incoming data packet. The purpose of consulting the bridge database is to determine whether or not the data packet should be forwarded to the other network controller. The bridge process will forward the data packet to another network controller other than the network controller from which the data packet was received if the bridge database 268 contains an entry indicating that the machine having the destination address of the data packet is not coupled to the network driven by the controller from which the data packet originated. If there is only one other network serviced by the bridge, the bridge process will forward the data packet to the network controller driving that other network. However, if the bridge process serves more than two networks, the bridge process will consult the bridge database to determine which network is coupled to the machine having the destination address of the data packet and forward the data packet to the appropriate network controller driving that network. Further, the bridge process will forward the data packet to another network controller if there is no entry in the bridge database indicating where the destination address lies.

The bridge database is built anew each time the machine is powered up. Therefore, while the bridge database is

building, more packets will be forwarded to the other network controllers than are actually necessary until the bridge database contains entries for substantially all the destination addresses on the network serviced by the bridge process. However, in most protocols, each destination machine issues an acknowledgment message after it receives a data packet, and these acknowledgement messages will cause entries to be made in the bridge database if an entry for the source network address does not already exist. Therefore, an address will be put in the bridge database when the machine having that destination address either sends or receives the packet.

In some embodiments, the bridge database can be stored in non-volatile memory such as non-volatile RAM and the bridge process can cross-check the accuracy of the bridge database each time a packet is handled. That is, when a packet is received, the bridge database is checked for an entry for the destination address, and the packet will be forwarded if appropriate according to the above noted rules. However, if the acknowledgement message comes back through a different network controller than the network controller to which the packet was forwarded, the bridge process will realize that the machine having that destination address has been physically relocated to a different network segment, and will correct the bridge database entry for that destination address.

There are three basic types of bridges and various types of router processes known in the prior art. Any of these known bridge or router machines or software processes that carry out bridging or routing processes, when integrated with a hub so as to share certain common circuit elements are within the scope of the teachings of the invention. That is, the details of the bridge or routing process are not critical to the invention. Any known bridge or routing machine or software process will suffice.

All bridges provide network connections at the data link layer in the OSI model. The first type of bridge is a transparent bridge. This bridging function provides network connection to local area networks that employ identical protocols at the data link and physical layers. A transparent type bridge places no burden on the physical devices which are attempting to communicate. These devices take no part in the route discovery or selection process. From the device's point of view, it appears that all devices are resident on a single extended network with each device identified by a unique address. Processing by a transparent bridge can be summarized as follows:

- (1) the bridge reads the data link layer destination addresses of all messages transmitted by devices on LAN 1;
- (2) the bridge ignores all messages addressed to devices on LAN 1;
- (3) the bridge accepts all messages addressed to devices on LAN 2, and, in the physical layer and data link layer protocols common to both networks, relays these messages to LAN 2
- (4) the bridge performs identical functions for all messages transmitted on LAN 2.

Obviously such processing requires that the bridge acquires some knowledge of the location of devices. All this information could be manually configured in some embodiments, but in the preferred embodiment, a learning function is used to acquire device addresses.

The bridge learns addresses by reading the data link source address of each message that it receives.

In some embodiments, the forwarding table entries include a timer value that indicates the age of the observation.

The translating bridge is a specialized form of transparent bridge. This type bridge provides network connection ser-

vices to local area networks that employ different protocols at physical and data link layers. For example a translating bridge would be used between a token ring protocol local area network and Ethernet protocol local area network.

A translating bridge provides connection services by manipulating the "envelopes" associated with each type of local area network. Processing performed by a translating bridge is relatively straightforward because the Ethernet, Token Ring and FDDI envelopes are somewhat similar. Each local area network type, however, sends message of different lengths. Because a translating bridge cannot fragment messages, each local area network device must be configured to transmit messages of the supportable length. For the example of a translating bridge being used between Token Ring and Ethernet networks, translating bridge processing can be summarized as follows:

1. The bridge, using the physical and data link layer protocols employed by LAN 1 (the Token Ring protocol), reads the data link layer destination addresses of all messages transmitted by devices on LAN 1.
2. The bridge ignores all messages addressed to devices on LAN 1.
3. The bridge accepts all messages addressed to devices on LAN 2 (the Ethernet protocol), and, using the physical data link protocols employed by LAN 2 relays these messages to LAN 2.
4. The bridge performs identical functions for all messages transmitted on LAN 2.

The second bridge type, an encapsulating bridge, is generally associated with so-called "backbone" topologies. In such a topology, several local area networks will be coupled by several bridges to a high volume backbone of such as a fiber optic FDDI protocol. A typical example of such a topology would be for Ethernet local area networks linked together by a high speed FDDI backbone. Each local area Ethernet network would be connected by an encapsulating bridge to the FDDI backbone. This would be necessary because the inter-network connection (the backbone) is coupled to networks that uses different physical and data link layer protocols.

Unlike translating bridges which manipulate the actual message envelope, encapsulating bridges place received messages within a backbone specific envelope (thus, the term encapsulating) and forward the encapsulated message to other bridges for eventual delivery to the message recipient. In the following example, four Ethernet networks are coupled to an FDDI backbone by four encapsulating bridges. The four bridges coupled to Ethernet networks 1 through 4 will all be referred to as bridge 1 through bridge 4. In the foregoing example, a message from a device on local area network 1 intended for a device on local area network 2 will be processed by an encapsulating bridge as follows:

1. The bridge coupled to local area network 1, using the physical and data link layer protocols employed by network 1 (Ethernet), reads the data link layer destination addresses of all messages transmitted by devices on network 1.
2. Bridge 1 ignores all messages addressed to devices on local area network 1.
3. Bridge 1 accepts all messages addressed to devices on other local area networks, places these messages within an FDDI specific envelope addressed to all bridges (such a collective address is called a multicast address), and sends this envelope across the FDDI backbone.
4. Bridge 3 receives the message, removes the outer envelope and checks the destination data link address. As the

destination address is not local (the destination address is a device coupled to local area network 2) bridge 3 ignores the message.

5. Bridge 2 receives the message, removes the outer envelope and checks the destination data link address. As the address is local, bridge 2 uses Ethernet physical and data link layer protocol to forward the message to the destination device.
6. Bridge 4 receives the message, removes the outer envelope and checks the destination data link address. As the address is not local, bridge 4 ignores the message.
7. Bridge 1 strips the encapsulated message from the FDDI backbone.

The third type of bridge is called a source routing bridge.

This term was coined by IBM to describe a method of bridging frames across Token Ring networks. Source routing requires that the message source (not the bridge) supply the information needed to deliver a message to its intended recipient.

Within a source routing network, bridges need not maintain forwarding tables. Rather they make the decision to forward or to drop a message solely on the basis of data contained within the message envelope. To implement such a scheme, each routing device determines the route to a destination through a process called route discovery. Route discovery can be accomplished in several ways.

One way is to implement a route discovery process using so-called "explorer packets." Each explorer packet has a unique envelope which is recognized by all the source routing bridges in a particular network configuration. When a device coupled to one local area network wishes to send a message to a device coupled to another local area network, the source device sends out an explorer packet which reaches one or more of the source routing bridges. Each source routing bridge adds its own name and the network connection from which the explorer packet was received in a section of the message envelope called the routing information field. Each source routing bridge then floods all of its network connections with copies of the packet.

Ultimately, the destination machine receives multiple copies of the explorer packet each of which has taken a different route through the network configuration. The route that each packet took can be traced from the information in the routing information field of each explorer packet.

The recipient machine then picks one of the packets for use either randomly or according to some criteria which is not critical to the invention such as the most direct route, and sends a response message back to the originator which lists the specific route to be used in communicating with that device. The source device then receives this message and records the route to be used in communicating with the destination device in a memory which stores routing information for each device for which a route has been discovered. Subsequent messages are enclosed in a different type of envelope which is recognized by source routing bridges. These bridges then consult their routing tables for the list of connections in bridges and forward the message based upon the routing information stored in memory.

Routers are different from bridges in that routers connect devices at the network layer of the OSI model. The connected networks may have different protocols at the data link and the physical layers. Routers actively select paths to use in connecting one device to another based on certain factors such as transmission costs, network congestion, transit delay or distance between the source and destination. Distance is usually measured in terms of the number of routers that must be traversed between the source and the destination. Routers

are not transparent in that the devices which wish to use the services of a router must address their messages directly to the router.

Each local area network has a unique local area network address which is resident in the network layer of the OSI model. Likewise, each device on a local area network has its own address which is unique to that local area network. This is the data link layer address in the OSI model. A complete device address then in a routing environment will be either the addition or concatenation of the network layer and data link layer addresses.

Each source device, after preparing a message packet, compares the source address with the destination address and recognizes whether or not the message can be sent directly to the recipient on the network segment to which the source device is connected or whether the message must be routed. If the message must be routed, it is placed in an outer envelope with an address of the first router to which the message must be sent. The targeted router then opens the outer envelope and looks at the destination address. However, the router has multiple addresses in its routing table, one for each network connection to which it is coupled. If the router determines that the destination address is a device on one of the networks to which it is coupled, it sends the message directly to the appropriate network using the appropriate data link and physical layer communication protocols.

If the router is coupled, for example by a wide area network, to other routers, a router table is consulted. This table has entries in it each of which has a pair of data fields. The first field identifies a destination network and the second field identifies an adjacent router in the direction of that destination. A message which must be forwarded through another router will be forwarded by consulting this routing table and will be enclosed within an outer envelope and sent to the adjacent router "in the direction of", the destination address. This second router will open the envelope of the message when it is received, do a network address comparison and then forward the packet directly to the destination device.

Routers use routing protocols to exchange information about the network.

These routing protocols are software routines which run in the router. The exchange of information implemented by these routing protocols eventually causes the routing tables in all of the routers to converge so as to reflect the same network topology.

There are two types of routing protocols. The older type distance-vector protocol periodically issues broadcasts which propagate routing tables across the network. These routing protocols are useful mainly for small and relatively stable networks. Large and/or growing networks generally use data link-state protocol exemplified by the IS-IS routing protocol of the OSI model. Link state protocols send routing information only to reflect changes in the network topology. While distance-vector routing protocols always pick the path with the fewest number of routers between the source and the destination, link state protocols are different. Link state protocols can use multiple paths for failure recovery and load balancing of message traffic. Link state protocols also allow users to specify selection of routes according to delay, throughput, or reliability concerns.

Referring again to FIG. 4, the teachings of the invention also therefore encompass substitution of any of the known bridge types or a routing process for the bridge process symbolized by block 260. In the case of a router, routing tables would be substituted for the bridge database 268.

In some alternative embodiments, the microprocessor shared by the bridge functions also runs two background processes for management purposes. These background processes are symbolized by the hub/bridge in-band management process 280 and the console command process 282 in FIG. 4. The in-band management process 280 consists of a number of subroutines each of which is capable of carrying out a particular management function. These management functions are well-known to those skilled in the art and will not be detailed here nor are the details of the in-band management process critical to the invention. The teachings of the invention contemplate the fact that a single in-band management process may be shared by both the hub and the bridge functions and this management process is distributed in the sense that it is contained within the same housing as the integrated hub and bridge hardware so no failure between the network address of the hub/bridge circuitry and the network address of the management process can cause the hub/bridge circuitry to be uncontrolled.

Also, to implement one aspect of the "open system" architecture, the management process 280 conforms to the SNMP network management protocol selected as a national standard by the Internet Engineering Task Force (hereafter IETF). This means that other systems that have management software can also manage the integrated hub/bridge of the invention by sending SNMP management commands in via the modem 300, the serial port 298 and the console command process 282 and these commands will be understood by the hub/bridge management process 280 and carried out. Further, under the prior art SNMP network management protocol, every device connected to a network has data structures called MIBs which are unique to the product. MIBs effectively describe every "object", i.e., every controllable entity and some entities that are "read-only" in a particular system and describes the various states that each entity can assume. The MIB data is used by SNMP management processes to control or read the objects thereby allowing management of the system described by the MIB data. To implement the open system architecture of the hub/bridge according to the teachings of the invention, the electronics and software of the hub/bridge 130 according to the "open architecture" species within the genus of the invention implement the following national open systems Internet and TCP/IP based standards: RFC 791 (Internet Protocol); RFC 792 (Internet Control Message Protocol); RFC 793 (Transmission Control Protocol); RFC 768 (User Datagram Protocol); RFC 783 (Trivial File Transfer Protocol); RFC 826 (Address Resolution Protocol); RFC 854 (Telnet Services); RFC 903 (Reverse Address Resolution Protocol); RFC 1058 (Routing Information Protocol); RFC 1157 (Simple Network Management Protocol) RFC 1213 (MIB II); RFC 1286 (Bridge MIB); RFC 1317 (RS232-Like MIB); RFC 1368 (Repeater MIB); RFC 1398 (Ether-Like MIB); Draft RFC 802.3 MAU; IEEE Standard 802.1(d) Spanning Tree Algorithm, Filtering by Protocol. All the foregoing national standards are published by the IEEE or the IETF and are hereby incorporated by reference.

What the foregoing means is that the hub/bridge according to the teachings of the invention can be mixed into a network environment with equipment made by other manufacturers that support the same national standards and all the equipment will work together. Further, the invention contemplates that the bridge, hub and all other MIB descriptions are all integrated into one easily manageable entity such that installation is simplified in that the installer does not have to learn the complexities of the installation process for a hub and then learn the complexities needed for a separate bridge circuit installation also.

An important aspect of the invention is in the "network slice" stackable architecture implemented by the integrated hub/bridge. This architecture is especially useful in small networks and to solve the problems noted above with concentrator technology. Fundamentally, a "network slice" is a small, stand-alone repeater with integrated bridge and integrated management. More specifically, as the term is used herein, the genus of machines each of which may be referred to as a "network slice" is a stand-alone hub or repeater with 26 or fewer ports, having its own enclosure and user interface switches and indicator light, and having a built-in, i.e., integrated bridge to couple the repeater ports to a backbone, a local backbone or another repeater and LAN segment, and having distributed management, i.e., a collection of subroutines in the control software that can understand management commands and requests for data originated by a network administrator. An important species of this genus has a bypass mode to allow small networks to grow beyond the Ethernet 10 Mb/sec speed limit and then turn off bypass to allow both segments on both sides of the bridge to have traffic at less than the 10 Mb/sec limit. Another important subspecies of this genus is remotely manageable.

Network slices can solve the problems of concentrators noted in the background section of this application by allowing a network slice to be located out at the location of a group of users which is too small to justify having a dedicated concentrator. The network slice is substantially less expensive than a concentrator and can handle up to 26 users connected to each network segment on either side of the bridge so substantial pockets of physically isolated users can be handled relatively inexpensively. Network slices are "stackable" in that the individual network slices can each stand alone or work together to handle large networks via connections to each other over local "backbones", i.e., network segments coupled to the integrated bridge which may have different physical media and protocols than the network one LAN segment coupled to the hub. This means that as the network grows in number of users, new network slices can be added in smaller increments than would be possible if concentrator technology was used, and this costs less. Thus, in the hypothetical situation posed in the background section of this application, the remote network slice located at the pocket of physically separated users can also send data to other network slices at the parent company or on a different floor via the single line of the local backbone connection. This can save substantially in installation costs by eliminating the need to run separate cables for each user from the physically isolated pocket of users back to the main concentrator. Of course, a local backbone LAN segment can be used only if the distance between the network slices is small enough to be less than the maximum allowable range for the media type used for the backbone connection. If the distance is larger than this maximum distance, the network two segment is replaced with a WAN transceiver (wide area network media access unit or MAU).

The fact that a network slice has on-board integrated management software means that the network slice can be remotely managed. This is a substantial advantage in a situation where a concentrator serves the main network but there is an isolated pocket of users which is too small to justify another concentrator but with users who need to share assets on the main network. With a network slice, the network administrator can run the management process on a work station coupled to the main network and send management commands and data requests either in-band over the backbone connection or via modem to the management

process resident in the network slice located out with the isolated pocket of users. With a concentrator, this is not possible, because the management process is located on a card that must fit in a slot in the concentrator, so unless the isolated pocket of users has their own concentrator, the network segment they are on cannot be managed from the main network.

As mentioned previously, the most important species in this "network slice" genus has "open architecture". Another important species utilizes a bypass mode wherein the bridge function is bypassed and the two LAN segments connected to either side of the bridge are connected together to form a single LAN.

In the broadest open architecture species of the invention, the software executed by the microprocessor 144 of the bridge/hub 130 in FIG. 2 will implement only the Internet Protocol defined by the national standard RFC 791 on the network layer of the ISO model. This specification is publicly available from the IETF and defines the network layer protocol used on Ethernet and defines, among other things, how destination network addresses and destination node addresses are used to route data packets to the appropriate machines on the LAN. This species of network slice would be a device which would not understand SNMP management commands.

There is a virtual necessity in the open systems market for network devices which can be managed from devices made by other manufacturers. The vehicle to achieve this interoperability is through implementation of the SNMP management protocol. Therefore, an important species of the network slice genus of the invention is a network slice, as that term was earlier defined herein (a network slice includes on-board "distributed management" functionality) with open architecture. Such a machine includes, in the software executed by the microprocessor 144, routines which implement the SNMP (simplified network management protocol) defined in the national standard RFC 1157 specification published by IETF at the session and/or presentation layer of the ISO model. The SNMP protocol routines in the control software interface to the Internet Protocol on the network layer through software executable by microprocessor 144 which implements the User Datagram Protocol defined in the national standard RFC 768 published by the IETF.

Although open systems management requires SNMP management protocol to be implemented in the control program, the network slice genus does not require that the distributed, i.e., on-board management process be restricted to only understanding SNMP commands and requests. The network slice genus can also be managed directly via direct connection of the network administrator's computer to the RS232 port 298 in FIG. 4 or via a modem. In such an embodiment, it is necessary for the control program executed by the microprocessor 144 to have routines that implement the Console Command Process 282 in FIG. 4 to issue appropriate commands and requests for data to the bridge process 260 or the repeater/controller 288 via data paths 306 and 308, respectively. However, if management of the network slice is to be done "in-band" via packets sent over the Ethernet™ LAN, then those skilled in the art will appreciate that the control software executed by the microprocessor 144 must include routines which implement the TELNET Services protocol defined by IETF in their national open systems standard RFC 854, the Transmission Control Protocol (TCP) defined by IETF in their national open systems standard RFC 793 as well as the Internet Protocol (IP) defined in RFC 791 along with the Internet Control Message Protocol (ICMP) defined in IETF national open

systems standard RFC 792 and the Address Resolution Protocol (ARP) defined in RFC 826 as well as the Reverse Address Resolution Protocol (RARP) defined in RFC 903.

The way this all works together to allow in-band management of the network slice via a non-SNMP management process is as follows. When a management packet arrives at a repeater port, the physical layer hardware and software examines the MAC, i.e., Ethernet destination address thereof (which will be the MAC address of the bridge process) and causes the packet to be directed to the bridge process 260 in FIG. 4. From there the packet is directed to the in-band management queue 284 in FIG. 4 and ultimately is retrieved for processing by the hub/bridge in-band management process 280. This process 280 includes routines which implement the IP, ICMP, ARA, and RARA protocols previously mentioned. These protocols examine the data portion of the data link layer packet received from the pond and derive the Destination Network Address and Destination Node IP addresses therein. The resulting data is then passed to the TCP protocol which converts the format of the data to text strings that the TELNET protocol can understand and converts the IP address to a port or socket address which is assigned to the hub/bridge in-band management process. The TELNET terminal emulator protocol then takes the data and converts it from its text string format to a line oriented format that can be understood by the Console Command Process 282 and passes the data to the non-SNMP Console Command Process 282 for execution in controlling the network slice. For data passing the other way, i.e., from the network slice hub/bridge to the remote non-SNMP management process in-band via the Ethernet physical media, the reverse sequence of events occurs. First, the TELNET protocol converts the line oriented strings of data that it receives from any object not of an SNMP type and converts that data into text strings that can be transported by the TCP/IP protocol. These strings must then be converted to data link layer packets suitable for transmission on the Ethernet physical media by other routines in the control program executed by the microprocessor 144 that implement the TCP, IP, ICMP, and the RARA protocols. The protocols discussed in this specification are known to those skilled in the art.

Other important species of the network slice genus allow remote upgrading of the software which the microprocessor 144 executes. In an open systems embodiment of a network slice with this capability, the software of the control program has routines which implement at the transport layer the Trivial File Transfer Protocol (TFTP) specified by IETF in the RFC 783 national standard. The TFTP Protocol interfaces with the physical media through the Internet Protocol on the network layer and the EtherTalk™ Link Access Protocol on the data link layer. This allows new and improved versions of the control program which controls processing by the microprocessor 144 to be loaded into the hub/bridge by the network administrator via a modem or in-band through whatever network path connects the network administrator's machine and the hub/repeater.

To understand the significance of the savings in network management traffic from combining the hub and bridge, some information about typical network management commands is helpful. Typical of management functions are to turn ponds on and off, set protocol filtering, inquire regarding network traffic volume, inquire as to polarities status at each port, inquire as to the number of errors which are occurring on a particular pond, analyzing traffic patterns on individual networks and across bridges, collecting data detected by the intelligent hub repeater circuits regarding errors and error types on a per-port basis, obtaining statis-

tical data regarding the number of packets forwarded versus the number of packets received configuring the repeaters via software commands, putting the bridge in bypass mode, etc. An intelligent hub allows ports to be turned on or off, provides error correction and can provide statistics regarding traffic volume.

The sole job of the hub/bridge in-band management process 280 is to receive so called in-band management commands and status inquiries and to process them. In-band management commands and inquiries are basically management messages which arrive like other data packets through one of the local area network controllers such as devices 262 or 264 in FIG. 4. This allows the network manager to manage components across the whole network while being connected to only one component thereof.

To implement the in-band management process, the bridge process 260 monitors data packet message traffic coming from the local area network controllers 262, 264 for any data packets having a destination address assigned to the integrated hub/bridge. These packets are forwarded to the in-band management process. These data packets are forwarded by placing them in a management input queue 284 implemented as a FIFO buffer in memory.

Typically the in-band management process runs in the background, so when a time slice is awarded to the management process or an in-band management interrupt occurs, the in-band management process 280 reads the next management command or inquiry in the input queue 284 and processes the management function appropriately. This process may involve sending protocol filtering commands to the bridge process 260 via data path 286 or collecting information from the repeater controller 288 via path 290. It may also involve sending commands to the repeater/controller 288 via path 292. Likewise, it may involve writing data to a configuration/database 294, or obtaining information from that database as symbolized by data paths 296.

Out-of-band management is carried out by the background console command process 282 in some embodiments. Out-of-band management commands and status inquiries are commands received not as data packets from the LAN controllers but received directly from the network manager's terminal. In some embodiments, these commands are received via a serial port 298 which may be connected to a modem 300 or a terminal. In some embodiments, two serial ports may be used, one connected to a modem and one connected to a terminal. This allows a network manager to dial in via the telephone lines from a terminal at home and issue management status inquiries and network management commands via the modem 300. The network manager may also issue any of the network commands or status inquiries via a terminal 302 in his or her office. The function of the console command process 282 is to receive these commands and status inquiries and interact appropriately with the repeaters, bridge process or configuration/status database to carry out the desired function. This interaction is carried out via data paths 306, 308 and 310. In the case where the management command is not addressed to the hub/bridge to which the network manager is directly connected, the console command process places the command in a data packet and places it in the transmit queue of the appropriate network controller so that it will eventually reach the destination component to be managed.

Referring to FIGS. 5A and 5B, there is shown a flow chart of the software bridge process used in the preferred embodiment. This process starts with the step symbolized by block 340 of getting the next received packet out of the FIFO receive buffer in memory after having received the interrupt from the LAN controller.

Next, the test of block 342 is performed to determine if there is any transmission error in the packet. If there is an error, the packet is discarded, as symbolized by block 344.

If there was no error, the process of block 346 is performed to update the bridge database. This is done by examining the source address of the packet and the network identification, i.e., the LAN controller, from which the packet came and writing this information into an entry into the bridge database forwarding table.

Next, the process of block 348 is performed to read the destination address of the packet and look for this address in the bridge database forwarding table.

The test of block 350 is then performed to determine if the destination address is on the same side of the bridge, i.e., on the same network, as the source address from which the packet originated. If the destination address is on the same side of the bridge as the source address, the packet is discarded as symbolized by block 352. The discard process involves rewriting the pointers on the linked list of the receive buffer to remove the discarded packet from the linked list.

Next, the destination address of the packet is read to determine if the destination address is the address which has been assigned to the integrated hub/bridge on which the bridge process is running. This process is symbolized by block 354 in FIG. 5A. If the packet is an internal hub management packet, then the test of block 356 is performed to determine if the packet is group addressed. In some protocols, the packets may be addressed to multiple network addresses with a single multicast or group address. If the packet is not a multicast packet, then the step symbolized by block 358 is performed to put the in-band management packet into the hub management input queue symbolized by block 284 in FIG. 4. Note that this monitoring process for in-band management data packets goes on even if the bridge is in bypass mode since there has been no step in the process shown in the flow chart of FIGS. 5A and 5B up to this point to determine whether the bridge is in bypass mode or bridge mode. Bypass mode is symbolized in FIG. 4 by the dashed lines 265. The process of filtering out in-band management packets for forwarding to the hub/bridge in-band management process while in the bypass mode is symbolized by dashed line 267 in FIG. 4. Also, note that FIG. 4 is somewhat deceptive in that the LAN 1 controller 262 is actually the network connection for the bridge to the network serviced by the repeater/controller shown at 90 in FIG. 2 and 140 in FIG. 3. Conversely, the network segment "on the other side" of the bridge is symbolized by the network connection to the LAN 2 controller 263. LAN controller 263 services the backbone connection network segment. The physical media for this backbone connection is shown at 124 in FIG. 2 and 164 in FIG. 3. Note that in FIG. 4 there is no apparent data path between the repeater/controller 288 and the bridge process, but this data path does exist through the LAN 1 controller 262.

In an alternative embodiment, an isolate mode is implemented in the hub/bridge software. The purpose of this isolate mode is to cut off data packets from being forwarded between networks one and two. This helps isolate problems on the network for troubleshooting purposes. In isolate mode, the bridge process discards all incoming data packets from either network except in-band management packets such that no data packets get forwarded from one network to the other. In-band management packets get selected from the data stream and are placed in the input queue of the hub/bridge in-band management process 280. Thus the management of the network can continue during isolate

mode to assist in the troubleshooting process. Isolate mode is symbolized by dashed barrier line 264 in FIG. 4, and the process of selecting in-band management packets for forwarding to the in-band management queue is symbolized by dashed line 269. The isolate mode can be implemented in any manner, but in the preferred embodiment of the integrated hub/bridge, it is implemented by setting the pointer addresses in the forwarding vectors to be described below to point to a packet discard routine. This is done in the initialization code in the preferred embodiment, but could be done at other times by the network manager in other embodiments. Further, in some embodiments, forwarding vectors need not be used, and the bridge process can, for example, check a status register having contents set by the user to determine the current mode of operation and then process the incoming packets accordingly.

Referring again to FIG. 5A, if the step of block 356 determines the packet is a multicast packet, then the step symbolized by block 360 is performed to copy the packet. After making a copy of the packet, the packet copy is placed in the input queue for the hub/bridge in-band management process as symbolized by block 361.

If the step of block 354 determines that the data packet is not an internal hub management packet, or the packet was a multicast hub management packet and was copied by block 360 and loaded in the management input queue, then the test of block 362 is performed. This test is to determine if the protocol of the packet is a protocol type for which a filter has been activated. An active filter condition indicates that the user does not desire packets with this communication protocol to be forwarded, even if the destination address is such that the packet would otherwise be forwarded. Protocol filtering is a feature of the bridging process which may be activated by the network manager either through an in-band management command or a out-of-band management command entered through the console command process symbolized by block 282 in FIG. 4. In some situations, it is desirable for example to prevent any Ethernet protocol packets from being forwarded from an Ethernet local area network on one side of the bridge to a Token Ring network or an FDDI backbone connection on the other side of the bridge. In this case, the network manager simply sets a protocol filter blocking any Ethernet data packets from being forwarded. This is the purpose of the test on block 362. If block 362 determines the packet should not be forwarded because it has a protocol which is being filtered out, then the step of block 364 is performed to discard the packet.

If the test of block 362 determines that the packet protocol is a type which is not being filtered, then the test of block 366 is performed to determine if the data packet came from the LAN 1 controller. If it did, the process of block 368 is carried out to read a pointer address from a LAN 2 forwarding vector. This pointer address is written during initialization of the integrated hub/bridge circuit by the microprocessor. The particular pointer address written into the memory location assigned to the LAN 2 forwarding vector will depend upon whether the user has indicated that the hub/bridge is to operate in the bypass mode, bridge mode or isolate mode. There is also a LAN 1 forwarding vector which is assigned a different memory location. The LAN 1 forwarding vector also stores a pointer address. This pointer address is also written during initialization time, and will point to a routine which carries out the desired processing of either the bypass mode, bridge mode or isolate mode. The user indicates in any known manner such as front panel switch positions which mode is desired. Thereafter, at initialization time, a pointer address appropriate to the selected

mode is written into the LAN 1 and 2 forwarding vectors. The process of step 368 will read the LAN 2 forwarding vector and vector processing to block 370 if either bypass or isolate mode is selected, or to block 380 if bridge mode is selected.

If the LAN 2 forwarding vector points to the bypass or isolate mode, then the step symbolized by block 372 is performed to discard the packet. Discarding the packet implements bypass mode by virtue of the switch positions, e.g. switch SW1 in FIGS. 6A and 6B being set by a routine which is not shown to a switch position in bypass mode so as to connect the LAN 2 interface 466 or 464 directly to the AUI port of the repeaters 440 and 442. Thus any packet that arrived at a LAN 1 port is automatically sent out on LAN 2, and vice versa. In isolate mode, the switch positions for, e.g., switch SW1 in FIGS. 6A and 6B, are set so that the LAN 2 interface 466 or 464 are connected to the LAN 2 controller 472. Thus, discarding the packet by the bridge process prevents any transfer of packets from LAN 1 to LAN 2 or vice versa. Processing then returns via path 374 to the top of the bridge loop at 376 in FIG. 5A.

If the LAN 2 forwarding vector points to a routine for the bridge mode, then the processing of step 382 is performed to transmit the data packet to LAN 2 using the LAN 2 controller. This is done by the LAN 2 controller placing the packet into the transmit buffer for LAN 2. This process entails rewriting the pointers on the linked list for the transmit buffer to include the new packet in sequence in some embodiments. Processing is then returned to the top of the loop via path 375.

If the test of block 366 determined that the packet did not come from LAN 1, then in the preferred embodiment, the data packet must have come from LAN 2. This is only true in the isolate or bridge modes however, because in the bypass mode, the switch positions of, for example, switch SW1 in FIGS. 6A and 6B, are set such that the LAN 2 controller is not coupled to any LAN. Therefore, path 367 will only be taken when the hub/bridge is operating in either the isolate or bridge modes. In that case, the process symbolized by block 390 is performed to read the pointer address from the LAN 1 forwarding vector and vector processing to the routine pointed to by that vector. In isolate mode, the processing of block 394 is performed to discard the packet, and control is returned via path 396 to the top of the bridge loop. This implements the isolate mode in the same way as described above by preventing the transmission of the packet from LAN 2 to LAN 1 as there is no direct connection in this mode from the repeater AUI port to LAN 2.

In bridge mode, the step of block 400 is performed to forward the packet to LAN 1 using the LAN 1 controller by a process similar to the process of block 382. Of course, in bypass mode, processing will never reach this step, so step 400 is really only performed for a packet arriving from LAN 2 in bridge mode. Processing is then returned to the top of the bridge loop via path 402.

In other embodiments, the pointer addresses in the forwarding vector memory locations may be written at any time by the network manager.

In still other embodiments, where the isolate mode described above is implemented, the pointer addresses of both the LAN 1 and LAN 2 forwarding vectors will be set to point to a packet discard routine. In isolate mode, the switches controlling whether the LAN 2 interface (switch 151 in FIG. 2, switch 112 in FIG. 3 or switch SW1 in FIG. 6B) is driven by the LAN 2 controller, i.e., the bridge process or by the AUI port of the repeater, are set in the same position as they are set for bridge mode of operation.

Forwarding vectors are used in the preferred embodiment to increase the speed of processing of data packets. In an alternative embodiment, forwarding vectors may be eliminated and the steps of blocks 368 and 390 may be altered to read the configuration database to determine whether the hub/bridge is in bridge mode or bypass mode and then carry out appropriate processing to either discard the packet or forward the packet to the other network.

Referring to FIGS. 6A and 6B, there is shown in a block diagram of the preferred embodiment of an integrated hub/bridge. Two repeater/controllers 440 and 442, implementing an Ethernet data link layer communication protocol, drive a plurality of 24 port interface transceiver circuits indicated generally at 446, 448 and 449. These port interface circuits can be 10Base2, 10BaseT, 10BaseF or FOIRL specific. The physical media connected to the port interface circuits can be unshielded twisted pair, coaxial cable, fiber optic waveguide etc. Any data entering on any one of the 24 ports is automatically repeated by the repeater/controller chips 440 and 442 out on all the other ports. The repeater/controllers are also known as RICs in the trade. Data is transmitted from one repeater/controller to the other via an interRIC data bus 450 which is also coupled to a LAN 1 controller 452 in the claims appended hereto, this bus is referred to as the network one data bus. The 24 ports indicated at 446, 448 and 449 comprise local area network 1 for the bridge process.

Each of the repeater controllers 440 and 442 has a AUI output port indicated at 456 and 458. The AUI port 458 is coupled to a software controlled switch SW1 which selectively couples either bus 458 or bus 470 coupled to a LAN 1 controller 452 to a bus 462 depending upon whether the integrated hub/bridge is operating in bridge or bypass mode. The bus 462 can be selectively coupled by switch SW3 to either of two LAN 2 interface transceivers which drive the physical media of LAN 2. A switch SW2 selectively couples an AUI port 456 on RIC 440 to LAN 2 transceiver 464 in some embodiments. This allows the integrated hub/bridge to have two backbone ports operating simultaneously one of which is a repeater and one of which is bridged by proper settings of switches SW1, SW2, and SW3. Switches SW1 and SW3 can be software driven, manually operated or some combination thereof.

In bypass mode, at initialization time, switch SW1 is set by the microprocessor 460 to connect the AUI port 458 to bus 462. Switch SW3 is also set during initialization time to select either LAN 2 interface 464 or 466. In some embodiments, upon failure of one of the LAN interfaces 464 or 466, the microprocessor will automatically attempt failure recovery by changing the state of switch SW3 to select the other LAN interface so as to maintain communications with LAN 2 in either the bridge mode or the bypass mode.

If the user has selected bridge mode, during initialization time, the microprocessor will set switch SW1 to connect bus 470 to bus 462. This allows the bridge process performed by CPU 460 in software to drive the LAN 2 interface via a LAN 2 controller 472 for packets that need to be forwarded from LAN 1 to LAN 2 or vice versa. Incoming packets from LAN 2 will arrive via the selected LAN 2 interface 464 or 466 and will be transferred to the LAN 2 controller 472. The LAN 2 controller will then generate an interrupt to the CPU 460 and deposit the packet in dynamic random access memory (DRAM) 478 using DMA controller 480.

The bridging routine is embodied in a computer program which is stored in nonvolatile memory in the form of field erasable programmable read-only memory 490. This software also contains the initialization code which sets up the

repeaters and sets the switch positions for bypass mode or bridge mode and writes the forwarding vector address pointers according to whatever mode is selected by the user. The initialization routine in pseudocode is as follows:

```

If dynamic RAM test fails:
  Stop
Set up software environment
If non-volatile RAM (NVRAM) checksum is OK:
  Read system parameters from NVRAM
Else
  Reinitialize system parameters in NVRAM to defaults
Determine hub type from ID PROM
Select hub mode of operation (bypass or bridge)
Do preliminary configuration of RICs
If any network interface tests fail:
  stop
Initialize the I/O buffers and the bridging database
Do final hub/RIC configuration
Initialize the network interfaces (hardware)
Activate bridging
Initialize hub management agent
Start console command processor

```

The central processing unit 460 initializes the repeater/controllers using tristate buffers 500 and 502. These buffers are coupled to the data, address and control buses, 504, 506 and 508, respectively, of the CPU 460, and essentially serve multiplexer functions in multiplexing data, address and control information from buses 504, 506 and 508 onto 8-bit shared RIC buses 510 and 512 of repeater/controllers 440 and 442.

The repeater/controllers 440 and 442, in the preferred embodiment, are National Semiconductor 83950 Ethernet RIC's, and are intelligent in the sense that they can sense certain things about the data packets being received and transmit data regarding network traffic to the microprocessor 460. This feedback data from the RIC's is transmitted to the microprocessor through the LAN 1 controller 452.

In the preferred embodiment, a microprocessor 460 is used to implement the bridge function, do initialization and carry out management functions. This CPU is any one of the Motorola 680X0 series.

The repeater/controllers also drive twenty-four CRS light emitting diodes symbolized by block 514. These diodes flicker to indicate when there is traffic on their respective ports. There is one CRS diode for every port, and each individual diode is driven through addressing and multiplexing LED logic units 516 and 518. These logic units allow the eight-bit buses 510 and 512 to be shared such that the repeater/controllers can use buses 510 and 512 to drive the LED's 514, while the CPU can use these buses to initialize the repeater/controllers and to send management commands to them to turn on and turn off ports, etc.

The logic units 516 and 518 are also used to address and store data from the RICs in a polarity status register 520. It is possible to connect the physical media to the LAN 1 interface ports with reversed polarity. If this happens, that port will not work, and this information is of interest to the network manager. Therefore, the logic circuits 516 and 518 are also used to convey polarity status information from the repeater/controllers 440 and 442 to the polarity status register 520. The polarity status information is read by the microprocessor and conveyed to the network manager.

Nonvolatile random access memory 540 is used to store the configuration and status database information as symbolized by block 294 in FIG. 4.

A serial communications controller 542 and interface logic 544 are used to couple the hub/bridge circuit to an out-of-band management control device such as terminal

302 or modem 300 in FIG. 4. This allows management functions to be invoked by the network manager via a direct coupling to the hub/bridge.

An ID PROM 546 stores the data link layer address of the hub/bridge such that in-band management data packets may use this address as their destination address and be forwarded by the bridge process to the management queue in memory 478.

The microprocessor 460 controls twenty-four status LED's symbolized by block 560. These LED's are controlled through an enable LED register 562 and a disable LED register 564. These registers may be addressed by writing their addresses on the CPU address bus 506. This address is decoded by an address decoder gate array 566 which generates appropriate chip select signals to enable the appropriate chips which are to have a transaction with the CPU.

The status LED's have a color which indicates the status of each port. If a port is functioning correctly, its corresponding status LED will be green. The corresponding status LED will be red if any of three error conditions exist for the port. These three error conditions are: improper polarity, the port is partitioned, or there is no link pulse.

The status register 580 and the LED and command circuit 582 are used by the central processing unit to signal certain conditions relating to the status of the combined hub/bridge. Typically there are eight LED's in circuit 582, four of which are used to signal hub status and four of which are used to signal network status. The CPU controls these LED's by writing data into registers in the circuit 582. The four hub status LED's are used to indicate whether power is on, whether a fault has occurred, whether the hub is in bridge or bypass mode, and whether the physical media is connected. The four network status LED's are used to indicate when data is being received from local area networks 1 and 2 and when data is being transmitted on networks 1 and 2. Normally, the local area network controllers 452 and 472 control these network status LED's during fault free operation. However, when a fault occurs, the microprocessor 460 takes over control of these LED's and writes data to the circuit 582 to cause the LED's to light in a pattern which indicates the type of fault which occurred.

The DIP switches 584 are used to troubleshoot the hub/bridge system, to select between AppleTalk Phase 1 or Phase 2 and to flush the NVRAM.

Static RAM 586 is used to store parameters for the network.

LAN 1 utilization register 588 and LAN 2 utilization register 590 are used to store counts which indicate the volume of traffic flow on local area networks 1 and 2, respectively.

Since the dynamic random access memory 478 is shared between the local area network controllers 452 and 472 via the DMA circuit 480, and the microprocessor 460, the data, address and control buses of memory 478 must be multiplexed to implement this sharing. Likewise, the microprocessor must be able to write data to the local area network controllers at the outset to inform these controllers of the locations of the transmit and receive FIFO buffers which are established in memory 478. The microprocessor also stores the bridge forwarding tables in DRAM 478.

The data bus 504 of the microprocessor is coupled to the data bus 602 of the DRAM and the data bus of the local area network controllers by a tristate buffer 604. Data bus 602 is coupled to a shared data/address bus 606 of the DMA device 480 by a tristate buffer 608. The buffers 604 and 608 have their tristate status controlled by a bus grant programmable

array logic 610. Three arbitration PALs 610, 612 and 614 are used to arbitrate requests for access to the data, address and control buses such that the DRAM 478 may be shared between the DMA controller 480 and the CPU 460. For simplicity, the connections between these PALs and the buffers and multiplexers they control are generally not shown.

The address bus of the DRAM 478 is multiplexed by an address multiplexer 616 which has as its two inputs the DMA address bus 618 and the CPU address bus 620. Tristate buffer 622, coupling the microprocessor address bus 506 to the address bus segment 620, is controlled by arbitration PAL 612 to isolate the CPU address bus 506 from the address bus segment 620 when the DMA address bus is active. Selection of the address input to apply to the DRAM address bus 630 is controlled by a programmable array logic 632.

Control signals from the DMA device on bus 634 and control signals from the CPU on bus 636 are coupled to the two selectable inputs of a control multiplexer/PAL 640. The MUX/PAL 640 also receives three control inputs from the control PAL 614, one of which controls selection of the particular control bus input to couple to the output bus 644. A portion of the signals on bus 644 are applied as input signals to the PAL 632 to control its state and two output signals from the address multiplexer 616 are also applied as inputs to this PAL 632. PAL 632 generates the row address strobe/column address strobe signal on line 648 to control whether the address on bus 630 to the DRAM is used to address a row or column. The PAL 632 also receives a refresh signal on line 650 from a timing circuit (not shown) which causes the PAL 632 to refresh the DRAM 478 at a 64-kilohertz rate. The PAL 632 also generates a handshake signal on line 652 to inform the control PAL 614 that a refresh cycle is under way and to not attempt to grant bus access to either the DMA control bus 634 or the CPU control bus 636 via buffer 637.

Finally, a tri-state buffer 660 is used to multiplex the data/high address bus 606 from the DMA device 480 such that when the DMA is granted access to the address bus of the DRAM, bus segment 606 is coupled to the low address bus segment 607 to form a 24-bit DMA address bus 618.

The control MUX/PAL 640 also generates a read/write control signal to the DRAM on line 680 to control whether the DRAM is reading or writing. The MUX/PAL 640 also generates a read/write control signal to the LAN controllers 452 and 472 to allow these controllers to be either written or read by the microprocessor 460.

The bus grant PAL 610, the arbitration PAL 612 and the control PAL 614 control the states of the PAL 632 and the buffers 604, 608, 660, 622, and 637 so as to time division multiplex or arbitrate the data, address and control buses such that the DRAM 478 may be shared. The details of this bus arbitration or multiplexing are not critical to the invention, and any other arbitration scheme known in the prior art may also be used and still be within the scope of the teachings of the invention. Further, in alternative embodiments, separate DRAM memories may be used for the local area network controllers, and the bridging process and for any other process which needs DRAM memory assigned to it such that bus arbitration can be simplified.

The program executed by the microprocessor 460, written in C source code and assembly language on an Apple MPW 3.0 development system, is attached hereto as Appendix A. The microprocessor 460 is a Motorola 68000 in the preferred embodiment, but other microprocessors in the 68000 series should also execute this code properly when compiled

for their particular machine language. The actual schematic diagrams and PAL equations of the best mode of carrying out the teachings of the invention are included herewith as Appendix B and Appendix C.

Referring to FIG. 7, there is shown a block diagram of one embodiment of a different type of packet switching network hub apparatus than the combined hub/bridge described above. The apparatus is comprised of a high-speed, shared, multiport memory system 800 which has two ports in this particular embodiment. One of the ports is coupled by an E bus 802 and an E bus driver circuit 799 to a plurality of conventional LAN controller chips, of which LCC #1 is typical. The LAN Controller Chips (hereafter sometimes referred to as LCC's) are available commercially from various suppliers like National Semiconductor of Santa Clara, Calif., and are sometimes also referred to as "Sonic" chips. Each LAN Controller Chip is coupled to its own Ethernet media segment via a Media Access Unit (hereafter MAU). Each Ethernet segment, such as segment 805 coupled to MAU 807 and LCC 809, typically has a 10 Megabit/sec data carrying capacity which is defined by the Ethernet standard.

The E bus 802 is also coupled to an Ethernet processor 804. The Ethernet processor 804 configures the LAN controller chips and creates in the high speed memory a separate transmit and receive buffer for each LAN controller and a separate area of memory for storing receive and transmit status data for each LAN controller, each separate area of memory storing status data being hereafter called a descriptor. In the particular class of embodiments symbolized by FIG. 7, the Ethernet processor 804 then assigns each LAN controller chip to a specific transmit buffer and a specific receive buffer in the high speed memory system 800, and these assignments are fixed and do not vary over time. In alternative embodiments to be described below, the Ethernet processor 804 allocates at least one receive buffer and one transmit buffer for each LCC, but the particular receive buffer in which is stored any particular packet being handled depends upon which buffers are free at the time the packet arrived. In other words a table of free receive buffers is kept and consulted when a packet arrives to find an open receive buffer in which to store the packet. In the preferred embodiment, each receive buffer is the same size, but in other embodiments, only enough memory is allocated for each packet as that particular packet needs for greater memory utilization efficiency at the expense of some processing power devoted to determining how much memory to allocate to each packet. Equal size buffers for all receive buffers increases data throughput by eliminating the need for processing to determine how much of the memory system to devote to each packet.

The Ethernet processor 804, in the class of embodiments represented by FIG. 7, creates the descriptors in the high speed memory 800 by assigning a unique range of addresses in the high speed memory 800 for the descriptor for Lan Controller Chip LCC #1 and another unique range of addresses for the descriptor for Lan Controller Chip LCC #2. These descriptors are then organized as a linked list by the Ethernet processor by writing as the last field (or some other predefined field in the descriptor space) a pointer to the start of the next descriptor. The pointer is the address in high speed memory where the descriptor for the next LAN controller starts.

The Ethernet processor 804 then assigns each particular LAN Controller Chip to a unique descriptor dedicated to supporting only that Lan Controller Chip (hereafter sometimes referred to as an LCC). The Ethernet processor 804

then assigns each LCC to unique transmit and receive buffers dedicated to supporting only that Lan Controller Chip. These two steps are done by informing each LAN Controller Chip of the range of addresses that comprise the descriptor for that LCC and the range of addresses comprising the receive buffer into which received data packets from that LCC are to be deposited. The Ethernet also informs each LCC of the address range in high speed memory 800 where each LCC can find data packets to be transmitted on the Ethernet segment connected to its corresponding MAU. In FIG. 7, transmit buffer 812 and receive buffer 814 are assigned to LCC 809, while transmit buffer 816 and receive buffer 818 are assigned to LCC 820. Each of LCC 809 and LCC 820 has a descriptor entry somewhere on the linked list symbolized by descriptor ring 808.

In alternative embodiments, the LAN controllers can have enough intelligence to coordinate with each other to assign their own descriptor memory spaces, and transmit and receive buffers thereby eliminating the need for a separate Ethernet processor 804. In another alternative embodiment, the LAN Controller Chips can have sufficient on-board memory to store incoming packets and status data and to temporarily buffer outgoing packets before they are transmitted. The central high speed memory class of embodiments symbolized by FIG. 7 is preferred however because LAN controllers with on-board memory would have to have enough memory to store a plurality of data packets in cases where the main microprocessor processing (to be described below in more detail) is too slow to take all packets as they are received. This could require too much memory and make the LCC's too expensive.

Returning to the consideration of the class of embodiments symbolized by FIG. 7, the individual transmit and receive buffers assigned to the LAN controller chips are located in an address space which is shared with a main microprocessor 806 which serves to do bridging and routing functions as will be described in more detail below.

Because each LAN Controller Chip has its own MAU, each of the LAN Controller Chips can be coupled to any type of Ethernet media. For example, LCC #1 may be connected through a 10BaseT type MAU to a twisted pair media segment 805, while LCC#2 may be connected through a 10Base2 type MAU #2 to a coaxial cable type media segment 820. Another LCC may be connected to a fiber optic backbone link etc. Each media segment such as segment 805 may be connected to a computer or other peripheral or it may be connected to a network input port of a hub or another switching apparatus such as the genus of apparatus symbolized by FIG. 7. In the preferred embodiment, there are 12 LCC's, 12 MAU's and 12 media segments. Thus, as many as 12 LAN's could be connected together by the packet switching machine shown in FIG. 7.

Each LAN controller chip may be coupled to a computer or other peripheral via a particular LAN segment, or may be connected to another packet switching device or hub such that networks of very large size may be built as well as networks of smaller size.

After the buffers for LCC #1 are assigned to it, the Ethernet processor 804 turns on LCC #1 and it begins to listen for incoming data packets on media #1. The same scenario applies to each LAN controller.

When a packet starts arriving, the LAN controller chip connected to the network segment on which the packet is arriving asynchronously starts depositing data from the packet into the receive buffer assigned to that LAN controller. The LAN controller also accesses descriptor file assigned to it and writes status data thereto indicating that a

packet is arriving. Typically, the LAN controller deposits the packet information in its receive buffer by performing a DMA transaction and then does a DMA access to the descriptor ring and sets a status bit indicating that the LAN controller is receiving a packet. However, mechanisms other than DMA may also be used in other embodiments such as conventional read and write transactions involving the Ethernet processor 804 to write the data to the main memory after the LCC generates an interrupt or upon the LCC being polled by the Ethernet processor 804.

After a LAN controller chip has received a complete packet, the LCC performs an error detection process on the packet. In some embodiments, the LCC may also correct any errors it finds within range of the ECC bits appended to the packet, and in other embodiments, the LCC may simply ask for retransmission.

Once the packet has been correctly received, the LCC does a DMA access to the descriptor buffer or record assigned to the LCC and sets a new status bit or changes the status bit previously accessed so as to indicate that a complete, correct packet has been received and is stored in the receive buffer of the LAN controller in the preferred embodiment, the descriptor buffer for the LAN controller that received the packet will also be updated with a pointer to the address in the appropriate receive buffer where the data of the received packet starts.

The Ethernet processor 804 also functions to determine when complete and correct data packets have been received and then refers these data packets to the main microprocessor 806 for further processing. To perform this function, the Ethernet processor continuously polls the "descriptor ring" 808 to determine which LCC's have stored received packets that are ready for routing or other processing such as passing the packet to a management function. To do this, the Ethernet processor 804 reads the status bit or bits of each descriptor buffer in the descriptor ring linked list 808. When status data is detected in a particular descriptor buffer indicating that a complete and correct data packet has been received and is waiting in the receive buffer of a particular LAN controller associated with the descriptor in which the data was found. When the Ethernet processor determines from polling the descriptor rings that a particular LAN controller has successfully received a packet, the Ethernet processor writes a pointer to the received packet into queue 810 of high speed memory 800. The queue 810 serves as a sort of FIFO stack of pointers used to prioritize the routing, bridging and other processing functions of the main microprocessor 806. In the preferred embodiment, the Ethernet processor 804 retrieves the pointer to be stored in queue 810 from the descriptor buffer itself. In other embodiments, the Ethernet processor 804 learns of the presence of a packet in a receive buffer from data in the descriptor and then reads an on-board memory or register in the corresponding LCC to retrieve a pointer to the packet. This pointer is then stored in the processing queue 810 for the main microprocessor. The processing queue must be in a shared address space of both the Ethernet processor 804 and the main microprocessor 806.

The processing queue 810 is essentially a table in high speed memory 800. This table serves the function of providing an expandable buffer for pointers to received packets in case the rate at which packets are being received by the LCC's exceeds the rate at which these packets are being processed by the main microprocessor 806. The main microprocessor starts processing received packets using the pointer at the top of the table and continues to process packets having pointers stored in other locations in the table

sequentially retrieving the pointers stored in lower slots of the table until the bottom of the table is reached. The main microprocessor keeps track of its position in the table using a pointer which is moved to the next table location when a packet has been processed by the main microprocessor. When the bottom of the table is reached, the pointer is reset to the top of the table. Likewise, a pointer is used by the Ethernet processor 804 in filling the table, and when the bottom of the table is reached, the pointer is reset to the top of the table to start filling the table again from the top.

The Ethernet processor cannot reset its pointer to the top of the table until it is sure that the main microprocessor 806 has processed the packet pointed to by the pointer in the top of the table which is about to be overwritten. This can be done in several ways. For example, a bit reserved for "processed/not processed status" in every table entry may be set by the main microprocessor 806 as a packet is processed. The bit would be set by the main microprocessor to a "processed" state whenever processing of the packet pointed to by the pointer in that table entry is complete. When this bit is found in the "processed" state, the Ethernet processor 804 would know that that table location is available for use in storing a pointer to a new packet awaiting processing in another embodiment, the Ethernet processor 804 would simply compare its pointer position to the current pointer position for processing by the main microprocessor, and, if the main microprocessor's pointer was lower in the table than the pointer of the Ethernet processor, then the Ethernet processor is free to assume that all storage locations down to the position of the main microprocessor pointer are available for use in storing new pointers. In some embodiments, the queue 810 may be organized as a linked list in such an embodiment, the easiest way to prevent overwriting pointers for packets that have not been processed is through use of a "processed/unprocessed" bit in each record in the linked list chain.

The main microprocessor 806 uses the pointers in queue 810 to access the received packets in whatever receive buffers they reside. The main microprocessor then looks at the addressing information in the packet header and decides what to do with the packet. The main microprocessor is responsible for doing bridging, routing, network management and possibly other miscellaneous functions. Some of the possibilities with regarding to handling a particular data packet by the main microprocessor are to discard the packet, transfer the packet to a management process or pass a pointer to the packet to a management process or bridge the packet to its destination on another media segment other than the one on which the packet arrived.

In the preferred embodiment, where a packet has to be bridged or routed by the main microprocessor and transmitted out on another media segment other than the one on which the packet arrived, the main microprocessor writes a pointer to the packet into the transmit buffer assigned to the LCC coupled to the media segment upon which the packet was transmitted. In the preferred embodiment, the LCC's have sufficient intelligence to continually poll their transmit buffers. Any pointers in a transmit buffer will indicate the address in the receive buffer where the packet associated with that pointer can be found. When a pointer to a packet is found, the LCC uses the pointer to access the data packet from the receive buffer where the packet is stored and retrieves the packet. The packet is then transmitted in some alternative embodiments, the main microprocessor may generate an interrupt signal or otherwise send a message to the LCC coupled to the media segment upon which a packet is to be transmitted when a pointer to the packet has been

placed in the transmit buffer of that LCC if a packet has not yet been transmitted, for example by LCC 809, and another packet arrives in the same or a different receive buffer which must also be re-transmitted on the media segment 805, the main microprocessor 806 places a pointer to that packet in the transmit buffer 812 in the next position that is unoccupied by other pointers therein.

To perform the routing, bridging and switching functions, the main microprocessor uses an 8000 entry routing, bridging and switching table stored in dynamic random access memory 822. The main microprocessor manages this table to implement a learning function similar to the bridge learning process described above for the network hub with integrated bridge.

The advantages of the packet switching structure shown in FIG. 7 over the network hub with integrated bridge are that many more local area networks may be connected together and the packet switching/bridging/routing functions are much faster. In fact, the switching/bridging and routing functions are performed at "media rate". For example, media segments such as segments 805 and 820, can each be receiving data at a rate of 10 megabits per second, the maximum allowable Ethernet rate of data transmission. If all 12 media segments are receiving data at that rate, the problem is to bridge, route and otherwise process all those packets without losing a packet. The class of embodiments symbolized by FIG. 7 can do this with the aid of the special memory structure shown. To handle the traffic volume mentioned above, extremely fast static random access memory having at least two and optionally 3 or 4 ports is used for high speed memory 800. To further speed up operations, data packets are not actually moved from the receive buffers to the transmit buffers to save the multiple memory cycles that would be required to do this. The only data that moves around the high speed memory are pointers to the data packets. In other embodiments where such high speed "media rate" operation is not required, the data packets themselves can be moved.

In the preferred embodiment, the high speed memory is designed to have three ports one of which is a high speed backbone interface. In a broader genus of the invention, this third high speed backbone port is omitted. In this genus, only two ports for the high speed memory 800 are needed. These two ports are coupled to the E bus 802 and the M bus 824. Like the E bus 802, the M bus 824 is coupled to the port of the high speed memory 800 through an M bus driver circuit. The third port to the high speed backbone, 828, is shown in dashed lines because it is optional. This port is actually an interface circuit to a very high speed backbone media 826 such as ATM, FDDI or Fast Ethernet. The ATM/FDDI/Fast Ethernet port 828 includes a microprocessor 830 that executes code stored in dynamic random access memory 832. The microprocessor 830 serves to convert the protocol used on the FDDI, ATM or Fast Ethernet media 826 to the protocol used on the regular Ethernet media such as media 805 and vice versa. The microprocessor 830 also stores any management packets arriving from the FDDI ring or other backbone media segment 826 in memory 832.

The fourth optional memory port is represented in dashed lines by expansion port 834. This interface circuit includes a microprocessor 836 which offloads part of the work of main microprocessor 806 is performing routing and management packet interpretation and execution of requested management functions.

Part of the high speed memory system is an arbitration circuit 838 that manages contention for the address and data ports of the memory chips that comprise the memory banks

of the high speed memory system 800. The details of the arbitration circuit are not critical to the invention and can be conventional, but in the preferred embodiment, the arbitration circuit is implemented with a field programmable gate array. This FPGA has as outputs all the data, address and control lines of the static RAM chips in the memory system SIMM, and has as inputs all the lines of whatever number of ports are implemented in the memory system. The Boolean equations that define the logical relationships between the inputs and outputs is appended hereto as Appendix A.

The fundamental issue handled by the arbitration circuitry is handling conflicting memory access requests from the main microprocessor, the LCC chips and the Ethernet processor. In embodiments where there are also third and/or fourth ports, the arbitration circuitry also handles contentions for access to high speed memory from these interfaces as well. Collisions of access request can be resolved by conventional reservation schemes, contention resolution schemes, polling schemes, fixed time slots of fixed "pecking order" type schemes such as where a microprocessor having second position on a pecking order is granted access until a higher pecking order microprocessor requests access at which time the lower pecking order microprocessor must immediately relinquish control of the high speed memories address and data buses. The preferred methodology is the contention resolution scheme whereby one microprocessor desiring access is granted access for as long as the microprocessor needs access until another microprocessor requests simultaneous access at which time the conflict is resolved by any contention resolution scheme such as fixed priorities etc.

The amount of memory needed for high speed memory system 800 to only do Ethernet switching to bridge packets between the various media such as media 805 and media 820 with no backbone port 828 is one megabyte of 20 nanosecond access time SRAM. In this situation, maximum traffic volume situation is 6 Ethernet ports carrying inbound traffic and 6 Ethernet ports carrying outbound traffic. Such a situation would involve a maximum of 90,000 packets per second. To do this Ethernet-to-Ethernet switching coupled with FDDI switching requires that memory system 800 have two megabytes of 20 nanosecond access time SRAM because approximately 150,000 packets per second need to be processed to achieve adequate performance levels. To do ATM switching requires that memory system 800 have four megabytes of 20 nanosecond access time SRAM. The required switching speed is achieved by having the memory system 800 be so much faster than the microprocessors such as Ethernet microprocessor 804 and main microprocessor 806 that it looks to the microprocessor like it has a piece of the high speed memory system 806 all to itself.

Typically, the receive buffer for each Ethernet media such as media 805 has 50 address locations, each of which can store one Ethernet packet of approximately 1,500 bytes length. If 12 Ethernet ports are all filling their buffers, practically all of one megabyte is filled. FDDI packets are longer however, being on the order of 4,500 bytes each. Therefore the addition of the FDDI adapter circuit 828 requires additional memory to support the longer length packets and higher traffic volume. ATM packets are only 53 bytes long, but these packets get concatenated. Also, ATM backbones require additional memory to support emulation mode where all the ATM network is made to look like an Ethernet to machines wishing to communicate over the ATM network.

The genus of packet switching machines represented by FIG. 7 is substantially faster and therefore better than other

prior ad packet switching technologies using high speed buses and buffer copy operations. Typically, these prior art packet switching machines use a single high speed bus to which are coupled a plurality of adapter circuits that couple the bus to each of a plurality of media such as Ethernet 10BaseT etc. When a packet is received on a first media, the packet is copied into a buffer memory on the adapter circuit coupling that media to the high speed bus. If the packet is addressed to a location on another media, the entire packet needs to be copied into the buffer of the adapter circuit coupled to the media on which the destination address resides (hereafter the target buffer). The packet is then copied out of the target buffer by the circuitry that drives the packet data onto the media to which the machine having the packet's destination address is coupled. This buffer copy operation is done using the high speed bus, but the very act of having to copy the entire packet from one buffer to another and having to do that with all the packets that need to be bridged from one media segment to another substantially slows down the operation of these prior art packet switching machines.

Another way in which switching speed is increased in the machines of the genus represented by FIG. 7 is through use of a "cut through" mode. The above described mode of operation of receiving an entire packet, error checking it and then notifying the main microprocessor of the existence and location of the packet so the main microprocessor can start examining the packet and take appropriate action will be hereafter referred to as the "store and forward" mode. The "cut through" mode is faster than the "store and forward" mode for the following reasons. In cut through mode, instead of waiting for the entire packet to be received and placed in the receive buffer before notifying the main microprocessor, the main microprocessor is notified of the existence of the packet after only the header is received. In other words, when a packet starts arriving on any particular media, the bytes of the packet header are sequentially stored in the receive buffer assigned to the media upon which the packet is arriving. After the complete header has been received, the Ethernet microprocessor notifies the main microprocessor of the existence and receive buffer location of the header of packet currently being received and requests that the main microprocessor start processing the packet. The main microprocessor then accesses the header and makes a determination of what kind of a packet it is, i.e., whether it needs to be routed to the management process or is a data packet, and whether the packet needs to be routed or bridged. If the packet needs to be bridged or routed to another media, the main microprocessor then notifies the LCC or adapter circuit coupled to the media to which is coupled the machine having an address corresponding to the destination address of the packet. That LCC or adapter circuit then begins sequentially emptying out the bytes of the packet from the receive buffer in which it is stored using a pointer to the start of the packet received from the main microprocessor. During all this processing the bytes of the incoming packet are being constantly received and stored in the receive buffer even as bytes earlier received are being emptied out of the same receive buffer by the LCC or adaptor circuit coupled to the media coupled to the destination machine.

After the incoming packet has been completely received, the LCC that deposited the bytes of the packet into the receive buffer checks the complete packet for errors such as a framecheck error. If there were errors, the packet that contained the errors will be discarded, and the system falls back to the "store and forward" mode. The reason that the

system falls back to the "store and forward" mode is because there is probably some source of noise that the network is picking up that corrupted the packet just received and this source of noise is likely to have corrupted more than one packet. Therefore, since the portion of the corrupted packet just received which has been transmitted on the destination machine's media cannot be retrieved, to avoid further erroneous packets from being propagated onto other media, the system falls back to the "store and forward" mode. In this mode, the main microprocessor is not notified that a received packet exists and needs to be processed until the entire packet has been received and has been checked for errors and found to be error-free.

If no errors were found at the end of the packet reception in the "cut through" mode, the system continues in the cut through mode for all received packets to achieve maximum throughput. Generally, it has been found that Ethernet networks are so reliable that the cut through mode can be used most of the time with error-free operation.

In alternative embodiments where speed is not so critical, the main microprocessor may move the packet out of the receive buffer for the LCC of the media segment upon which the packet arrived and moves it to the transmit buffer assigned to the LCC coupled to the media segment upon which the packet is to be transmitted.

In the preferred embodiment, the LCC that ultimately transmits the packet will, upon successful completion of the transmission, set a bit in its descriptor indicating that the packet has been successfully transmitted. Then, either the Ethernet processor or the main microprocessor 806 will access the packet that has been transmitted and erase it from the receive buffer. The packet may not be literally erased in some embodiments. The addresses which the packet occupied may simply be indicated as available in a table kept in high speed memory 800 or on-board one of the microprocessors 804 or 806. This memory management process to keep track of available memory may be done by the main microprocessor 806 or the Ethernet processor 804, or, in some embodiments, by the LCC chips themselves.

A key aspect of the invention is design of a high speed memory system which has sufficient bandwidth, i.e., low enough access times and enough throughput so as to be able to accept up to 10 megabits/second traffic volume on each network media coupled through a MAU and LCC to the E bus so as to be able to receive at least 120 megabits/second on the aggregate over the E bus and route this traffic to the high speed backbone port at media rate while having enough memory bandwidth left over to allow the two or more microprocessors in the system to still be able to have access to the high speed memory for purposes of executing their programs without constriction. Media rate for both FDDI and Fast Ethernet are 100 megabits/second, and media rate for planned ATM systems is 155 megabits/sec. In the preferred embodiment, the bandwidth of the high speed memory is 1.2 gigabits/second. This aspect of the teachings of the invention is accomplished by making high speed memory a shared static RAM array which has multiple ports and bus arbitration for access from the multiple ports to the shared address, data and control lines of the memory chips themselves.

Referring to FIG. 8, there is shown a more detailed block diagram representing a species of machines built in accordance with and operating in accordance with the genus of the invention. The LCC's of FIG. 7 such as block 809 correspond to Sonic chips marked Sonic 1 through Sonic 12. The MAU chips of FIG. 7 such as block 807 are represented by blocks MAUI through MAU 12. The Ethernet media such as

media 805 are represented by the lines marked Port 1 through Port 12. The Ethernet processor 804 is implemented by a Motorola MC68EC040 microprocessor and the main processor 806 is also implemented by a Motorola MC68EC040 microprocessor. In an alternative embodiment, the functions of the main processor 806 and the Ethernet processor 804 could be combined and performed by a single more powerful processor such as the PowerPC RISC microprocessor, a Pentium microprocessor etc. It is preferred to use two microprocessors however so that load sharing can be accomplished to increase data throughput and performance of the system. The main microprocessor 806 stores data comprising its bridging and routing tables in dynamic random access memory 822 or in SRAM 800. Factory configuration and manufacture data is stored in EEPROM 801. This data is not accessible to the user and consists of serial number, board revision level, software version number, date of manufacture, configuration data. Nonvolatile RAM memory 803 stores user programmable configuration data such as at what baud rate the ports work, what addresses have been assigned and other things that are user configurable. Flash ROM 805 stores the programs for the main processor 806 and the Ethernet processor 804 that are listed in Appendix E, Parts I and II. Timers and front panel display circuits 807 are used in support of the user interface and management functions. Reset and watchdog timer circuit 809 resets the microprocessors when a system crash occurs so that the microprocessors clear themselves and start again at the top of their program loops. The SCC circuit is a serial communications controller for bidirectionally communicating data between the packet switching machine and the console. The circuitry of FIG. 8 is programmed to operate in the fashion described in FIGS. 9, 10A and 10B and 11.

Referring to FIG. 9, there is shown a conceptual diagram of the process carried out according to the teachings of the invention. The diagram of FIG. 9 assumes that the process is being carried out by an Ethernet processor and a main processor, although, it could be also carried out by a single processor doing the functions described for both the Ethernet processor and the main processor. FIG. 10, comprised of FIGS. 10A and 10B, is a flow chart of the general sequence of events in the handling of a packet in the store and forward mode. The reader should refer to FIGS. 9 and 10 jointly for purposes of the following discussion. References to a main processor and an Ethernet processor should be read as references to a single processor where single processor structures are being used to carry out the process of FIG. 10. The operating system kernel, block 841 is executed by the main processor 806 (not shown). A function of the kernel is to implement a round robin, time slot based sharing of processor power among three tasks. Those three tasks are represented by block 843 for the Packet Switching Task, block 845 for the SNMP or Simple Network Management Protocol agent and block 847 for the Console Process. The operating system kernel, the Packet Switching Task, the SNMP agent and the Console Process are all programs or suites of programs which control operations of the main microprocessor during their respective time slots or in any other manner such as polled (kernel polls tasks and awards control of buses and main microprocessor assets when a task says it has business to transact), on demand (kernel awards control of buses and main microprocessor assets when receives request from task) etc. Each of these three processes gets awarded a 100 millisecond time slot by the kernel to perform its task and can perform its task to completion or until the end of 100 milliseconds, whichever occurs first.

The kernel 841 may also provide functions that may be invoked by each of the three tasks to assist them in performing their tasks such as "read shared memory" or "write shared memory" etc.

In alternative embodiments, each task could be running simultaneously on its own microprocessor or each task could set a flag or generate an interrupt when it needs attention from the main processor so that processing by the main processor is allocated to tasks only when they ask for it. Obviously, the three tasks 843, 845 and 847 could also be implemented fully in hardware for even higher operating speeds or partially in hardware and partially in software.

Block 849 in FIG. 10A represents the award of a 100 millisecond time slot to the Packet Switching Task by the main processor in the preferred embodiment. For purposes of FIG. 10 and illustration of the flow of processing, it will be assumed that each of the three tasks 843, 845 and 847 will have some processing needs during their respective time slots, and these processing needs will be handled sequentially.

During the time allocated to the packet switching task 843, the Packet Switching Task polls queue 810 in shared memory to determine if any pointers to packets to be processed are waiting therein, as symbolized by block 851 in FIG. 10A. If there is a pointer to a packet waiting in some receive buffer, the switching task accesses the appropriate receive buffer, indexes into the header information and examines the header data. This examination of the header data tells the Packet Switching Task whether the packet is to be discarded, transmitted out on another media (a media will sometimes hereafter be referred to as a port) from the one the packet arrived on, routed to the SNMP agent etc. This processing is represented by block 853 in FIG. 10A. The Packet Switching Task then takes the pointer off the queue 810, as symbolized by block 855, and processes the packet pointed to by that pointer accordingly, as symbolized by block 855. The packet may be a management packet that needs to be directed to the SNMP agent. Blocks 863 and following explain how this process works. Alternatively, the packet being processed by the main microprocessor may be a data packet that needs to be transmitted out a different port to another machine. Block 857 in FIG. 10A represents a bridging process to handle this type packet where a packet arrives from a first machine on one port or media and must be retransmitted via another port or media to a different machine. To implement this process, and as symbolized by block 857, the Packet Switching Process places a pointer to the packet in the appropriate transmit buffer assigned to the LCC coupled to the media or port upon which the packet is to be retransmitted, as represented by path 857A in FIG. 9, and updates the packet's reference count. Updating the reference count involves the main microprocessor writing a reference count number into a reference count field in the packet stored in the receive buffer. This reference count number is equal to the number of transmit buffers in which a pointer to the packet has been stored thereby indicating how many ports on which the packet is to be transmitted. This reference count is used to aid in managing the memory usage of the receive buffer for maximum utilization, especially in situation where some ports have heavy traffic or bottlenecks and packets are piling up while other ports are able to transmit their packets without delay as soon as pointers thereto are placed in their transmit buffers. Without the reference count and the Free Queue buffer, individual dedicated blocks of memory would have to be allocated to each LCC for its transmit and receive buffer as in the embodiment of FIG. 7. This does not result in optimum utilization of memory locations of the shared memory 800.

In the specific example diagrammed in FIG. 9, it is assumed that the packet is to be retransmitted to some machine coupled to media/port 1, so the pointer to the packet is placed in the transmit buffer 870 for port 1. This in effect triggers the appropriate LCC to begin transmitting the packet since the LCC's regularly poll their transmit buffers, as symbolized by block 859. The transmit buffer is a queue that is assigned to the LCC during the initialization process by the Ethernet processor. The LCC knows exactly where to look in the shared memory when it polls its transmit buffer as the addresses included within the transmit buffer assigned to any particular LCC do not change. Block 857 also represents the process carried out by the main microprocessor in carrying out the process of assisting in freeing the memory locations in the receive buffers for re-use in storing new incoming packets. To carry out this process, the main microprocessor, after processing a received packet by transferring it to the SNMP agent by placing a pointer to it in the management queue 865 or placing a pointer to the packet in some transmit buffer, also places a pointer to the packet in a free queue 896. The free queue is used to store pointers to packets that have been scheduled for transmission by the main microprocessor. Another function of the Ethernet processor 804 is to poll the free queue 896 periodically and use the pointers stored therein to free for re-use the memory space consumed in the receive buffer(s) by packets pointed to by pointers in the free queue.

If a pointer is found in the transmit buffer, the LCC transmits the packet using the pointer to retrieve the bytes of the packet from the receive buffer of the LCC which received it where the packet is stored, as symbolized by block 861. The data communication paths in FIG. 9 implementing this transaction are symbolized by paths 851A, 853A and 857A corresponding the steps having like root reference numbers in the flow chart of FIG. 10A.

Of course, some packets need to be broadcast or multicast. This is determined from the header addressing information. If a packet is to be broadcast, a pointer to the packet is placed in every transmit buffer, whereas if a packet is to be multicast, a pointer to is placed in all the transmit buffers coupled to media or ports having machines coupled thereto having destination addresses in the range given in the multicast address.

Assume for the next part of the discussion that a packet has arrived that is a management packet, and is sitting in the receive buffer of the LCC that received it. Whenever the management packet has been completely received, the Ethernet processor 804 places a pointer to it in the queue 810. Then, the next time the Packet Switching Task runs, the main preset, set will see the pointer to the management packet in the queue and examine the MAC layer address and realize that the packet is a management packet because the MAC layer address will indicate the SNMP agent 645 as the destination. The Packet Switching Task then places a pointer to the management packet in a portion of a management queue 865 in FIG. 9 devoted to pointers to management packets. Then the pointer to the management packet is removed from queue 810. All this processing is symbolized by block 863 in FIG. 10A and paths 851A, 853A and 883A in FIG. 9.

Block 865 in FIG. 10A, represents the process carried out by the main microprocessor in allocating a time slice to the SNMP agent/process 845 in FIG. 9. The SNMP agent block 845 really represents an SNMP agent as well as a stack of IP protocols that serve to decode the IP portion of the address of the packet and strip off the portions of the address that will not be understood by the SNMP agent. The remaining

portion of the packet, which will be referred to as the management portion of the packet, is forwarded to the SNMP agent for execution. The SNMP agent 845 in FIG. 9 then executes whatever request is embodied in the management packet. Such requests could include enabling or shutting down a port, reconfigure a port, gather traffic information etc. All management packets will come in through one of the ports from an external source. All this processing regarding receiving the management packet and getting it to the SNMP agent is symbolized by block 867 in FIG. 10A.

If the management packet requests information, the SNMP agent gathers that information and assembles a reply packet in an outgoing management packet reply buffer 866 in FIG. 9 as symbolized by block 869 in FIG. 10B and path 869A in FIG. 9. Block 869 in FIG. 10B also represents the process of placing a pointer to the management reply packet into the appropriate transmit buffer, as symbolized by path 869B in FIG. 9 in this hypothetical, it is assumed that the reply packet is to be sent to some machine coupled to port/media 2 since the pointer to the reply packet is placed by the SNMP agent into the #2 transmit/buffer 868. After the pointer is placed in the transmit buffer for the appropriate port, the LCC assigned to that port will find the pointer in its transmit buffer during polling thereof and begin transmitting the packet.

The Ethernet processor 804 in FIG. 1 is assigned to allocate memory in the shared, multiport, high-speed memory 800 for the receive buffer 871 and to program the LCC's so that they know where their respective portions of the receive buffer 871 are located in the preferred embodiment shown in FIG. 9, only one block of memory is allocated for the receive buffer 871, and each LCC uses whatever portion of this block is indicated to be free by data stored in a Free Queue 896 to be discussed further below. The embodiment of FIG. 9 differs from the embodiment of FIG. 7 in that in FIG. 7 there is a dedicated block of memory for the receive buffer and the transmit buffer for each LCC. The paths in FIG. 9 representing storage of received packets in the receive buffer 871 by LCC's #1 and 2 are paths 873 and 875. To store a received packet in the receive buffer 871, the LCC consults the Free Queue 896 to determine which portions of the receive buffer are free, and then stores the packet therein. The paths in FIG. 9 representing consulting the Free Buffer 896 for the location of free memory space by the LCCs are 901 and 903. The paths representing polling of the transmit buffers are 877 and 879 in FIG. 9.

Block 883 in FIG. 10B represents the process carried out by the main microprocessor under control of the operating system kernel of awarding a time slot to the console process 847 in FIG. 9. Block 885 then represents the process carried out by the Console Process driving the main microprocessor to carry out any necessary or requested command and/or control operation.

Referring to FIG. 11, there is shown a flow chart of the processing carried out by the Ethernet processor 804. The first task to be performed is represented by block 887. This block represents the process wherein the Ethernet processor allocates an adequate block of memory to accommodate both receive and transmit buffers for the number of LCC's present. In some embodiments, the Ethernet processor determines the number of LCC's present and actually connected to media before allocating memory for the buffers, and, in other embodiments, the Ethernet processor assumes that the number of LCC's present and connected to media is constant, and allocates memory adequate for receive and transmit buffers for all the LCC's. Next, in block 889, the Ethernet processor informs the LCC's where their transmit

and receive buffers are in memory. This can be done by, for example, writing length information and pointers to the start of the transmit and receive buffer for each LCC in the descriptor ring for that LCC. The LCC's then find out where their buffers are and the size thereof upon regular polling of their descriptor rings. Alternatively, the Ethernet processor can send messages directly the LCC's telling them the locations and sizes of their respective buffers.

Block 891 represents the process carried out by the LCC's in doing the following things: receiving packets and storing them in their respective receive buffers, transmitting packets pointed to by pointers in the transmit buffers of the LCCs, updating the packet reception status bits in the receive portions of their descriptor rings when packet reception starts and when it is completed, and updating status bits in the transmit portions of the descriptor rings each time a packet has been completely transmitted. These operations are symbolized by paths 873, 875, 876 and 878 in FIG. 9.

Block 893 represents the process carried out by the Ethernet processor 804 of monitoring the receive portions of the descriptor ring 808 for completion of successful packet reception and monitoring the transmit portions of the descriptor ring to determine when transmission of packets by each LCC has been successfully completed. These operations are symbolized by path 881 in FIG. 9. The descriptor rings 808 are portions of shared memory 800 which are used by the LCC's to store pointers to their receive and transmit buffers, CRC error and collision information, and bits, the logical state of which indicate when packet reception is starting and when it is finished and when transmission of a packet pointed to by a pointer in the transmit buffer of that LCC has been completed.

When the Ethernet processor finds a status bit in a receive portion of the descriptor ring in a state indicating that packet reception has been completed and the packet is correct, the Ethernet processor determines where the packet is in the receive buffer of the corresponding LCC and then writes a pointer to the location of that packet in the receive buffer 871 into the queue 810 of the main microprocessor, as symbolized by block 895. This transaction is represented by path 892 in FIG. 9. This function of the Ethernet processor essentially multiplexes the status bits of the 12 descriptor rings into a single location (queue 810) that the main microprocessor polls so that the main microprocessor does not have to poll 12 different descriptor rings itself. In alternative embodiments, the main microprocessor could poll all 12 descriptor locations on the descriptor linked list or ring itself and then locate the received packets in the receive buffers of any LCC's that have set status bits in their descriptor rings that indicate that a packet has been successfully received. In another embodiment, the descriptor ring could be a table instead of a linked list.

Block 895 also represents the process that the Ethernet processor performs when monitoring of the transmit portions of the descriptor ring indicates that a packet has been successfully transmitted, the Ethernet processor must determine whether the packet has been transmitted by all LCC's scheduled by the main microprocessor to send the packet before the Ethernet processor can mark that packet's storage locations as available to store new incoming packets. To do this, the Ethernet processor examines the reference count of the packet. This is done as follows. When the Ethernet processor discovers through monitoring the transmit portions of the descriptor ring that a packet transmission has occurred, the Ethernet processor reads the pointer to the packet in the transmit buffer of the LCC which indicated it had transmitted the packet. The Ethernet processor then

marks that location in the transmit buffer as available to store another pointer, and uses the pointer to access the packet. A specific field at the beginning or end of the packet stores a reference count. This is a number stored there by the main microprocessor which indicates how many ports on which the packet is scheduled to be transmitted. When the Ethernet processor determines from the descriptor ring that the packet has been successfully transmitted, the Ethernet processor reads the reference count and decrements it by one.

Block 897 is then performed by the Ethernet processor to determine if the reference count has reached zero. If not, processing returns to block 893 to continue monitoring the descriptor ring. If so, processing proceeds to the process of block 899 to mark the storage locations occupied by the bytes of the packet as available for storage of new packets. In carrying out this process, the Ethernet processor writes a pointer to the packet just transmitted into the Free Queue 896. As a result, the Free Queue serves as a map of all available memory storage locations in receive buffer 871. This permits optimum utilization of the storage capacity of the block of memory reserved for the receive buffer over the embodiment symbolized by FIG. 7 since some receive buffers will empty faster than others because of bottlenecks or high traffic volume on particular ports causing slower rates of transmission of packets out that port. Typically that will happen on ports coupled to servers or serving as backbone connections to hubs coupled to other high volume networks.

In the embodiments of FIG. 9, the receive buffer is comprised of a plurality of fixed size blocks of memory which are each large enough to store at least one packet of the maximum allowable length defined by the TCP/IP protocol. The pointers in the Free Queue therefore do not need to include length information and only need to point to the starting address of one of the blocks of predetermined length in the receive buffer. Because these fixed length blocks make programming simpler and the program executes faster, this approach represents a tradeoff of memory inefficiency for increased performance. Because some packets are smaller than the maximum allowable length, more efficient use of the memory could be made if the pointers in the Free Queue included both a starting address to the free block as well as the length of the block. In such an embodiment, there would be no blocks of predefined length, and each received packet would consume as much of the receive buffer as it needed. In such an embodiment, all pointers to packets would include both the starting address of the packet in the receive buffer as well as its length. This approach yields greater memory efficiency at the expense of performance.

Of course, in another alternative embodiment, the functions of FIGS. 10 and 11 could all be performed by a single microprocessor. Arbitration of contention for the ports of the shared high speed memory is accomplished in the subgenus of embodiments represented by FIGS. 8-11 in the same manner as it was accomplished in the subgenus of embodiments represented by FIG. 7. Specifically, a field programmable gate array (not shown) like FPGA 838 in FIG. 7 can be included as part of the high speed memory system and used to monitor for contention on the address, data and control pins of the memory chips in the high speed memory and award control thereof to one of the microprocessors.

DUAL PASSWORD SECURITY FEATURE

The following feature is applicable not only to provision of security for the configuration and password data on the hub with integrated bridge and packet switching machines disclosed here, but to any other password protected hard-

ware or software system as well. However, the discussion herein will be limited to protection of the hub with integrated bridge.

Generally speaking, the double password security feature allows a user or network administrator to set and alter configuration data using his or her password, but requires that a second user correctly enter a second password in order for the network administrator to alter his or her password. This prevents a network administrator who is being terminated from entering the system, shutting off the ports, changing user privileges or otherwise rendering the system less useable or inoperative and then changing his or her password unbeknownst to other employees so those other employees cannot get the system properly reconfigured after the network administrator is terminated. In some embodiments, the second employee cannot have access to the privilege and configuration data through the second password gateway.

In the preferred embodiment of the dual password security arrangement as symbolized by FIG. 12, two secure passwords and a master password are used. Referring to FIG. 12, a system administrator 910 can have access to some operational functionality 912 of the system he is administering by entering the correct Master Privilege Alteration Password (MPPW), as symbolized by line 914. The MPPW is entered through any user input device and is passed by an operating system (not shown) to a master privilege alteration password gateway function 916. Typically this function will be carried out by a software routine that controls a computer to compare the MPPW entered against a stored MPPW. If there is a match, access is granted to the desired functionality, as symbolized by line 918. In the case of the intelligent Hub and Packet Switching machines described herein, the functionality 912 is a routine to accept user input from the system administrator via paths 914 and 918 through the gateway to alter user privileges, turn ports on or off, or otherwise set or modify the machine configuration. Alternatively, in other contexts, the block 912 can represent the operational arena or main functionality of the system being controlled such as an operating system, financial reporting or accounting system, document or other file in any word processing, spreadsheet, database or other system to be operated, configured or controlled. If the MPPW gateway function 916 finds a mismatch between the MPPW password entered and the previously stored MPPW, access to the functionality 912 is blocked, as symbolized by block 920.

If the system administrator wishes to change the MPPW, two secret password gateways need to be satisfied. The first step in this process requires that system administrator enter a command or select a menu option requesting to change the MPPW. The computer programmed in accordance with the teachings of the invention then responds by asking the system administrator to enter a first secret password. Entry of this secret password #1 is symbolized by path 922. A first secret password gateway function 924 then compares the secret password #1 entered by the system administrator to a stored secret password #1 to which the system administrator has no access. If the password entered by the system administrator does not match the stored secret password #1, access to the function 926 to change the MPPW password is blocked, as symbolized by block 928 and path 930. If the password entered by the system administrator matches the stored secret password #1, path 932 is taken to the second secret password gateway functionality 934. The second secret password gateway 934 is a routine which controls the computer to ask for a second secret password which the

system administrator 910 does not know and to which he or she has no access. To satisfy this gateway, another user, which for this example will be called the supervisor 936 enters secret password #2, as symbolized by path 938. If this password matches a stored version of secret password #2, access to the function 926 to change the MPPW password is granted, as symbolized by path 940. If the password received by the second gateway 934 is incorrect, access to the function 926 to change MPPW is blocked, as symbolized by path 942. Thus, the system administrator can change privileges, alter the configuration etc. as long as he knows the MPPW password, but he cannot alter the MPPW without permission from the supervisor or unless he knows both secret passwords #1 and #2.

In alternative embodiments, secret password #2 gateway function 934 can impose a time limit on the time to enter secret password #2 or can impose a maximum limit on the number of incorrect attempts before access is blocked from all further attempts for a prolonged period of time or until the system is reset.

In the genus of packet switching machines, the double password security system described above can be implemented as part of any command and control process. For example, the double password security system can be implemented as part of step 885 in FIG. 10B. Likewise, the double password security system can be implemented as part of any management and control or console process in the genus of embodiments described herein having a hub with an integral bridge such as any of the "network slice" embodiments. For example, the double password security process could be implemented as part of the console command process 282 in FIG. 4.

In some embodiments of the double password security system, the computer which implements said system will have multiple terminals or will be a server computer in a network with multiple satellite computers coupled to said server computer through a hub and local area network segments. In such embodiments, the computer or server computer implementing the double password security system is programmed to assume that when access to shared assets on said computer or server such as shared files, shared programs or shared functions etc. is sought through a particular terminal or a particular satellite computer that a particular user is attempting the access as the computer assumes that particular users always use the same satellite computer or terminal. Each user has his or her own secret password that must be entered properly at a sign on screen to gain access to the shared assets on the computer. Thus, when access is sought through a particular terminal, the computer or server will assume that a particular user is logging on and ask for that user's password. In these embodiments, the function to change the master password can be one of the shared assets. To implement such an embodiment, the central computer or server computer is programmed to implement the two secret password gateways 924 and 934 on two separate satellite computers or terminals. To implement such an embodiment, if user 1 logs in on terminal 1 using the correct log on password for that user and requests to change the master password of the gateway to the system configuration or privileges file, the central computer or server is programmed to request entry of the first secret password on terminal 1 (or satellite computer 1) and request entry of the second secret password on terminal 2 (or satellite computer 2). Thus, a second user has to successfully log on on terminal 2 and then enter the correct second secret password before access to the function to change the master password will be granted. This embodi-

ment provides a third level of password security over the first and second secret password gateways. The following appendices are indicated in the application but not printed.

Appendix A hereto is the source code for the hub/bridge embodiments of FIGS. 1-6 and 12. This code is intended for the Motorola 68000 microprocessor, but may be ported to other platforms.

Appendix B hereto are the schematic diagrams for the hub/bridge embodiments of FIGS. 1-6 and 12.

Appendix C hereto is the Boolean logic for the programmable gate arrays for the hub/bridge embodiments of FIGS. 1-6 and 12.

Appendix D hereto is the Boolean logic for the field programmable gate array structure in the packet switching embodiments disclosed herein in FIGS. 7-12.

Appendix E, comprised of two portions, is a hexadecimal version of the code that controls the microprocessors in the preferred packet switching embodiment disclosed herein. Part I of Appendix E, comprised of pages 1-29, is the boot code that allows the code of Part II to be read into DRAM 822 in FIG. 8 from nonvolatile flash ROM 805 in FIG. 8 at boot time for execution from DRAM. Part II of Appendix E, comprised of pages 1 through 97, is the portion of the code that does the packet switching function. The main microprocessor 806 in FIG. 8 begins executing the code of Part I and determines therefrom that there is a portion of code from Part II that is intended for execution by the Ethernet processor 804 that needs to be loaded into the shared SRAM 800. The main microprocessor then loads the appropriate portion of the Part II code into SRAM 800 such that the Ethernet processor can start execution. The remaining portion of the Part II code is executed by the main microprocessor out of DRAM 822. Both Parts I and II are ported for the Motorola MC68EC040 microprocessor.

Although the invention has been described in terms of the preferred and alternative embodiments disclosed herein, those skilled in the art will appreciate other modifications which may be made without departing from the spirit and scope of the invention. All such modifications and enhancements are intended to be included within the scope of the claims appended hereto.

What is claimed is:

1. An apparatus for coupling to first and second networks upon which data packets are being transmitted on one or more data transmission media, comprising:
 - a physical support for supporting electronic circuitry;
 - a memory mounted on said physical support for storing data packets to be transmitted on either said first or second network and for storing data packets received from either said first or second network;
 - a bridge circuit mounted on said physical support and comprising at least first and second network interface circuits coupled to said first and second networks, respectively, each said network having data transmission media, said first and second network interfaces for converting data packets to be transmitted on said first and/or second networks which have been retrieved from said memory into signals capable of propagating on said data transmission media of said first and/or second networks, said first and second networks each coupling a one or more computing machines together for data exchange, each said computing machine having a network address, said first and second network interfaces also for receiving signals transmitted by one or more of said computing machines via said data transmission media of said first or second networks and

converting said signals into data packets for storage in said memory, each said data packet having a source network address which is the network address of the computing machine which transmitted said data packet and a destination network address which is the network address of the computing machine for which said data packet is bound, said bridge circuit for selectively forwarding a data packet received from a computing machine coupled to said first network to a computing machine coupled to said second network via said second network interface if the destination address of said data packet received from said computing machine coupled to said first network is the network address of a computing machine coupled to said second network or if said destination address of the received data packet is not known to be the network address of a computing machine coupled to either of said first or second networks, and for selectively forwarding a data packet received from a computing machine coupled to said second network to a computing machine coupled to said first network via said first network interface if the destination address of said data packet received from said computing machine coupled to said second network is the network address of a computing machine coupled to said first network or if said destination address of the received data packet is not the network address of a computing machine coupled to either of said first or second networks;

and wherein at least one of said network interface circuits includes a repeater mounted on said physical support and which has one or more ports each of which is coupled to one of said data transmission media forming part of the network coupled to said network interface circuit which includes said repeater, said repeater for receiving data packets at one or more of said ports and automatically retransmitting said data packets out from the remaining ones of said one or more ports of said repeater thereby causing propagation of said data packets on all other data transmission media coupled to said repeater regardless of the destination network address of any particular data packet being retransmitted;

and further comprising management means including a microprocessor, said management means mounted on said support structure and coupled to said bridge circuit and to said first and second network interface means, for receiving Simple Network Management Protocol commands and carrying out said commands, said microprocessor in said management means, said management means and said memory being shared by and supporting operations of said bridge circuit, and said first and second network interface circuits.

2. An apparatus for coupling to first and second networks upon which data packets are being transmitted on one or more media segments, comprising:

- a physical support for supporting electronic circuitry;
- a memory for storing data packets to be transmitted on either said first or second network and for storing data packets received from either said first or second network;
- a bridge circuit mounted on said physical support and comprising at least first and second network interface circuits coupled to said first and second networks, respectively, each said network having data transmission media, said first and second network interfaces for converting data packets to be transmitted on said first and/or second networks and retrieved from said

memory into signals to propagate on said data transmission media of said first and/or second networks, said first and second networks each coupling one or more computing machines together for data exchange, each said computing machine having a network address which is a source network address if said computing machine is transmitting data as signals on either said first or second network and is a destination network address if said computing machine is receiving data as signals from either first or second network, said first and second network interfaces also for receiving signals transmitted by one or more of said computing machines via said data transmission media of said first or second networks and converting said signals into data packets for storage in said memory, each said data packet having a source network address indicating the computing machine which transmitted said data and a destination network address indicating the computing machine for which said data packet is bound, said bridge circuit for selectively forwarding a data packet received from a computing machine coupled to said first network to a computing machine coupled to said second network via said second network interface if the destination address of said data packet received from said computing machine coupled to said first network is the network address of a computing machine coupled to said second network or if said destination address of the received data packet is not the network address of a computing machine coupled to either of said first or second networks, and for selectively forwarding a data packet received from a computing machine coupled to said second network to a computing machine coupled to said first network via said first network interface if the destination address of said data packet received from said computing machine coupled to said second network is the network address of a computing machine coupled to said first network or if said destination address of the received data packet is not the network address of a computing machine coupled to either of said first or second networks;

and wherein at least one of said network interface circuits includes a repeater mounted on said physical support and which has one or more ports each of which is coupled to a media segment forming part of the network coupled to said network interface circuit which includes said repeater, said repeater for receiving data packets at one or more of said ports and automatically retransmitting said data packets out on at least one other of said one or more ports of said repeater regardless of the destination network address of any particular data packet being retransmitted; and

a management circuit coupled to said bridge circuit, said repeater(s) and said network interface circuits, for receiving and carrying out management commands, said management circuit including a microprocessor which is part of said bridge circuit and which is also part of said first and second network interface circuits, said microprocessor being coupled to said bridge circuit and said first and second network interface circuits by a data bus and not a local area network segment, and wherein said memory is also part of said bridge circuit and said first and second network interface circuits, said memory being coupled to said bridge circuit and said first and second network interface circuits by a data bus and not a local area network segment.

3. The apparatus of claim 2 wherein said management circuit implements the simple network management proto-

col and includes in-band management means for receiving in-band management data packets from either said first or second second network and carrying out any network management function identified by the data in said in-band management data packet, and further includes console command means for receiving management commands via a serial port or modem and carrying out said management commands.

4. The apparatus of claim 3 further comprising isolate means for selectively preventing any transfer of data packets from said first network to said second network, and wherein said isolate means includes means for directing any in-band management packets that arrive from either said first or second network to said management circuit for execution thereby even when said isolate means is active in preventing any data packet transfer between said networks.

5. An apparatus for coupling to first and second networks upon which data packets are being transmitted on one or more data transmission media, comprising:

- a physical support for supporting electronic circuitry;
- a memory mounted on said physical support for storing data packets to be transmitted on either said first or second network and for storing data packets received from either said first or second network;
- a bridge circuit mounted on said physical support and comprising at least first and second network interface circuits coupled to said first and second networks, respectively, each said network having data transmission media, said first and second network interfaces for converting data packets to be transmitted on said first and/or second networks which have been retrieved from said memory into signals capable of propagating on said data transmission media of said first and/or second networks, said first and second networks each coupling a one or more computing machines together for data exchange, each said computing machine having a network address, said first and second network interfaces also for receiving signals transmitted by one or more of said computing machines via said data transmission media of said first or second networks and converting said signals into data packets for storage in said memory, each said data packet having a source network address which is the network address of the computing machine which transmitted said data packet and a destination network address which is the network address of the computing machine for which said data packet is bound, said bridge circuit for selectively forwarding a data packet received from a computing machine coupled to said first network to a computing machine coupled to said second network via said second network interface if the destination address of said data packet received from said computing machine coupled to said first network is the network address of a computing machine coupled to said second network or if said destination address of the received data packet is not known to be the network address of a computing machine coupled to either of said first or second networks, and for selectively forwarding a data packet received from a computing machine coupled to said second network to a computing machine coupled to said first network via said first network interface if the destination address of said data packet received from said computing machine coupled to said second network is the network address of a computing machine coupled to said first network or if said destination address of the received data packet is not the network address of a computing machine coupled to either of said first or second networks;

and wherein at least one of said network interface circuits includes a repeater mounted on said physical support and which has one or more ports each of which is coupled to one of said data transmission media forming part of the network coupled to said network interface circuit which includes said repeater, said repeater for receiving data packets at one or more of said ports and automatically retransmitting said data packets out from the remaining ones of said one or more ports of said repeater thereby causing propagation of said data packets on all other data transmission media coupled to said repeater regardless of the destination network address of any particular data packet being retransmitted;

and further comprising management means mounted on said support structure and coupled to said bridge circuit and to said first and second network interface means, for receiving Simple Network Management Protocol commands and carrying out said commands, and

wherein said bridge circuit includes means for selective deactivation of bridging activity in a bypass mode which, when active, causes all data packets received from said first network to be retransmitted on said second network and which causes all data packets received from said second network to be retransmitted on said first network, and further comprising isolate means for selectively preventing any transfer of data packets from said first network to said second network and which prevents any transfer of data packets from said second network to said first network.

6. An apparatus for coupling to first and second networks upon which data packets are being transmitted on one or more data transmission media, comprising:

- a physical support for supporting electronic circuitry;
- a memory mounted on said physical support for storing data packets to be transmitted on either said first or second network and for storing data packets received from either said first or second network;
- a bridge circuit mounted on said physical support and comprising at least first and second network interface circuits coupled to said first and second networks, respectively, each said network having data transmission media, said first and second network interfaces for converting data packets to be transmitted on said first and/or second networks which have been retrieved from said memory into signals capable of propagating on said data transmission media of said first and/or second networks, said first and second networks each coupling a one or more computing machines together for data exchange, each said computing machine having a network address, said first and second network interfaces also for receiving signals transmitted by one or more of said computing machines via said data transmission media of said first or second networks and converting said signals into data packets for storage in said memory, each said data packet having a source network address which is the network address of the computing machine which transmitted said data packet and a destination network address which is the network address of the computing machine for which said data packet is bound, said bridge circuit for selectively forwarding a data packet received from a computing machine coupled to said first network to a computing machine coupled to said second network via said second network interface if the destination address of said data packet received from said computing machine coupled to said first network is the network address of

a computing machine coupled to said second network or if said destination address of the received data packet is not known to be the network address of a computing machine coupled to either of said first or second networks, and for selectively forwarding a data packet received from a computing machine coupled to said second network to a computing machine coupled to said first network via said first network interface if the destination address of said data packet received from said computing machine coupled to said second network is the network address of a computing machine coupled to said first network or if said destination address of the received data packet is not the network address of a computing machine coupled to either of said first or second networks;

and wherein at least one of said network interface circuits includes a repeater mounted on said physical support and which has one or more ports each of which is coupled to one of said data transmission media forming part of the network coupled to said network interface circuit which includes said repeater, said repeater for receiving data packets at one or more of said ports and automatically retransmitting said data packets out from the remaining ones of said one or more ports of said repeater thereby causing propagation of said data packets on all other data transmission media coupled to said repeater regardless of the destination network address of any particular data packet being retransmitted;

and further comprising management means mounted on said support structure and coupled to said bridge circuit and to said first and second network interface means, for receiving Simple Network Management Protocol commands and carrying out said commands,

and wherein said repeater is part of said first network interface circuit and includes an Attachment Unit Interface port from which are transmitted said data packets being transmitted from the other of said ports of said repeater and which can also receive data packets, and further comprising switch means for selectively coupling said Attachment Unit Interface port to said second network interface circuit such that data packets transmitted from said Attachment Unit Interface port propagate on said second network and data packets propagating on said second network are received by said Attachment Unit Interface port and are retransmitted by said repeater on all physical data transmission media of said first network, and wherein said bridge circuit includes bypass means for setting said switch means such that said Attachment Unit Interface port is coupled to said second network in a bypass mode such that any data packet arriving at said Attachment Unit Interface from either of said first or second networks is automatically repeated on the other network, and wherein said management means includes means for receiving in-band management data packets from computing machines coupled to either said first network or second network and carrying out said commands, and

wherein said bridge circuit includes means for selective deactivation of bridging activity in a bypass mode which, when active, causes all data packets received from said first network to be retransmitted on said second network and which causes all data packets received from said second network to be retransmitted on said first network, and further comprising isolate means for selectively preventing any transfer of data packets from said first network to said second network and which prevents any transfer of data packets from said second network to said first network, and

wherein said management means comprises in-band management means for receiving in-band management data packets via either said first or second network interfaces from computing machines coupled to either said first or second network and carrying out any network management function identified by the data in said in-band management data packets, and further comprising a serial port interface circuit mounted to said support for coupling either to a modem, a terminal or a personal computer for receiving management commands, and wherein said management means further comprises console command means for receiving management commands via said serial port and carrying out said management commands.

7. An apparatus for coupling to first and second networks upon which data packets are being transmitted on one or more media segments, comprising:

- a physical support for supporting electronic circuitry;
- a memory for storing data packets to be transmitted on either said first or second network and for storing data packets received from either said first or second network;
- a bridge circuit mounted on said physical support and comprising at least first and second network interface circuits coupled to said first and second networks, respectively, each said network having data transmission media, said first and second network interfaces for converting data packets to be transmitted on said first and/or second networks and retrieved from said memory into signals to propagate on said data transmission media of said first and/or second networks, said first and second networks each coupling one or more computing machines together for data exchange, each said computing machine having a network address which is a source network address if said computing machine is transmitting data as signals on either said first or second network and is a destination network address if said computing machine is receiving data as signals from either first or second network, said first and second network interfaces also for receiving signals transmitted by one or more of said computing machines via said data transmission media of said first or second networks and converting said signals into data packets for storage in said memory, each said data packet having a source network address indicating the computing machine which transmitted said data and a destination network address indicating the computing machine for which said data packet is bound, said bridge circuit for selectively forwarding a data packet received from a computing machine coupled to said first network to a computing machine coupled to said second network via said second network interface if the destination address of said data packet received from said computing machine coupled to said first network is the network address of a computing machine coupled to said second network or if said destination address of the received data packet is not the network address of a computing machine coupled to either of said first or second networks, and for selectively forwarding a data packet received from a computing machine coupled to said second network to a computing machine coupled to said first network via said first network interface if the destination address of said data packet received from said computing machine coupled to said second network is the network address of a computing machine coupled to said first network or if said destination address of the received data packet is not the network

address of a computing machine coupled to either of said first or second networks;

and wherein at least one of said network interface circuits includes a repeater mounted on said physical support and which has one or more ports each of which is coupled to a media segment forming part of the network coupled to said network interface circuit which includes said repeater, said repeater for receiving data packets at one or more of said ports and automatically retransmitting said data packets out on at least one other of said one or more ports of said repeater regardless of the destination network address of any particular data packet being retransmitted; and

a management circuit coupled to said bridge circuit, said repeater(s) and said network interface circuits, for receiving and carrying out management commands, and

wherein said apparatus includes means coupled to said bridge circuit, said network interface circuit and said repeater(s) for implementing the national standard Internet Protocol defined by RFC 791, the Internet Control Message Protocol defined by RFC 792, the Address Resolution Protocol defined by RFC 826, and the Reverse Address Resolution Protocol defined by RFC 903 as these protocols were defined and published by the IETF at the time this application was filed, and

wherein each said repeater drives an attachment unit interface port and twenty-six or fewer media access units, and further comprising a bypass circuit for bypassing said bridge circuit such that any data packet arriving from either of said first or second networks is automatically transmitted on the other network via the network interface circuit and all media access units connected thereto in addition to being retransmitted via the network interface circuit of the network from which said data packet originated and substantially all media access units coupled thereto.

8. An apparatus for coupling to first and second networks on which data packets are being transmitted on one or more media segments, comprising:

- a physical support for supporting electronic circuitry;
- a memory for storing data packets to be transmitted on either said first or second network and for storing data packets received from either said first or second network;
- bridge circuit mounted on said physical support and comprising at least first and second network interface circuits coupled to said first and second networks, respectively, each said network having data transmission media, said first and second network interfaces for converting data packets to be transmitted on said first and/or second networks and retrieved from said memory into signals to propagate on said data transmission media of said first and/or second networks, said first and second networks each coupling one or more computing machines together for data exchange, each said computing machine having a network address which is a source network address if said computing machine is transmitting data as signals on either said first or second network and is a destination network address if said computing machine is receiving data as signals from either first or second network, said first and second network interfaces also for receiving signals transmitted by one or more of said computing machines via said data transmission media of said first or second networks and converting said signals into

data packets for storage in said memory, each said data packet having a source network address indicating the computing machine which transmitted said data and a destination network address indicating the computing machine for which said data packet is bound, said bridge circuit for selectively forwarding a data packet received from a computing machine coupled to said first network to a computing machine coupled to said second network via said second network interface if the destination address of said data packet received from said computing machine coupled to said first network is the network address of a computing machine coupled to said second network or if said destination address of the received data packet is not the network address of a computing machine coupled to either of said first or second networks, and for selectively forwarding a data packet received from a computing machine coupled to said second network to a computing machine coupled to said first network via said first network interface if the destination address of said data packet received from said computing machine coupled to said second network is the network address of a computing machine coupled to said first network or if said destination address of the received data packet is not the network address of a computing machine coupled to either of said first or second networks;

and wherein at least one of said network interface circuits includes a repeater mounted on said physical support and which has one or more ports each of which is coupled to a media segment forming part of the network coupled to said network interface circuit which includes said repeater, said repeater for receiving data packets at one or more of said ports and automatically retransmitting said data packets out on at least one other of said one or more ports of said repeater regardless of the destination network address of any particular data packet being retransmitted; and

a management circuit coupled to said bridge circuit, said repeater(s) and said network interface circuits, for receiving and carrying out management commands, and

wherein said management circuit implements the Simplified Network Management Protocol defined in national standard RFC 1157 and the User Datagram Protocol defined in national standard RFC 768 as these protocols were defined and published by the IETF at the time this application was filed. The apparatus of claim 40 further comprising isolate means for selectively preventing any transfer of data packets from said first network to said second network, and wherein said isolate means includes means for directing any in-band management packets that arrive from either said first or second network to said management circuit for execution thereby even when said isolate means is active in preventing any data packet transfer between said networks.

9. An apparatus for coupling to first and second networks on which data packets are being transmitted on one or more data transmission media, comprising:

- a physical support for supporting electronic circuitry;
- a memory mounted on said physical support for storing data packets to be transmitted on either said first or second network and for storing data packets received from either said first or second network;
- a bridge circuit mounted on said physical support and comprising at least first and second network interface

Packet Filtering in the SNMP Remote Monitor

Controlling remote monitors on a LAN

William Stallings

William is president of Comp-Comm Consulting of Brewster, MA. This article is based on his recent book, SNMP, SNMPv2, and CMIP: The Practical Guide to Network Management Standards (Addison-Wesley, 1993). He can be reached at stallings@acm.org.

The Simple Network Management Protocol (SNMP) architecture was designed for managing complex, multivendor internetworks. To achieve this, a few *managers* and numerous *agents* scattered throughout the network must communicate. Each agent uses its own management-information database (MIB) of managed objects to observe or manipulate the local data available to a manager.

The remote-network monitoring (RMON) MIB, defined as part of the SNMP framework, provides a tool that an SNMP or SNMPv2 manager can use to control a remote monitor of a local-area network. The RMON specification is primarily a definition of a data structure containing management information. The effect, however, is to define standard network-monitoring functions and interfaces for communicating between SNMP-based management consoles and remote monitors. In general terms, the RMON capability provides an efficient way of monitoring LAN behavior, while reducing the burden on both other agents and management stations; see [Figure 1](#).

The accompanying text box entitled, "Abstract Syntax Notation One (ASN.1)" gives details on defining the communication formats between agents and managers.

The key to using RMON is the ability to define "channels"--subsets of the stream of packets on a LAN. By combining various filters, a channel can be configured to observe a variety of packets. For example, a monitor can be configured to count all packets of a certain size or all packets with a given source address.

To use RMON effectively, the person responsible for configuring the remote monitor must understand the underlying filter and channel logic used in setting it up. In this article, I'll examine this filter and channel logic.

The RMON MIB contains variables that can be used to configure a monitor to observe selected packets on a particular LAN. The basic building blocks are a data filter and a status filter. The data filter allows the monitor to screen observed packets based on whether or not a portion of the packet matches a certain bit pattern. The status filter allows the monitor to screen observed packets on the basis of their status (valid, CRC error, and so on). These filters can be combined using logical AND and OR operations to form a complex test to be applied to incoming packets. The stream of packets that pass the test is referred to as a "channel," and a count of such packets is maintained. The channel can be configured to generate an alert if a packet passes through the channel when it is in an enabled state. Finally, the packets passing through a channel can be captured in a buffer. The logic defined for a single channel is quite complex. This gives the user enormous flexibility in defining the stream of packets to be counted.

Filter Logic

At the lowest level of the filter logic, a single data or status filter defines characteristics of a packet. First, consider the logic for defining packet characteristics using the variables *input* (the incoming portion of a packet to be filtered), *filterPktData* (the bit pattern to be tested for), *filterpktDataMask* (the relevant bits to be tested for), and *filterPktDataNotMask* (which indicates whether to test for a match or a mismatch). For the purposes of this discussion, the logical operators AND, OR, NOT, XOR, EQUAL, and NOT-EQUAL are represented by the symbols $\&$, $+$, $-\&$, $_$, $=$, and $_$, respectively.

Suppose that initially, you simply want to test the input against a bit pattern for a match. This could be used to screen for packets with a specific source address, for example. In the expression in [Example 1\(a\)](#), you would take the bit-wise exclusive-OR of *input* and *filterPktData*. The result has a 1 bit only in those positions where *input* and *filterPktData* differ. Thus, if the result is all 0s, there's an exact match. Alternatively, you may wish to test for a mismatch. For example, suppose a LAN consists of a number of workstations and a server. A mismatch test could be used to screen for all packets that did not have the server as a source. The test for a mismatch would be just the opposite of the test for a match; see [Example 1\(b\)](#). A 1 bit in the result indicates a mismatch.

The preceding tests assume that all bits in the input are relevant. There may, however, be some "don't-care" bits irrelevant to the filter. For example, you may wish to test for packets with any multicast destination address. Typically, a multicast address is indicated by one bit in the address field; the remaining bits are irrelevant to such a test. The variable *filterPktDataMask* is introduced to account for "don't-care" bits. This variable has a 1 bit in each relevant position and 0 bits in irrelevant positions. The tests can be modified; see [Example 1\(c\)](#).

The XOR operation produces a result that has a 1 bit in every position where there is a mismatch. The AND operation produces a result with a 1 bit in every relevant position where there is a mismatch. If all of the resulting bits are 0, then there is an exact match on the relevant bits; if any of the resulting bits is 1, there is a mismatch on the relevant bits.

Finally, you may wish to test for an input that matches in certain relevant bit positions and mismatches in others. For example, you could screen for all packets that had a particular host as a destination (exact match of the DA field) and did not come from the server (mismatch on the SA field). To enable these more complex tests to be performed, use *filterPktDataNotMask*, where:

- The 0 bits in *filterPktDataNotMask* indicate the positions where an exact match is required between the relevant bits of *input* and *filterPktData* (all bits match).
- The 1 bits in *filterPktDataNotMask* indicate the positions where a mismatch is required between the relevant bits of *input* and *filterPktData* (at least one bit does not match).

For convenience, assume the definition in [Example 2\(a\)](#). Incorporating *filterPktDataNotMask* into the test for a match gives [Example 2\(b\)](#).

The test for a mismatch is slightly more complex. If all of the bits of *filterPktDataNotMask* are 0 bits, then no mismatch test is needed. By the same token, if all bits of *filterPktDataNotMask* are 1 bits, then no match test is needed. However, in this case, *filterPktDataNotMask* is all 0s, and the match test automatically passes *relevant_bits_differ*₀=0. Therefore, the test for mismatch is as in [Example 2\(c\)](#).

The logic for the filter test is summarized in [Figure 2](#). If an incoming packet is to be tested for a bit pattern in a portion of the packet, located at a distance *filterPktDataOffset* from the start of the packet, the following tests will be performed:

- Test #1: As a first test (not shown in the figure), the packet must be long enough so that at least as many bits in the packet follow the offset as there are bits in *filterPktData*. If not, the packet fails this filter.
- Test #2: Each bit set to 0 in *filterPktDataNotMask* indicates a bit position in which the relevant bits of the packet portion should match *filterPktData*. If there is a match in every desired bit position, then the test passes; otherwise the test fails.
- Test #3: Each bit set to 1 in *filterPktDataNotMask* indicates a bit position in which the relevant bits of the packet portion should not match *filterPktData*. In this case, the test is passed if there is a mismatch in at least one desired bit position.

A packet passes this filter if and only if it passes all three tests.

Why use the filter test? Consider that you might want to accept all Ethernet packets that have a destination address of "A5" but do not have a source address of "BB". The first 48 bits of the Ethernet packet constitute the destination address and the next 48 bits, the source address. [Example 3](#) implements the test. The variable *filterPktDataOffset* indicates that the pattern matching should start with the first bit of the packet; *filter PktData* indicates that the pattern of interest consists of "A5" in the first 48 bits and "BB" in the second 48 bits; *filter PktDataMask* indicates that the first 96 bits are relevant; and *filterPktDataNotMask* indicates that the test is for a match on the first 48 bits and a mismatch on the second 48 bits.

The logic for the status filter has the same structure as that for the data filter; see [Figure 2](#). For the status filter, the reported status of the packet is converted into a bit pattern. Each error-status condition has a unique integer value, corresponding to a bit position in the status-bit pattern. To generate the bit pattern, each error value is raised to a power of 2 and the results are added. If there are no error conditions, the status-bit pattern is all 0s. An Ethernet interface, for example, has the error values defined in [Table 1](#). Therefore, an Ethernet fragment would have the status value of $6(21+22)$.

Channel Definition

A channel is defined by a set of filters. For each observed packet and each channel, the packet is passed through each of the filters defined for that channel. The way these filters are combined to determine whether a packet is accepted for a channel depends on the value of an object associated with the channel (*channelAcceptType*), which has the syntax *INTEGER {acceptMatched(1), acceptFailed(2)}*.

If the value of this object is *acceptMatched(1)*, packets will be accepted for this channel if they pass both the packet-data and packet-status matches of at least one associated filter. If the value of this object is *acceptFailed(2)*, packets will be accepted to this channel only if they fail either the packet-data match or the packet-status match of every associated filter.

[Figure 3](#) illustrates the logic by which filters are combined for a channel whose accept type is *acceptMatched*. A filter is passed if both the data filter and the status filter are passed; otherwise, that filter has failed. If you define a pass as a logical 1 and a fail as a logical 0, then the result for a single filter is the AND of the data filter and status filter for that filter. The overall result for a channel is then the OR of all the filters. Thus, a packet is accepted for a channel if it passes at least one associated filter pair for that channel.

If the accept type for a channel is *acceptFailed*, then the complement of the function just described is used. That is, a packet is accepted for a channel only if it fails every filter pair for that channel. This would be represented in [Figure 3](#) by placing a NOT gate after the OR gate.

Channel Operation

The value of *channelAcceptType* and the set of filters for a channel determine whether a given packet is accepted for a channel or not. If the packet is accepted, then the counter *channelMatches* is incremented. Several additional controls are associated with the channel: *channelDataControl*, which determines whether the channel is on or off; *channelEventStatus*, which indicates whether the channel is enabled to generate an event when a packet is matched; and *channelEventIndex*, which specifies an associated event.

When *channelDataControl* has the value off, then, for this channel, no events may be generated as the result of packet acceptance, and no packets may be buffered. If *channelDataControl* has the value on, then these related actions are possible.

Figure 4 summarizes the channel logic. If *channelDataControl* is on, then an event will be generated if: 1. an event is defined for this channel in *channelEventIndex*; and 2. *channelEventStatus* has the value *eventReady* or *eventAlwaysReady*. If the event status is *eventReady*, then each time an event is generated, the event status is changed to *eventFired*. It then takes a positive action on the part of the management station to reenable the channel. This mechanism can therefore be used to control the flow of events from a channel to a management station. If the management station is not concerned about flow control, it may set the event status to *eventAlwaysReady*, where it will remain until explicitly changed.

Summary

The packet-filtering facility of RMON provides a powerful tool for the remote monitoring of LANs. It enables a monitor to be configured to count and buffer packets that pass or fail an elaborate series of tests. This facility is the key to successful remote-network monitoring.

Abstract Syntax Notation One (ASN.1)

Steve Witten

Steve, a software engineer for Hewlett-Packard, specializes in network testing and measurement. You can contact him at stevewi@hpspd.spd.hp.com.

SNMP protocol and MIB are formally defined using an abstract syntax. This allowed SNMP's authors to define data and data structures without regard to differences in machine representations. This abstract syntax is an OSI language called "abstract syntax notation one" (ASN.1). It is used for defining the formats of the packets exchanged by the agent and manager in the SNMP protocol and is also the means for defining the managed objects.

ASN.1 is a formal language defined in terms of a grammar. The language itself is defined in ISO #8824. The management framework defined by the SNMP protocol, the SMI, and the MIB use only a subset of ASN.1's capabilities. While the general principles of abstract syntax are good, many of the bells and whistles lead to unnecessary complexity. This minimalist approach is taken to facilitate the simplicity of agents.

Listings [One](#) through [Three](#) show an MIB, using a fictitious enterprise called SNMP Motors. [Listing One](#) is an ASN.1 module that contains global information for all MIB modules. [Listing Two](#), another

ASN.1 module, contains the definitions of specific MIB objects. Finally, Listing Three illustrates manageable objects.

Once data structures can be described in a machine-independent fashion, there must be an unambiguous way of transmitting those structures over the network. This is the job of the transfer-syntax notation. Obviously, you could have several transfer-syntax notations for an abstract syntax, but only one abstract-syntax/transfer-syntax pair has been defined in OSI. The basic encoding rule (BER) embodies the transfer syntax. The BER is simply a recursive algorithm that can produce a compact octet encoding for any ASN.1 value.

At the top level, the BER describes how to encode a single ASN.1 type. This may be a simple type such as an *Integer*, or an arbitrarily complex type. The key to applying the BER is understanding that the most complex ASN.1 type is nothing more than several simpler ASN.1 types. Continuing the decomposition, an ASN.1 simple type (such as an *Integer*) is encoded.

Using the BER, each ASN.1 type is encoded as three fields: a tag field, which indicates the ASN.1 type; a length field, which indicates the size of the ASN.1 value encoding which follows; and a value field, which is the ASN.1 value encoding.

Each field is of variable length. Because ASN.1 may be used to define arbitrarily complex types, the BER must be able to support arbitrarily complex encodings.

It is important to note how the BER views an octet. Each octet consists of eight bits. BER numbers the high-order (most significant) bit as bit 8 and the low-order (least significant) bit as bit 1. It's critical that this view be applied consistently because different machine architectures use different ordering rules.

Figure 1 RMON description.

Figure 2 Logic for the filter test.

Figure 3 Logic by which filters are combined for a channel whose accept type is acceptMatched.

Figure 4: Logic for channel filter.

```

procedure packet_data_match;
begin
  if (result = 1 and channelAcceptType = acceptMatched) or
     (result = 0 and channelAcceptType = acceptFailed)
  then begin
    channelMatches := channelMatches + 1;
    if channelDataControl = on
    then begin
      if (channelEventStatus = eventFired) and
         (channelEventIndex = 0) then generate_event;
      if (channelEventStatus = eventReady) then
        channelEventStatus := eventFired
      end;
    end;
  end;
end;

```

Example 1: Testing the input against a bit pattern for a match.

- ```

a) (input XOR filterPktData) = 0 --> match
b) (input XOR filterPktData) (does not equal) 0 --> mismatch
c) ((input XOR filterPktData) (and) filterPktDataMask) =
 0 --> match on relevant bits
 ((input XOR filterPktData) (and) filterPktDataMask) (does not equal)
 0 --> mismatch on relevant bits

```

**Table 1: Ethernet-interface error values.**

| Bit | Error                                        |
|-----|----------------------------------------------|
| 0   | Packet is longer than 1518 octets.           |
| 1   | Packet is shorter than 64 octets.            |
| 2   | Packet experienced a CRC or alignment error. |

**Example 2:** Assuming the definition in (a), incorporating filterPktDataNotMask into the test for a match, you end up with (b). Test for a mismatch is shown in (c).

- ```

(a) relevant_bits_different =
    (input XOR filterPktData) (and) filterPktDataMask
(b) (relevant_bits_different (and) filterPktDataNotMask') =
    0 --> successful match
(c) ((relevant_bits_different (and) filterPktDataNotMask) (does not equal) 0) +
    (filterPktDataNotMask = 0) --> successful mismatch

```

Example 3: Launching a filter test.

```

filterPktDataOffset = 0
filterPktData       = "00 00 00 00 00 A5 00 00 00 00 00 BB"h
filterPktDataMask   = "FF FF FF FF FF FF FF FF FF FF FF FF"h
filterPktDataNotMask = "00 00 00 00 00 00 FF FF FF FF FF FF"h

```

Listing One

```

SNMP-motors-MIB DEFINITIONS ::= BEGIN
IMPORTS
    enterprises
        FROM RFC1155-SMI;
SNMP-motors OBJECT IDENTIFIER ::= { enterprises 9999 }
expr      OBJECT IDENTIFIER ::= { SNMP-motors 2 }
END

```

Listing Two

```

SNMP-motors-car-MIB DEFINITIONS ::= BEGIN
IMPORTS
    SNMP-motors
        FROM SNMP-motors-MIB;

```

```

IMPORTS
    OBJECT TYPE, ObjectName, NetworkAddress,
    IpAddress, Counter, Gauge, TimeTicks, Opaque
    FROM RFC1155-SMI;
car OBJECT IDENTIFIER ::= { SNMP-motors 3 }
-- this is a comment
-- Implementation of the car group is mandatory
-- for all SNMP-motors cars.
-- ( the rest of the SNMP-motors-car-MIB module )
END

```

Listing Three

```

carName OBJECT TYPE
    SYNTAX DisplayString (SIZE (0..64))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "A textual name of the car."
    ::= { car 1 }
carLength OBJECT TYPE
    SYNTAX INTEGER (0..100)
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The length of the car in feet."
    ::= { car 2 }
carPassengers OBJECT TYPE
    SYNTAX INTEGER (0..4)
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of passengers in the car."
    ::= { car 3 }
carPassengerTable OBJECT TYPE
    SYNTAX SEQUENCE OF CarPassengerEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
        "A table describing each passenger."
    ::= { car 4 }
carPassengerEntry OBJECT TYPE
    SYNTAX SEQUENCE OF CarPassengerEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
        "A entry table describing each passenger."
    ::= { carPassengerTable 1 }
CarPassengerEntry ::= SEQUENCE {
    carPindex
        INTEGER,
    carPname
        DisplayString,
    carPstatus
        INTEGER
}
carPindex OBJECT TYPE

```

GTrace – A Graphical Traceroute Tool

Ram Periakaruppan, Evi Nemeth
University of Colorado at Boulder
Cooperative Association for Internet Data Analysis (CAIDA)
{ramanath, evi}@cs.colorado.edu

Abstract

Traceroute [Jacobson88], originally written by Van Jacobson in 1988, has become a classic tool for determining the routes that packets take from a source host to a destination host. It does not provide any information regarding the physical location of each node along the route, which makes it difficult to effectively identify geographically circuitous unicast routing. Indeed, there are examples of paths between hosts just a few miles apart that cross the entire United States and back, phenomena not immediately evident from the textual output of *traceroute*. While such path information may not be of much interest to many end users, it can provide valuable insight to system administrators, network engineers, operators and analysts. We present a tool that depicts geographically the IP path information that *traceroute* provides, drawing the nodes on a world map according to their latitude/longitude coordinates.

1. Introduction

Today's Internet has evolved into a large and complex aggregation of network hardware scattered across the globe, with resources accessed transparently with respect to their location, be it in the next room or on another continent. As the Internet becomes increasingly commercialized among many different corporate administrative entities, it is more difficult to ascertain the geographical routes that packets actually travel across the network. Knowledge of these geographical paths can provide useful insight to system administrators, network engineers, operators and analysts.

It is challenging to obtain the location for a given node of a path since there is no existing database that accurately maps hostnames

or IP addresses to physical locations. Although RFC 1876 [RFC1876] defined a DNS resource record to carry such location information (the LOC record) for hosts, networks and subnets, very few sites maintain LOC records. Hence there is no straightforward way to determine the physical location of hosts.

GTrace is a graphical front end to *traceroute* that uses a number of heuristics to determine the location of a node. Often the name of a node in the path contains geographical information such as a city name/abbreviation or airport code. GTrace operates on the assumption that these codes and names indicate the physical location of the node. The locations obtained are connected together on a world map to show the geographical path that packets take from the source to destination host. GTrace also tries to verify the validity of each location obtained, eliminating ones that are incorrect.

The following sections review the *traceroute* tool and describe the design and implementation of GTrace. We also show example output from GTrace.

2. Traceroute

Traceroute is a tool that discovers the route an IP datagram takes through the Internet from a source host to a destination host. It works by exploiting the TTL (Time To Live) field of the IP Header. Each router that handles an IP datagram decrements the TTL field. When the TTL reaches zero, a router must discard the packet and send an error message to the originator of the datagram.

Traceroute uses this feature, initially sending a datagram with the TTL set to one. The first router along the path, upon receiving the datagram decrements the TTL, discards the datagram and sends back an ICMP error

message. *Traceroute* records this first IP address (source address of the error message packet) and then sends the next datagram with the TTL set to two. This process continues until the datagram finally reaches the target host, or until the maximum TTL threshold is reached.

3. Design and Implementation of GTrace

Recognizing that it is not possible to obtain precise physical location information for all existing IP addresses, our main design criteria for GTrace was that it be sufficiently flexible to support the addition of new databases and heuristics. We chose to implement GTrace in Java, for both its portability and its new Swing [Swing] user interface toolkit. GTrace operates in two phases. In the first phase GTrace executes *traceroute* to the destination host and tries to determine locations for each node along the path.

During the second phase, GTrace verifies whether the locations obtained in the previous phase are reasonably correct.

GTrace is composed of the following seven key components: Graphical User Interface, Dispatcher Thread, Hop Threads, Lookup Client, NetGeo Server, Lookup Server and Location Verifier. Fig. 1 illustrates the overall architecture of the tool. The function of each component is described below.

3.1 Graphical User Interface

The Main Thread handles all features of the Graphical User Interface and is responsible for spawning the dispatcher thread when a destination host is specified. Fig. 2 shows a snapshot of GTrace on startup. The GUI has two sections, with a map on the top and traditional

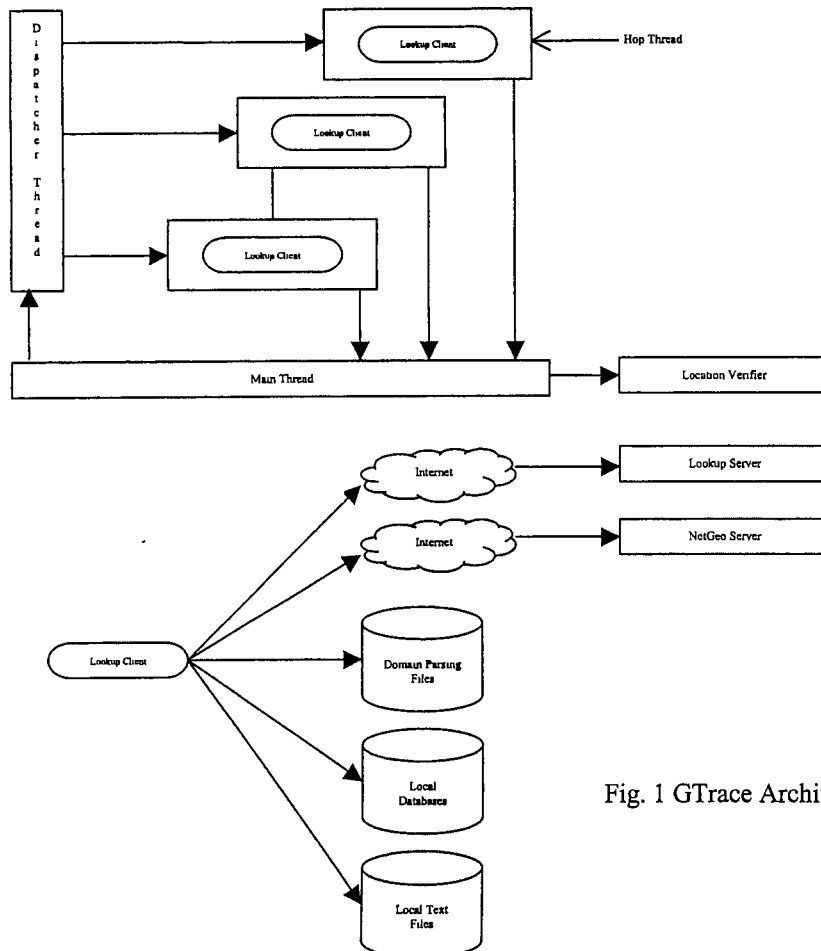


Fig. 1 GTrace Architecture

traceroute output below. The tool supports zooming in or out of particular regions of the maps. Twenty-three maps are available courtesy of VisualRoute [VisualRoute] and users can also add their own. We later provide an example that highlights some of the features of the GUI.

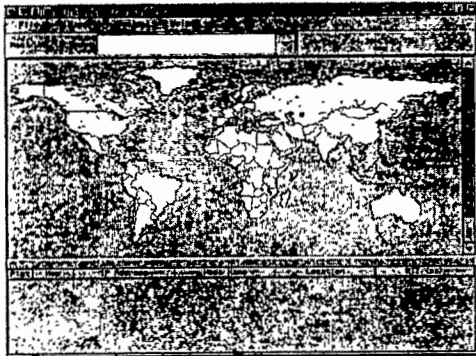


Fig. 2 GTrace's startup screen

3.2 Dispatcher Thread

The function of the dispatcher thread is to execute *traceroute* to the destination host. It then reads the output of *traceroute*, creating a new thread for each line of output. These threads are referred to as *hop threads*. The dispatcher thread can also read *traceroute* output from a file, which allows users to visualize traceroutes performed using third-party *traceroute* servers.

3.3 Hop Threads

Each hop thread parses its line of *traceroute* output and immediately notifies the main thread so that it can update the display with relevant *traceroute* fields for the corresponding hop. It then creates an instance of the Lookup Client, which tries to determine the location of the node and return the resulting information to the main thread before exiting.

3.4 Lookup Client

The Lookup Client tries to determine the location of a node by using a set of search heuristics. Many of the nodes in a typical *traceroute* path are in the ".net" domain. Often

the names of these nodes have some geographical hint in them. The Lookup Client uses customized domain parsing files that specify rules for extracting these geographic hints. We have such files for several ".net" domains that use internally consistent naming conventions within their domain.

However this technique does not solve the problem of locating nodes that do not have embedded geographical hints. GTrace also utilizes databases from CAIDA [DBCAIDA] and NDG Software [DBNDG] that map hostnames and IP addresses to latitude/longitude coordinates. For nodes with no information in these databases, the Lookup Client uses the domain's registered address (unfortunately often only the headquarters for a geographically distributed infrastructure) obtained through a *whois* lookup to determine the location. Nodes for which the Lookup Client is unable to determine a location are listed in the text portion, but skipped in the geographical display.

The search algorithm is described below. We try each heuristic in turn, stopping as soon as one yields a location. The Lookup Client also makes a note of the search step that produced the location, providing this information to the user as well as the Location Verifier.

Search Algorithm:

1. Check the cache to see if the location for the IP address has already been determined from a previous trace.
2. Check if the host has a DNS LOC record. If not, reduce the hostname to the next higher level domain (i.e., remove the first component of the name) and check again for a LOC record. Continue until we have reached the last meaningful component of the name (for example *foo.com* in *xxx.foo.com* or *bar.com.au* in *xxx.yyy.bar.com.au*). Note that if a site has a LOC record for the whole domain, but machines are located outside the scope of that LOC record, GTrace would end up using incorrect data. If the Location Verifier detects such a situation, GTrace will notify the user and optionally can be configured to notify GTrace's author, who will contact the DNS administrator at the corresponding site to correct their LOC records.

3. Search for a complete match of the hostname/IP address in the databases and files specified in the GTrace configuration file.
4. If the hostname has a corresponding domain parsing file, use the rules defined in the file to extract geographical hints and proceed as indicated in the file.
5. Reduce the hostname to the next higher level domain as in step 2 and search for a match as in step 3. The process is repeated until we have reached the last meaningful component of the name.
6. Query the NetGeo [NetGeo] server with the IP address. NetGeo determines the location based on *whois* registrant information.
7. If still no match occurs and the last two letters of the hostname end in a two-letter country code, map it to the geographic center of that country.

The search algorithm is ordered in decreasing level of location reliability. Locations obtained from steps 2 and 3 are taken as authoritative, while those from step 4 onward are considered a guess. Cache entries will indicate whether the location was authoritatively determined or was a guess; this status determines the color of the lines connecting the nodes on the map.

The Lookup Client does not determine locations for IP addresses that fall in the ranges 10.0.0.0 - 10.255.255.255, 172.16.0.0 - 172.31.255.255 or 192.168.0.0 - 192.168.255.255, as these blocks are reserved for private internet use [RFC 1918]. Unfortunately some addresses in these blocks do occur in traces since some ISPs use this address space for internal router interfaces. These nodes are shown in the text portion of the display with the location marked as private internet use.

The Lookup Client queries the Lookup Server if one is defined in the GTrace configuration file and if location information has not been obtained through step 1, 2 or 3 of the search algorithm. GTrace compares the reply from the Lookup Server with any obtained previously from local lookups, with preference given to the location obtained through a lower numbered search step. Based on the GTrace

configuration file, the Lookup Client also uses databases, text files and domain parsing files as follows.

Databases

The Lookup Client may need to perform lookups in many databases before determining a location. GTrace's database support is provided by the BerkeleyDB [BerkeleyDB] embedded database system, which supports a Java API that the Lookup Client uses to query the databases. The database interface allows multiple thread reads on the same database at the same time. Locking is not an issue, since Lookup Clients only read, do not write.

The following five databases are packaged with the GTrace distribution.

Machine.db [DBCAIDA]	Maps machine names to their latitude/longitude values.
Organization.db [DBCAIDA]	Maps organizations to their latitude/longitude values.
Hosts.db [DBNDG]	Maps IP addresses to their latitude/longitude values.
Cities.db [DBCAIDA]	Maps cities around the world to their latitude /longitude values.
Airport.db [AirportCodes]	Maps airport codes to their latitude/longitude values.

One can add a new database in BerkeleyDB format to GTrace with *GTraceCreateDB* and by adding an entry to the GTrace configuration file. The contents of the database ie., whether it maps hostnames, IP addresses, or both to latitude/longitude values, also have to be indicated in the configuration file. The user can also add records to existing databases using *GTraceAddRec*. *GTraceCreateDB* and *GTraceAddRec* are Java classes packaged with the GTrace distribution.

Text Files

Users may also specify new locations for nodes in text files, though it is more efficient to create a database for large data sets. New files have to be listed in the GTrace configuration file

in order for the search algorithm to have access to them.

Domain Parsing files

Files describing properties of each domain are used to ferret out geographical hints embedded in hostnames. These files define parsing rules using Perl5 compatible regular expressions. GTrace uses the regular expression library from ORO Inc. [OROMatcher] for parsing. New files can be added and existing ones modified without requiring any changes to GTrace.

For example, ALTER.NET (a domain name used by UUNET, a part of MCI/WorldCom) names some of their router interfaces with three letter airport codes as shown below:

```
193.ATM8-0-0.GW2.EWR1.ALTER.NET  
(EWR -> Newark, NJ)
```

```
190.ATM8-0-0.GW3.BOS1.ALTER.NET  
(BOS -> Boston, MA)
```

```
198.ATM6-0.XR2.SCL1.ALTER.NET  
(Exception)
```

```
199.ATM6-0.XR1.ATL1.ALTER.NET  
(ATL -> Atlanta, GA)
```

Fig. 3 shows an example of a GTrace domain parsing file that would work for ALTER.NET hosts. The file first defines the regular expressions, followed by any domain specific exceptions. The exceptions are strings that match the result of the regular expressions. The user may identify the exception's location either by city or by latitude/longitude value using the format shown below:

```
exception=city,state,country  
          city,country  
          L: latitude, longitude
```

In the former case, the user should also use *GTraceQueryDB* to ensure that the cities database has a latitude/longitude entry for the city specified. The first line in Fig. 3 defines a substitution operation, which when matched against 193.ATM8-0-0.GW2.EWR1.ALTER.NET, would return "EWR". The contents following the last "/" of the first line indicate what to do with a successful match, namely in

this case to instruct the program to first check for a match in the data specified in the current file and then for a match in the airport database.

```
s/. *?\.([\.\.]+)\d\.\ALTER\.\NET/$1/this,airport.db  
scl=santaclara, ca, us  
tco=tysonscorner, va, us  
noi=neworleans, la, us
```

Fig. 3 Example of a domain parsing file for ALTER.NET.

The reason for checking the domain parsing file first is that sometimes the naming scheme for a given domain is not consistent. For example, a search for SCL obtained from 198.ATM6-0.XR2.SCL1.ALTER.NET in the airport database would return a location for Santiago de Chile. In the case of ALTER.NET, they also use three letter codes that are not airport codes but abbreviations for US cities (Fig. 3 illustrates three such abbreviations.) Note that if this exception list were not present and SCL did get mapped to Chile, the Location Verifier would likely have eliminated it using the Round Trip Time (RTT) heuristic described later, which would have recognized the RTT as much too small to get a packet to Chile and back.

Sometimes ISPs name their hosts with more than one geographical hint in them. For example VERIO.NET names some of their hosts in the following format: den0.sjc0.verio.net, which typically suggests source and destination of the interface. If there is no rule on whether the convention is to use the source or destination label first in the hostname, the rule could be defined to extract both and GTrace could use the Location Verifier's heuristics to guess.

The advantage of this technique is that one can describe an entire domain as a set of rules without needing database entries for every host in the domain. The limitation of the technique is that it will fail for domains that do not use internally consistent naming schemes.

3.5 NetGeo Server

The original design of the Lookup Client performed and parsed results of *whois* lookups directly, which required storage of a prohibitively large number of mappings of world

locations to latitude/longitude values. Distributing such a large database with GTrace was not ideal. CAIDA's NetGeo [NetGeo] tool, with its ability to determine geographical locations based on the data available in *whois* records, provided a vital resource.

NetGeo is a database and collection of Perl scripts used to map IP addresses to geographical locations. Given an IP address, NetGeo will first search its own local database. If a record for the target address is found in the database, NetGeo will return the requested location information, e.g., latitude and longitude. If NetGeo finds no matching record in its database, it will perform one or more *whois* lookups until it finds a *whois* record for the appropriate network. The NetGeo Perl scripts will then parse the *whois* record and extract location information, which NetGeo both returns to the client and stores in its local database for future use.

The NetGeo database contains tables for mapping world location names (city, state/province/district, country) or US zip codes to latitude/longitude values. Most *whois* records provide enough address information for NetGeo to be able to associate some latitude/longitude value with the IP address. Occasionally the *whois* record only suggests a country or state, in which case NetGeo returns a generic latitude/longitude for that country or state. In preliminary testing, NetGeo has been able to parse addresses and find (albeit sometimes imprecise) latitude/longitude information for 89% of 17,000 RIPE *whois* records, 76% of 700 APNIC *whois* records and for more than 95% of 30,000 ARIN *whois* records.

3.6 Lookup Server

The Lookup Server handles requests from Lookup Clients and tries to determine the location of a host or IP address by executing steps 3, 4 and 5 of the search algorithm. This information is sent back to the client, which then decides whether to use the location information or not depending on the locations it might have received from other Lookup Servers or lookups it performed locally. The Lookup Client selects the location that was obtained from the lowest numbered search step.

The Lookup Server can also be requested by the Lookup Client to execute step 2

of the search algorithm. This is because not all versions of *nslookup* support queries for LOC records. GTrace tests the version of *nslookup* on the machine it is running on to determine if such a request is necessary.

3.7 Location Verifier

The Main Thread invokes the Location Verifier once all the hop threads have died and the trace is complete. The task of the Location Verifier is to check whether the locations obtained for nodes along the path are reasonable. The verifier does not determine new locations for nodes, it only indicates to the user why an existing location might be wrong and where the node could possibly be located.

The verifier algorithm is based on the fact that IP packets can not travel faster than the speed of light. Light travels across different mediums at different speeds: 3.0×10^8 m/s in vacuum, 2.3×10^8 m/s in copper and 2.0×10^8 m/s in fiber [Peterson]. GTrace uses the speed of light in copper for all of its calculations.

For each successive pair of hops that have locations, the verifier algorithm uses the deltas of the round-trip times (RTT) returned by *traceroute* to rule out locations that are physically not possible. *Traceroute* measures RTT rather than one way latency, as this would require control over both end nodes and delays are often not symmetric. Also, one must be cautious with the RTT values since they incorporate several components of delay. The RTT between two nodes has four components: the speed-of-light propagation delay, the amount of time it takes to transmit the unit of data, queuing delays inside the network and the processing time at the destination node to generate the ICMP time exceeded message. *Traceroute* typically sends 40-byte UDP datagrams, so it is safe to assume negligible transmit time. Ideally, for the verifier algorithm one would like the RTT to represent only the propagation delay, but this is not the case due to variable queuing and processing delays, hence it is not possible to set the upper bound on the RTT to a hop. Accordingly the verifier algorithm uses the minimum RTT returned by *traceroute*, as this would represent the best approximation of the propagation delay. Things are further complicated by the fact that the RTT delta between hops k and $k+1$ can be biased because

the return path the ICMP packet takes from hop k can be totally different from the return path it takes from hop $k+1$. The Location Verifier tries to re-determine RTT values for hops it thinks are biased using *ping*.

By default, *traceroute* sends three datagrams each time it increments the TTL to search for the next hop. Changing the value of the q parameter in the GTrace configuration file will modify this behavior. The larger the value of q , the more accurate the estimate of the propagation delay, but large values of q also slow down GTrace as *traceroute* has to send q packets for each hop.

Knowing the geographical distance between two nodes, GTrace can calculate the time-of-flight RTT (the propagation delay at the speed-of-light in copper), compare it against *traceroute's* value and flag a problem if the RTT is smaller than physically possible. In such a case either the location of the source or of the destination or both is incorrect. The details of the verification algorithm are as follows:

Verifier Algorithm:

1. Ideally, the RTT to hop k in a path should always be less than the RTT to hop $k+1$ or $k+2$... But this is not always true due to queuing delays, asymmetric paths and other delays. We allow a 1ms fudge factor to cover such discrepancies. Thus the RTTs between hops k and $k+1$ should be such that $RTT(k) \leq RTT(k+1) + 1ms$. If this condition does not hold true then the RTT to each of the out-of-order hops preceding hop k is estimated again with *ping*, i.e. till the first hop j preceding k such that $RTT(j) \leq RTT(k+1) + 1ms$. If the RTT estimates obtained using *ping* still do not satisfy the condition $RTT(k) \leq RTT(k+1) + 1ms$, then hop k is not used in the later stages of the verifier algorithm.
2. Cluster the *traceroute* path into regions having similar RTT values. This is based on the assumption that nodes with similar RTTs will tend to be in the same geographic region.
3. For each region identified in the previous step, calculate the time-of-flight RTT for pairs of hops that have locations. If the RTT

delta reported by *traceroute* for that pair of hops is smaller than the time-of-flight RTT, flag the pair of hops so that it is corrected in step 5.

4. Repeat step 3 for hops falling on the edges of adjacent regions.
5. Try to "correct" unreasonable location values that were identified in steps 3 and 4 using the reliability of the search step that produced the location match. Adjacent nodes between regions are corrected first because they represent larger and probably more inaccurate locations. Correcting the nodes identified in step 3 follows this. By correct, we mean trying different alternatives for the incorrect location based on the cluster in which it falls, flagging it to the user and not plotting it in the display.

Example:

Consider the trace shown in Fig. 4, where locations are expressed as city names for ease of illustration. The *Search Step* column indicates which step of the search algorithm produced the location for that hop. Step 1 of the verifier algorithm would mark hop 13 as unusable since its RTT is greater than its subsequent hops. In this case it is probably due to the return path from hop 13 being longer than that from hop 14. Next, step 2 of the algorithm would cluster the *traceroute* path into the following regions: 1-4, 5, 6-8, 9-10, 11-12 and 14-16. Step 3 would flag that there is a problem between hops 7 and 8 since it is not possible for a packet to travel from San Francisco to New Jersey in less than a millisecond. Likewise, step 4 would flag a problem between hops 10 and 11. Step 5 would first try to correct hops 10 and 11 since they fall in different regions. Seeing that the location for hop 11 was obtained through step 3 of the search algorithm and hop 10 was from a higher step, the Location Verifier would change hop 10's location to that of hop 11's, in this example to Washington and rerun the algorithm from step 3. This process is repeated until all locations from one hop to the next are physically realistic. In the end the Location Verifier would have indicated to the user that hop 8 is incorrect and is most probably located somewhere near San Francisco. Hops 9 and 10 are also incorrect and may be in Washington with their interfaces labeled San Francisco to

Hop	Node Name	IP Address	Search Step	Location	RTT (ms)
1	pinot-fe2-0-0	(192.172.226.65)	6	San Diego	0.917ms
2	medusa.sdsc.edu	(198.17.46.10)	3	San Diego	0.881ms
3	sdsc-gw.san-bb1.cerf.net	(192.12.207.9)	4	San Diego	1.944 ms
4	pos0-0-155M.san-bb6.cerf.net	(134.24.29.130)	4	San Diego	4.640 ms
5	atm6-0-1-622M.lax-bb4.cerf.net	(134.24.29.142)	4	Los Angeles	9.598 ms
6	pos6-0-622M.sfo-bb3.cerf.net	(134.24.29.233)	4	San Francisco	15.317 ms
7	pos10-0-0-155M.sfo-bb1.cerf.net	(134.24.32.86)	4	San Francisco	16.813 ms
8	192.205.31.29	(192.205.31.29)	6	New Jersey	16.917 ms
9	att-gw.sf.cw.net	(192.205.31.78)	4	San Francisco	81.281 ms
10	corerouter2.SanFrancisco.cw.net	(204.70.9.132)	4	San Francisco	81.254 ms
11	core1.Washington.cw.net	(204.70.4.129)	3	Washington	89.727 ms
12	mix1-fddi-0.Washington.cw.net	(204.70.2.14)	4	Washington	89.708 ms
13	vsnlpoone.Washington.cw.net	(204.189.152.134)	4	Poone	706.301 ms
14	202.54.6.17	(202.54.6.17)	6	Madras	697.946 ms
15	202.54.6.254	(202.54.6.254)	6	Madras	702.893 ms
16	giasmda.vsnl.net.in	(202.54.6.161)	4	Madras	704.856 ms

Fig. 4 A sample *traceroute* output produced by the first phase of GTrace.

identify the other end of that link.

describe their own intranet topology and then use GTrace as a graphical debugging tool within their network.

4. Configuration Files

The configuration options in GTrace are quite flexible. How it functions and executes the search algorithm depends on the contents of two configuration files: *GTrace.conf* and *GTraceMaps.conf*

4.1 GTrace.conf

GTrace.conf specifies the location of the commands GTrace uses and lists databases, text files, Lookup Servers if any, to use in the search algorithm. Fig. 5 shows an example configuration file. This file is automatically generated by the configure scripts while installing GTrace.

4.2 GTraceMaps.conf

The *GTraceMaps.conf* configuration file specifies attributes of the maps that GTrace uses in displays. Users can add their own maps as part of or independent from the existing world hierarchy. Independent maps allow users to

#GTrace configuration file

```
#Paths
TRACEROUTE=/usr/sbin/traceroute -q 3
WHOIS=/usr/bin/whois
PING=/usr/sbin/ping
NSLOOKUP=/usr/sbin/nslookup
DOMAINFILES=/home/ram/gtrace/data
DATABASES=/home/ram/gtrace/db
```

```
#Names of databases and text files to be used
#for location lookups. Order is important, list
#them in the order they should be searched.
CITIES=cities.db
AIRPORTS=airport.db
```

```
HOSTSLOC=Machine.db,hostnames/ipaddr;
Hosts.db,ipaddr;
Organization.db,hostnames/ipaddr;
```

```
TEXTFILES=England.txt,hostnames/ipaddr;
```

```
#Location of Lookup Servers if any
LOOKUPSRVS=
```

Fig. 5 Sample *GTrace.conf* file

5. GTrace Features

Fig. 6 shows an example of a trace that was executed from University of Colorado, Boulder to CAIDA in San Diego. On the display, the colors of the lines on the map indicate the

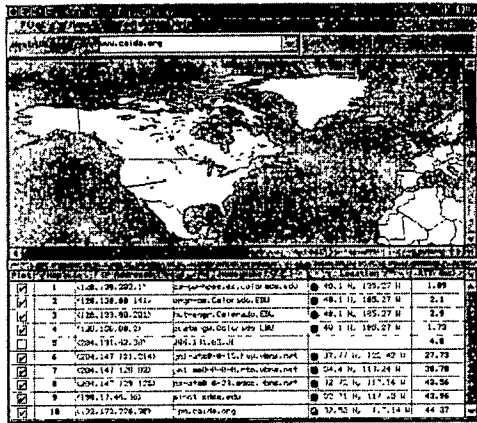


Fig. 6 Example of a trace produced by GTrace

reliability of the location obtained for the endpoints. The colors are decided based on the following criteria:

Green	Both endpoints are authoritative locations.
Yellow	One endpoint is authoritative and the other is a guess whose location is not a country center, state center or obtained from a <i>whois</i> record.
Blue	Both endpoints are guesses and the locations of both the endpoints are not a country center, state center or obtained from a <i>whois</i> record.
Red	One endpoint is a location that is a country center, state center or obtained from a <i>whois</i> record.

The table in the lower section of the display consists of six columns. The first column provides the user with a checkbox that is enabled for each location plotted on the map. The user can disable a checkbox and the corresponding location will be skipped. Locations that are flagged as unreasonable by the Location Verifier are not plotted by default.

The second, third and fourth columns display the hop number, IP address and host name respectively. Clicking on columns three

and four will bring up *whois* information for the node.

Column five provides the latitudes and longitudes obtained for each hop. Clicking on this column will provide an explanation of how the location was determined and whether the Location Verifier detected any problems. A small colored ball in front of the latitude and longitude value indicates which search step produced the location. The colors and the search step they represent are given below:

Green	Step 2 LOC record.
Yellow	Step 3 Complete match
Blue	Step 4 Domain parsing file
Cyan	Step 5 Hostname reduction match
Red	Step 6 <i>whois</i> record
Gray	Step 7 Country code

The last column shows the smallest of the round trip times returned by *traceroute*. The color of the value indicates how many packets timed out: black implies that no packets timed out, blue implies that one packet timed out, and a value in red indicates that two or more packets timed out.

6. Using GTrace in the Local Environment

System Administrators often use *traceroute* as a debugging tool to identify problems in their network. GTrace provides a visual representation that can facilitate understanding and debugging of their network. It can be used to discover routing loops as well as for deciding routes. For example in a large campus if a path from host A to host B (located in the same building) goes across campus and back, the routing could be fixed to avoid such inefficient paths. GTrace can also be useful from an end user perspective. Students can use the tool to work out the topology of their campus network.

7. Conclusion

GTrace is a handy tool for identifying network topology and routing problems as well as gaining more macroscopic insight into the Internet infrastructure. While GTrace uses several heuristics to determine locations and its

approach does not guarantee accuracy, it is robust and extensible. New databases, new Lookup Servers and learned insights into ISP's naming conventions can easily be added to GTrace. We hope that users and system administrators will find GTrace useful and contribute their own domain parsing files, or even run their own Lookup Servers for community use.

The practical success of GTrace lies in the rules defined for the ".net" domains, since these comprise the majority of hops in many traceroutes. Looking up a ".net" name in the *whois* database is only useful for small localized ISPs. Relying on *whois* heuristics would result in backbone providers' ".net" nodes to all uselessly map to a single corporate headquarters for that provider.

The accuracy of this tool would be much improved if the Internet community maintained LOC records in the DNS. Unfortunately since LOC records are optional, non-trivial in effort to support and without any clear payoff to ISPs, pervasive use of them will probably never occur and geographic visualization of arbitrary Internet infrastructure will continue to require heuristics to determine physical location of nodes.

8. Acknowledgments

We would like to thank kc claffy at CAIDA for suggesting the idea to develop this tool. We would also like to mention a special word of thanks to the following people and institutions: VisualRoute for permission to use their maps and labels, Sleepycat Software for the BerkeleyDB Package, Jim Donohoe for developing NetGeo and to the entire research team at CAIDA who helped with many aspects during the development of GTrace.

Several students (Colorado: Robert Cooksey, Brent Halsey, Jamey Wood, Jeremy Barga and UCSD: Jim Anderson) wrote graphical *traceroute* tools as class projects in Evi Nemeth's Network System's class. Many good ideas from these students' projects were incorporated into GTrace.

9. Availability and Support

GTrace-1.0 is the current release and it can be downloaded from the GTrace home page at <http://www.caida.org/Tools/GTrace>. The source code comes with the GTrace distribution. Further information on using the tool or how you can contribute domain parsing files can be found on the GTrace home page.

10. Author Information

Ram Periakaruppan is pursuing his Master's degree in Computer Science at the University of Colorado, Boulder. He can be reached at <ramanath@cs.colorado.edu>.

Evi Nemeth has been a computer science faculty member at the University of Colorado for years. Currently she is on leave doing the IEC (Internet Engineering Curriculum) project at CAIDA (Cooperative Association for Internet Data Analysis) on the UCSD campus and working furiously to make the publisher's deadline for the third edition of the UNIX System Administration Handbook. She can be reached at <evi@cs.colorado.edu>.

References

[AirportCodes] Listing of Airport Codes, <http://www.mapping.com/airportcodes.html>

[BerkeleyDB] BerkeleyDB Package Distribution, <http://www.sleepycat.com>

[DBCAIDA] Database files compiled by CAIDA, <http://www.caida.org/NetGeo/NetGeo/>

[DBNDG] Database file compiled by NDG Software, <http://www.dtek.chalmers.se/~d3august/xt/dl/>

[Jacobson88] Van Jacobson, Traceroute source code and documentation. Available from: <ftp://ftp.ee.lbl.gov/traceroute.tar.Z>.

[NetGeo] The Internet Geographic Database, <http://www.caida.org/Tools/NetGeo>

[OROMatcher] OROMatcher - Regular Expression Package for Java, <http://www.savarese.org>

[Peterson] Peterson, Larry L., & Davie, Bruce S.,
Computer Networks - A Systems Approach,
Morgan Kaufmann, (1996).

[RFC1876] RFC 1876, Davis, C., Vixie, P.,
Goodwin, T., and Dickinson I., A means for
Expressing Location Information in the Domain
Name System, January (1996).

[RFC1918] RFC 1918, Rekhter, Y., Moskowitz,
B., Karrenberg, D., Groot, G. J., Lear E.,
Address Allocation for Private Internets,
February (1996).

[Swing] Java Foundation Classes - Swing
<http://java.sun.com/products/jfc/>

[VisualRoute] Maps from VisualRoute,
<http://www.visualroute.com>

RECEIVED
CENTRAL FAX CENTER 002

APR 01 2004

#9
S. Sand
4/15/04

PTO/SB/122 (06-03)

Approved for use through 11/30/2005. OMB 0651-0035
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

CHANGE OF CORRESPONDENCE ADDRESS <i>Application</i> Address to: Commissioner for Patents P.O. Box 1450 Alexandria, VA 22313-1450	Application Number	09/608126
	Filing Date	30 Jun 2000
	First Named Inventor	Dietz
	Art Unit	2142
	Examiner Name	Thong Vu
	Attorney Docket Number	APPT-001-3

OFFICIAL

Please change the Correspondence Address for the above-identified patent application to:

Customer Number:

OR

<input type="checkbox"/> Firm or Individual Name			
Address			
Address			
City	State	Zip	
Country			
Telephone	Fax		

This form cannot be used to change the data associated with a Customer Number. To change the Data associated with an existing Customer Number use "Request for Customer Number Data Change" (PTO/SB/124).

I am the:

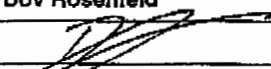
Applicant/Inventor

Assignee of record of the entire interest. Statement under 37 CFR 3.73(b) is enclosed. (Form PTO/SB/96).

Attorney or Agent of record. Registration number 38587

Registered practitioner named in the application transmittal letter in an application without an Executed oath or declaration. See 37 CFR 1.33(a)(1). Registration Number _____

Typed or Printed Name **Dov Rosenfeld**

Signature 

Date **April 1, 2004** Telephone **+1-510-547-3378**

NOTE: Signatures of all the inventors or assignees of record of the entire interest or their representative(s) are required. Submit multiple Forms if more than one signature is required, see below.

*Total of _____ forms are submitted.

This collection of information is required by 37 CFR 1.33. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 3 minutes to complete, including gathering, preparing and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comment on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner of Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

INVENTEK

Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618, USA
Phone: (510)547-3378; Fax: (510)291-2985
dov@inventek.com

Fax

RECEIVED
CENTRAL FAX CENTER

APR 01 2004

OFFICIAL

<i>Patent Application Ser. No.:</i> 09/608126	<i>Ref./Docket No.:</i> <u>APPT-001-3</u>
<i>Applicant(s):</i> Dietz, et al.	<i>Examiner.:</i> Thong Vu
<i>Filing Date:</i> June 30, 2000	<i>Art Unit:</i> 2142

FAX COVER PAGE

TO: Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

United States Patent and Trademark Office
(Examiner Thong Vu, Art Unit 2142)

Fax No.: 703-872-9306

DATE: April 01, 2004

FROM: Dov Rosenfeld, Reg. No. 38687

RE: Request for Change of Correspondence Address

Number of pages including cover: 2

Dear Madam/Sir,

I was told that the above referenced patent application had not been associated with my customer number and that the attached Change of Correspondence Address Form would rectify this. I would also like to request that this case be put into PAIR.

Respectfully,



Dov Rosenfeld, Reg. No. 38,687

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number 703-872-9306 addressed the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: April 1, 2004

Signed: 
Name: Amy Drury



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/608,126	06/30/2000	Russell S. Dietz	APPT-001-3	2145
21921	7590	04/16/2004	EXAMINER	
DOV ROSENFELD 5507 COLLEGE AVE SUITE 2 OAKLAND, CA 94618			VU, THONG H	
			ART UNIT	PAPER NUMBER
			2142	
			DATE MAILED: 04/16/2004	1 D

Please find below and/or attached an Office communication concerning this application or proceeding.

Interview Summary

Application No.

09/608,126

Applicant(s)

DIETZ ET AL.

Examiner

Thong H Vu

Art Unit

2142

All participants (applicant, applicant's representative, PTO personnel):

(1) Thong H Vu. (3) _____.

(2) Dov Rosenfeld#38,687. (4) _____.

Date of Interview: 15 April 2004.

Type: a) Telephonic b) Video Conference
c) Personal [copy given to: 1) applicant 2) applicant's representative]

Exhibit shown or demonstration conducted: d) Yes e) No.
If Yes, brief description: _____.

Claim(s) discussed: 1-21.

Identification of prior art discussed: Anderson and Chapman.


Agreement with respect to the claims f) was reached. g) was not reached. h) N/A.

Substance of Interview including description of the general nature of what was agreed to if an agreement was reached, or any other comments: Applicant discussed the invention in view of the prior art and suggested the amended claim via faxed on 4/14/04. Examiner will take to consider, make further search and response in due time.

(A fuller description, if necessary, and a copy of the amendments which the examiner agreed would render the claims allowable, if available, must be attached. Also, where no copy of the amendments that would render the claims allowable is available, a summary thereof must be attached.)

THE FORMAL WRITTEN REPLY TO THE LAST OFFICE ACTION MUST INCLUDE THE SUBSTANCE OF THE INTERVIEW. (See MPEP Section 713.04). If a reply to the last Office action has already been filed, APPLICANT IS GIVEN ONE MONTH FROM THIS INTERVIEW DATE, OR THE MAILING DATE OF THIS INTERVIEW SUMMARY FORM, WHICHEVER IS LATER, TO FILE A STATEMENT OF THE SUBSTANCE OF THE INTERVIEW. See Summary of Record of Interview requirements on reverse side or on attached sheet.

Examiner Note: You must sign this form unless it is an Attachment to a signed Office action.



Examiner's signature, if required

D. Sand
5/13/04

Approved for use through 10/31/2002, GMB 0651-0031
U.S. Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number

REQUEST FOR CONTINUED EXAMINATION (RCE) TRANSMITTAL Address to: Main Stop RCE Commissioner for Patents P.O. BOX 1450 Alexandria, VA 22313-1450	Application Number	09/608126
	Filing Date	30 Jun 2000
	First Named Inventor	Dietz et al.
	Art Unit	2142
	Examiner Name	Thong VU
	Attorney Docket Number	APPT-001-3

This is a Request for Continued Examination (RCE) under 37 CFR 1.114 of the above-identified application. Request for Continued Examination (RCE) practice under 37 CFR 1.114 does not apply to any utility or plant application filed prior to June 8, 1995, or to any design application. See Instruction Sheet for RCEs (not to be submitted to the USPTO) on page 2.

1. **Submission required under 37 CFR 1.114**

a. Previously submitted

i. Consider the amendment(s)/reply under 37 CFR 1.116 previously filed on _____
(Any unentered amendment(s) referred to above will be entered).

ii. Consider the arguments in the Appeal Brief or Reply Brief previously filed on _____

iii. Other _____

b. Enclosed

i. Amendment/Reply

ii. Affidavit(s)/Declaration(s)

iii. Information Disclosure Statement (IDS)

iv. Other Terminal Disclaimer

2. **Miscellaneous**

a. Suspension of action on the above-identified application is requested under 37 CFR 1.103(c) for a period of _____ months. (Period of suspension shall not exceed 3 months; Fee under 37 CFR 1.17(i) required)

b. Other _____

3. **Fees** The RCE fee under 37 CFR 1.17(e) is required by 37 CFR 1.114 when the RCE is filed.

a. Payment by credit card is enclosed (Form enclosed)

b. The Director is hereby authorized to charge any missing amounts for the following fees, or credit any overpayments, to Deposit Account No. 50-0292

i. RCE fee required under 37 CFR 1.17(e)

ii. Extension of time fee (37 CFR 1.136 and 1.17) [**\$110.00 for 1 month**]

iii. Other Terminal Disclaimer Fee (\$110)

c. Check in the amount of \$ _____ enclosed

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT REQUIRED			
Name (Print/Type)	Dov Rosenfeld	Registration No. (Attorney/Agent)	38,687
Signature		Date	Apr. 19, 2004

CERTIFICATE OF MAILING OR TRANSMISSION			
I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to: Mail Stop RCE, Commissioner for Patents, P.O. BOX 1450 Alexandria, VA 22313-1450, or facsimile transmitted to the U.S. Patent and Trademark Office on the date shown below:			
Name (Print/Type)	Amy Drury	Date	Apr. 19, 2004
Signature			

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND Fees and Completed Forms to the following address: Assistant Commissioner for Patents, Box RCE, Washington, DC 20231.

04/27/2004 SSANDARA 00000017 500292 09608126
01 FC:1801 770.00 DA

04/27/2004 SSANDARA 00000017 500292 09608126
03 FC:1251 110.00 DA

INVENTEK

Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618, USA
Phone: (510)547-3378; Fax: (510)291-2985
dov@inventek.com

Fax

RECEIVED
CENTRAL FAX CENTER

APR 19 2004

12 B

OFFICIAL
6/17/04

OUR REF: APPT-001-3

TO: Main Stop RCE
Commissioner for Patents
P.O. BOX 1450
Alexandria, VA 22313-1450

FAX No.: (703) 872-9306

DATE: April 19, 2004

FROM: Dov Rosenfeld, Reg. No., 38,687

RE: Notice of Appeal for Application No.: 09/608126

Number of pages including cover: 13

OFFICIAL COMMUNICATION
REQUEST FOR CONTINUED EXAMINATION (RCE)

Included herewith are:

- A Request For Continued Examination (RCE) with any attached response(s).
- Credit Card charge form

RECEIVED
APR 22 2004
DIP/JOHNS

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number (703) 872-9306 addressed to Commissioner for Patents, Mail Stop RCE, P.O. BOX 1450, Alexandria, VA 22313-1450 on.

Date: Apr. 19, 2004 Signed: Amy Drury
Name: Amy Drury

B

Our Ref./Docket No: APPT-001-3

Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

RECEIVED
CENTRAL FAX CENTERApplicant(s): Dietz, *et al.*

Group Art Unit: 2142

APR 19 2004

Application No.: 09/608126

Examiner: Vu, Thong H.

Filed: June 30, 2000

Title: RE-USING INFORMATION FROM DATA
TRANSACTIONS FOR MAINTAINING
STATISTICS IN NETWORK MONITORING

OFFICIAL

RESPONSE TO FINAL OFFICE ACTION ACCOMPANYING
REQUEST FOR CONTINUED EXAMINATIONMail Stop RCE
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

This is a response to the Final Office Action of Dec 23, 2003 accompanying a Request for Continued Examination (RCE).

Any *amendments to the specification* begin on a new page immediately after these introductory remarks.Any *amendments to the claims* begin on a new page immediately after such *amendments to the specification*, if any.Any *amendments to the drawings* begin on a new page immediately after such *amendments to the claims*, if any.The *Remarks/arguments* begin on a new page immediately after such *amendments to the drawings*, if any.If there are drawing amendments, an *Appendix* including amended drawings is attached following the *Remarks/arguments*.

S/N 09/608,126

Page 2

APPT-001-3

AMENDMENT(S) TO THE CLAIMS:

The following listing of claims will replace all prior versions, and listings, of claims on the application. All claims are set forth below with one of the following annotations.

- (Original): Claim filed with the application.
- (Currently amended): Claim being amended in the current amendment paper.
- (Canceled): Claim cancelled or deleted from the application. No claim text is shown.
- (Withdrawn): Claim still in the application, but in a non-elected status.
- (New): Claim being added in the current amendment paper.
- (Previously presented): Claim added or amended in an earlier amendment paper.
- (Not entered): Claim presented in a previous amendment, but not entered or whose entry status unknown. No claim text is shown.

-
1. (Currently amended) A method of analyzing a flow of packets passing through a connection point on a computer network, the method comprising:
- (a) receiving a packet from a packet acquisition device coupled to the connection point;
- (b) for each received packet, looking up a flow-entry database ~~that may contain~~ for containing one or more flow-entries for previously encountered conversational flows, the looking up to determine if the received packet is of an existing flow, a conversational flow including an exchange of a sequence of one or more packets in any direction between two network entities as a result of a particular activity using a particular layered set of one or more network protocols, a conversational flow further having a set of one or more states, including an initial state;
- (c) if the packet is of an existing flow, identifying the last encountered state of the flow, performing any state operations specified for the state of the flow, and updating the flow-entry of the existing flow including storing one or more statistical measures kept in the flow-entry; and

S/N 09/608,126

Page 3

APPT-001-3

(d) if the packet is of a new flow, performing any state operations required for the initial state of the new flow and storing a new flow-entry for the new flow in the flow-entry database, including storing one or more statistical measures kept in the flow-entry,

wherein every packet passing through the connection point is received by the packet acquisition device, and

wherein at least one step of the set consisting of of step (a) and step (b) includes identifying the protocol being used in the packet from a plurality of protocols at a plurality of protocol layer levels,

such that the flow-entry database is to store flow entries for a plurality of conversational flows using a plurality of protocols, at a plurality of layer levels, including levels above the network layer.

2. (Currently amended) A method according to claim 1, wherein step (b) includes including:

extracting identifying portions from the packet,

wherein the extracting at any layer level is a function of the protocol being used at the layer level, and

wherein the looking up uses a function of the identifying portions.

3. (Original) A method according to claim 1, wherein the steps are carried out in real time on each packet passing through the connection point.

4. (Original) A method according to claim 1, wherein the one or more statistical measures include measures selected from the set consisting of the total packet count for the flow, the time, and a differential time from the last entered time to the present time.

5. (Original) A method according to claim 1, further including reporting one or more metrics related to the flow of a flow-entry from one or more of the statistical measures in the flow-entry.

S/N 09/608,126

Page 4

APPT-001-3

6. (Original) A method according to claim 7, wherein the metrics include one or more quality of service (QOS) metrics.
7. (Original) A method according to claim 5, wherein the reporting is carried out from time to time, and wherein the one or more metrics are base metrics related to the time interval from the last reporting time.
8. (Original) A method according to claim 7, further comprising calculating one or more quality of service (QOS) metrics from the base metrics.
9. (Original) A method according to claim 7, wherein the one or more metrics are selected to be scalable such that metrics from contiguous time intervals may be combined to determine respective metrics for the combined interval.
10. (Previously presented) A method according to claim 1, wherein step (c) includes if the packet is of an existing flow, identifying the last encountered state of the flow and performing any state operations specified for the state of the flow starting from the last encountered state of the flow; and wherein step (d) includes if the packet is of a new flow, performing any state operations required for the initial state of the new flow.
11. (Original) A method according to claim 10, further including reporting one or more metrics related to the flow of a flow-entry from one or more of the statistical measures in the flow-entry.
12. (Original) A method according to claim 11, wherein the reporting is carried out from time to time, and wherein the one or more metrics are base metrics related to the time interval from the last reporting time.
13. (Original) A method according to claim 12, wherein the reporting is part of the state operations for the state of the flow.
14. (Original) A method according to claim 10, wherein the state operations include updating the flow-entry, including storing identifying information for future packets to be identified with the flow-entry.

S/N 09/608,126

Page 5

APPT-001-3

15. (Original) A method according to claim 14, further including receiving further packets, wherein the state processing of each received packet of a flow furthers the identifying of the application program of the flow.
16. (Original) A method according to claim 15, wherein one or more metrics related to the state of the flow are determined as part of the state operations specified for the state of the flow.
17. (Currently amended) A packet monitor for examining packets passing through a connection point on a computer network, each packets conforming to one or more protocols, the monitor comprising:
- (a) a packet acquisition device coupled to the connection point and configured to receive packets passing through the connection point;
 - (b) a memory for storing a database ~~that may contain~~ for containing one or more flow-entries for previously encountered conversational flows to which a received packet may belong, a conversational flow including an exchange of a sequence of one or more packets in any direction between two network entities as a result of a particular activity using a particular layered set of one or more network protocols, a conversational flow further having a set of one or more states, including an initial state; and
 - (c) an analyzer subsystem coupled to the packet acquisition device configured to lookup ~~for each packet~~ for each received packet whether a received packet belongs to a flow-entry in the flow-entry database, to update the flow-entry of the existing flow including storing one or more statistical measures kept in the flow-entry in the case that the packet is of an existing flow, and to store a new flow-entry for the new flow in the flow-entry database, including storing one or more statistical measures kept in the flow-entry if the packet is of a new flow,
- wherein the analyzer subsystem is further configured to identify the protocol being used in the packet from a plurality of protocols at a plurality of protocol layer levels,
and

S/N 09/608,126

Page 6

APPT-001-3

wherein the database is to store flow entries for a plurality of conversational flows using a plurality of protocols, at a plurality of layer levels, including levels above the network layer.

18. (Original) A packet monitor according to claim 17, further comprising:

a parser subsystem coupled to the packet acquisition device and to the analyzer subsystem configured to extract identifying information from a received packet,

wherein each flow-entry is identified by identifying information stored in the flow-entry, and wherein the cache lookup uses a function of the extracted identifying information.

19. (Original) A packet monitor according to claim 17, wherein the one or more statistical measures include measures selected from the set consisting of the total packet count for the flow, the time, and a differential time from the last entered time to the present time.

b1

20. (Original) A packet monitor according to claim 17, further including a statistical processor configured to determine one or more metrics related to a flow from one or more of the statistical measures in the flow-entry of the flow.

21. (Original) A packet monitor according to claim 20, wherein the statistical processor determine and reports the one or more metrics from time to time.

84

B

S/N 09/608,126

Page 7

APPT-001-3

REMARKS*Status of the Application:*

Claims 1-21 are the claims of record of the application. Claims 1-21 have been finally rejected in the Final Office Action of Dec 23, 2003.

Included herein is a Request for Continued Examination (RCE) with appropriate fees. This response to submitted with the RCE.

Telephone Interview on April 15, 2004

Applicants and the undersigned appreciate Examiner's attention and courtesy during the telephone interview held April 15, 2004 between the Examiner and the undersigned. The invention, the prior art, and the rejections were discussed. In the interview, an amendment to claim 1 was discussed. Agreement was reached that such an amendment would overcome the cited prior art. The examiner reported that he will take this amendment into consideration, make a further search, and report in due time. Such an amendment is included herein.

Amendment to the Claims:

Applicants have amended the claims to overcome the cited prior art by explicitly including that a conversational flow includes an exchange of a sequence of one or more packets in any direction between two network entities as a result of a particular activity using a particular layered set of one or more network protocols, that a conversational flow further has a set of one or more states, including an initial state, and that, with respect to claim 1, at least one of steps (a) or step (b) includes identifying the protocol being used in the packet from a plurality of protocols at a plurality of protocol layer levels, such that the flow-entry database is to store flow entries for a plurality of conversational flows using a plurality of protocols, at a plurality of layer levels, including levels above the network layer.

Applicants have further amended the claims to overcome the indefiniteness rejection.

Examiner's assertion of non-compliance with 37 CFR 1.111

In the interview, the compliance or not of the previous response to 37 CFR 1.111 was discussed and it was agreed that this is moot in view of the amendment submitted herein.

Claim Rejections - 35 USC § 112

In paragraph 8 of the final office action, the examiner asserts that Claims 1 and 17 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention (i.e.: a flow entry database that may contain one or more flow entries).

S/N 09/608,126

Page 8

APPT-001-3

Applicants have amended the phrase "flow entry database that may contain one or more flow entries" to "flow-entry database for containing one or more flow entries." The term "one or more" is an accepted synonym for "at least one." Thus, the rejections of claims 1 and 17 for being indefinite is believed overcome.

Claim Rejections - Double Patenting

In paragraph 9 of the final office action, the examiner has rejected claims 1-21 under the judicially created doctrine of double patenting over claims 1-10 of U. S. Patent No. 6,651,099 B1.

A terminal disclaimer is included with this response. Thus, the rejection under the judicially created doctrine of double patenting is overcome.

Claim Rejections - 35 USC § 102

In paragraph 10 of the final office action, the examiner has rejected claims 1-21 under 35 U.S.C. § 102(e) as being anticipated by Anderson et al [Anderson 5,850,388].

In paragraph 30 of the final office action, the examiner asserts that Claims 1-21 are rejected under 35 U.S.C. § 102(e) as being anticipated by Chapman et al [Chapman 6,330,226 B1].

In paragraph 59 of the final office action, the examiner asserts that Claims 1-21 are rejected under 35 U.S.C. § 102(e) as being anticipated by Bullard [6,625,657 B1].

Applicants have amended the claims to overcome these rejections. For example, with respect to claim 1, Applicants have explicitly states that a conversational flow includes an exchange of a sequence of one or more packets in any direction between two network entities as a result of a particular activity using a particular layered set of one or more network protocols, that a conversational flow further has a set of one or more states, including an initial state. None of the cited prior art deals with such a conversational flow. Furthermore, with respect to claim 1, Applicants have added that at least one of steps (a) or step (b) includes identifying the protocol being used in the packet from a plurality of protocols at a plurality of protocol layer levels, such that the flow-entry database is to store flow entries for a plurality of conversational flows using a plurality of protocols, at a plurality of layer levels, including levels above the network layer.

The Applicants have also amended claim 17 to include similar limitations.

None of the cited prior art includes this feature. Chapman and Bullard, for example, each only deal with IP packets, and with TCP flows.

Thus, the rejections are believed overcome, and claims 1 and 17 are believed allowable.

While Applicants do not admit that any of Examiner's arguments on the dependent claims are correct, such arguments are now moot. All dependent claims are also allowable.

For these reasons, and in view of the above amendment, this application is now considered to be in condition for allowance and such action is earnestly solicited.

S/N 09/608,126

Page 9

APPT-001-3

The present amendment is such that Bullard does not disclose or suggest the invention as claimed in claims 1 and 17. Note however that Bullard was filed March 25, 1999. Therefore, the claims presented herein would have to have been invented after March 25, 1999 for Bullard to be prior art under 35 USC 102(e). The inventor is prepared to swear behind this reference with a declaration that independent claims 1 and 17 were invented prior to the reference date, i.e., March 25, 1999. This is moot however, in view of the present amendment.

Conclusion

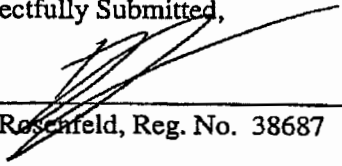
The Applicants believe all of Examiner's rejections have been overcome with respect to all remaining claims (as amended), and that the remaining claims are allowable. Action to that end is respectfully requested.

If the Examiner has any questions or comments that would advance the prosecution and allowance of this application, an email message to the undersigned at dov@inventek.com, or a telephone call to the undersigned at +1-510-547-3378 is requested.

Respectfully Submitted,

Apr. 19, 2004

Date


Dov Rosenfeld, Reg. No. 38687

Address for correspondence:

Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618
Tel. +1-510-547-3378
Fax: +1-510-291-2985
Email: dov@inventek.com

#13
M. East

Approved for use through 07/31/2006. OMB 0651-0031
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PTO/SB/26 (08-03)

**TERMINAL DISCLAIMER TO OBTAIN A DOUBLE PATENTING
REJECTION OVER A PRIOR PATENT**

Docket Number (Optional)
APPT-001-3

In re Application of: Dietz, et al.

Application No.: 09/608126

Filed: 30 Jun 2000

For: Re-using Information from Data Transactions for Maintaining Statistics in Network Monitoring

The owner, Hi/tn, Inc. of 100 percent interest in the instant application hereby disclaims, except as provided below, the terminal part of the statutory term of any patent granted on the instant application, which would extend beyond the expiration date of the full statutory term defined in 35 U.S.C. 154 and 173, as presently shortened by any terminal disclaimer, of prior Patent No. 6,651,099. The owner hereby agrees that any patent so granted on the instant application shall be enforceable only for and during such period that it and the prior patent are commonly owned. This agreement runs with any patent granted on the instant application and is binding upon the grantee, its successors or assigns.

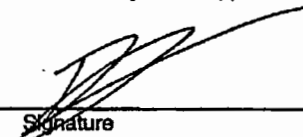
In making the above disclaimer, the owner does not disclaim the terminal part of any patent granted on the instant application that would extend to the expiration date of the full statutory term as defined in 35 U.S.C. 154 and 173 of the prior patent, as presently shortened by any terminal disclaimer, in the event that it later: expires for failure to pay a maintenance fee, is held unenforceable, is found invalid by a court of competent jurisdiction, is statutorily disclaimed in whole or terminally disclaimed under 37 CFR 1.321, has all claims canceled by a reexamination certificate, is reissued, or is in any manner terminated prior to the expiration of its full statutory term as presently shortened by any terminal disclaimer.

Check either box 1 or 2 below, if appropriate.

- 1. For submissions on behalf of an organization (e.g., corporation, partnership, university, government agency, etc.), the undersigned is empowered to act on behalf of the organization.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

- 2. The undersigned is an attorney or agent of record.



Signature
April 19, 2004

Date
Dov Rosenfeld

Typed or printed name
+1-510-547-3378

Telephone Number

- Terminal disclaimer fee under 37 CFR 1.20(d) included.

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

*Statement under 37 CFR 3.73(b) is required if terminal disclaimer is signed by the assignee (owner).
Form PTO/SB/98 may be used for making this certification. See MPEP § 324.

This collection of information is required by 37 CFR 1.321. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Office, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

04/27/2004 SSANDARA 00000017 500292 09608126
02 FC:1814 110.00 DA

Approved for use through 07/31/2006. OMB 0851-0031
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PTO/SB/26 (08-03)

**TERMINAL DISCLAIMER TO OBTAIN A DOUBLE PATENTING
REJECTION OVER A PRIOR PATENT**

Docket Number (Optional)
APPT-001-3

In re Application of: Dietz, et al.

Application No.: 09/608126

Filed: 30 Jun 2000

For: Re-using Information from Data Transactions for Maintaining Statistics in Network Monitoring

The owner*, Hi/fn, Inc. of 100 percent interest in the instant application hereby disclaims, except as provided below, the terminal part of the statutory term of any patent granted on the instant application, which would extend beyond the expiration date of the full statutory term defined in 35 U.S.C. 154 and 173, as presently shortened by any terminal disclaimer, of prior Patent No. 6,651,099. The owner hereby agrees that any patent so granted on the instant application shall be enforceable only for and during such period that it and the prior patent are commonly owned. This agreement runs with any patent granted on the instant application and is binding upon the grantee, its successors or assigns.

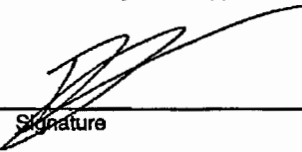
In making the above disclaimer, the owner does not disclaim the terminal part of any patent granted on the instant application that would extend to the expiration date of the full statutory term as defined in 35 U.S.C. 154 and 173 of the prior patent, as presently shortened by any terminal disclaimer, in the event that it later: expires for failure to pay a maintenance fee, is held unenforceable, is found invalid by a court of competent jurisdiction, is statutorily disclaimed in whole or terminally disclaimed under 37 CFR 1.321, has all claims canceled by a reexamination certificate, is reissued, or is in any manner terminated prior to the expiration of its full statutory term as presently shortened by any terminal disclaimer.

Check either box 1 or 2 below, if appropriate.

- 1. For submissions on behalf of an organization (e.g., corporation, partnership, university, government agency, etc.), the undersigned is empowered to act on behalf of the organization.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

- 2. The undersigned is an attorney or agent of record.



Signature
April 19, 2004

Date
Dov Rosenfeld

Typed or printed name
+1-510-547-3378

Telephone Number

- Terminal disclaimer fee under 37 CFR 1.20(d) included.

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

*Statement under 37 CFR 3.73(b) is required if terminal disclaimer is signed by the assignee (owner).
Form PTO/SB/98 may be used for making this certification. See MPEP § 324.

This collection of information is required by 37 CFR 1.321. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Office, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

Notice of Allowability	Application No.	Applicant(s)	
	09/608,126	DIETZ ET AL.	
	Examiner	Art Unit	
	Thong H Vu	2142	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address--

All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance (PTOL-85) or other appropriate communication will be mailed in due course. **THIS NOTICE OF ALLOWABILITY IS NOT A GRANT OF PATENT RIGHTS.** This application is subject to withdrawal from issue at the initiative of the Office or upon petition by the applicant. See 37 CFR 1.313 and MPEP 1308.

1. This communication is responsive to 4/19/04.
2. The allowed claim(s) is/are 1-21.
3. The drawings filed on 6/30/00 are accepted by the Examiner.
4. Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
 - a) All b) Some* c) None of the:
 1. Certified copies of the priority documents have been received.
 2. Certified copies of the priority documents have been received in Application No. _____.
 3. Copies of the certified copies of the priority documents have been received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

* Certified copies not received: _____.

Applicant has THREE MONTHS FROM THE "MAILING DATE" of this communication to file a reply complying with the requirements noted below. Failure to timely comply will result in ABANDONMENT of this application. **THIS THREE-MONTH PERIOD IS NOT EXTENDABLE.**

5. A SUBSTITUTE OATH OR DECLARATION must be submitted. Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL PATENT APPLICATION (PTO-152) which gives reason(s) why the oath or declaration is deficient.
 6. CORRECTED DRAWINGS (as "replacement sheets") must be submitted.
 - (a) including changes required by the Notice of Draftsperson's Patent Drawing Review (PTO-948) attached
 - 1) hereto or 2) to Paper No./Mail Date _____.
 - (b) including changes required by the attached Examiner's Amendment / Comment or in the Office action of Paper No./Mail Date _____.
- Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the drawings in the front (not the back) of each sheet. Replacement sheet(s) should be labeled as such in the header according to 37 CFR 1.121(d).
7. DEPOSIT OF and/or INFORMATION about the deposit of BIOLOGICAL MATERIAL must be submitted. Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Attachment(s)

- | | |
|---|--|
| 1. <input type="checkbox"/> Notice of References Cited (PTO-892) | 5. <input type="checkbox"/> Notice of Informal Patent Application (PTO-152) |
| 2. <input type="checkbox"/> Notice of Draftsperson's Patent Drawing Review (PTO-948) | 6. <input type="checkbox"/> Interview Summary (PTO-413),
Paper No./Mail Date _____. |
| 3. <input type="checkbox"/> Information Disclosure Statements (PTO-1449 or PTO/SB/08),
Paper No./Mail Date _____ | 7. <input type="checkbox"/> Examiner's Amendment/Comment |
| 4. <input type="checkbox"/> Examiner's Comment Regarding Requirement for Deposit
of Biological Material | 8. <input type="checkbox"/> Examiner's Statement of Reasons for Allowance |
| | 9. <input type="checkbox"/> Other _____. |





UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

NOTICE OF ALLOWANCE AND FEE(S) DUE

21921 7590 06/04/2004
DOV ROSENFELD
5507 COLLEGE AVE
SUITE 2
OAKLAND, CA 94618

EXAMINER

VU, THONG H

ART UNIT PAPER NUMBER

2142

DATE MAILED: 06/04/2004

Table with 5 columns: APPLICATION NO., FILING DATE, FIRST NAMED INVENTOR, ATTORNEY DOCKET NO., CONFIRMATION NO.
09/608,126 06/30/2000 Russell S. Dietz APPT-001-3 2145

TITLE OF INVENTION: RE-USING INFORMATION FROM DATA TRANSACTIONS FOR MAINTAINING STATISTICS IN NETWORK MONITORING

Table with 6 columns: APPLN. TYPE, SMALL ENTITY, ISSUE FEE, PUBLICATION FEE, TOTAL FEE(S) DUE, DATE DUE
nonprovisional NO \$1330 \$0 \$1330 09/07/2004

THE APPLICATION IDENTIFIED ABOVE HAS BEEN EXAMINED AND IS ALLOWED FOR ISSUANCE AS A PATENT. PROSECUTION ON THE MERITS IS CLOSED. THIS NOTICE OF ALLOWANCE IS NOT A GRANT OF PATENT RIGHTS. THIS APPLICATION IS SUBJECT TO WITHDRAWAL FROM ISSUE AT THE INITIATIVE OF THE OFFICE OR UPON PETITION BY THE APPLICANT. SEE 37 CFR 1.313 AND MPEP 1308.

THE ISSUE FEE AND PUBLICATION FEE (IF REQUIRED) MUST BE PAID WITHIN THREE MONTHS FROM THE MAILING DATE OF THIS NOTICE OR THIS APPLICATION SHALL BE REGARDED AS ABANDONED. THIS STATUTORY PERIOD CANNOT BE EXTENDED. SEE 35 U.S.C. 151. THE ISSUE FEE DUE INDICATED ABOVE REFLECTS A CREDIT FOR ANY PREVIOUSLY PAID ISSUE FEE APPLIED IN THIS APPLICATION. THE PTOL-85B (OR AN EQUIVALENT) MUST BE RETURNED WITHIN THIS PERIOD EVEN IF NO FEE IS DUE OR THE APPLICATION WILL BE REGARDED AS ABANDONED.

HOW TO REPLY TO THIS NOTICE:

I. Review the SMALL ENTITY status shown above.

If the SMALL ENTITY is shown as YES, verify your current SMALL ENTITY status:

A. If the status is the same, pay the TOTAL FEE(S) DUE shown above.

B. If the status is changed, pay the PUBLICATION FEE (if required) and twice the amount of the ISSUE FEE shown above and notify the United States Patent and Trademark Office of the change in status, or

If the SMALL ENTITY is shown as NO:

A. Pay TOTAL FEE(S) DUE shown above, or

B. If applicant claimed SMALL ENTITY status before, or is now claiming SMALL ENTITY status, check the box below and enclose the PUBLICATION FEE and 1/2 the ISSUE FEE shown above.

[] Applicant claims SMALL ENTITY status. See 37 CFR 1.27.

II. PART B - FEE(S) TRANSMITTAL should be completed and returned to the United States Patent and Trademark Office (USPTO) with your ISSUE FEE and PUBLICATION FEE (if required). Even if the fee(s) have already been paid, Part B - Fee(s) Transmittal should be completed and returned. If you are charging the fee(s) to your deposit account, section "4b" of Part B - Fee(s) Transmittal should be completed and an extra copy of the form should be submitted.

III. All communications regarding this application must give the application number. Please direct all communications prior to issuance to Mail Stop ISSUE FEE unless advised to the contrary.

IMPORTANT REMINDER: Utility patents issuing on applications filed on or after Dec. 12, 1980 may require payment of maintenance fees. It is patentee's responsibility to ensure timely payment of maintenance fees when due.

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: **Mail**

**Mail Stop ISSUE FEE
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450
or Fax (703) 746-4000**

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 4 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Legibly mark-up with any corrections or use Block 1)

21921 7590 06/04/2004

DOV ROSENFELD
5507 COLLEGE AVE
SUITE 2
OAKLAND, CA 94618

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission

I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO, on the date indicated below.

(Depositor's name)
(Signature)
(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/608,126	06/30/2000	Russell S. Dietz	APPT-001-3	2145

TITLE OF INVENTION: RE-USING INFORMATION FROM DATA TRANSACTIONS FOR MAINTAINING STATISTICS IN NETWORK MONITORING

APPLN. TYPE	SMALL ENTITY	ISSUE FEE	PUBLICATION FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	NO	\$1330	\$0	\$1330	09/07/2004

EXAMINER	ART UNIT	CLASS-SUBCLASS
VU, THONG H	2142	709-204000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).

- Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.
- "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. Use of a Customer Number is required.

2. For printing on the patent front page, list (1) the names of up to 3 registered patent attorneys or agents OR, alternatively, (2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.

1 _____
2 _____
3 _____

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. Inclusion of assignee data is only appropriate when an assignment has been previously submitted to the USPTO or is being submitted under separate cover. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE (B) RESIDENCE: (CITY and STATE OR COUNTRY)

Please check the appropriate assignee category or categories (will not be printed on the patent); individual corporation or other private group entity government

4a. The following fee(s) are enclosed:

- Issue Fee
- Publication Fee
- Advance Order - # of Copies _____

4b. Payment of Fee(s):

- A check in the amount of the fee(s) is enclosed.
- Payment by credit card. Form PTO-2038 is attached.
- The Director is hereby authorized by charge the required fee(s), or credit any overpayment, to Deposit Account Number _____ (enclose an extra copy of this form).

Director for Patents is requested to apply the Issue Fee and Publication Fee (if any) or to re-apply any previously paid issue fee to the application identified above.

(Authorized Signature)	(Date)
------------------------	--------

NOTE: The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office.

This collection of information is required by 37 CFR 1.311. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, Alexandria, Virginia 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, Alexandria, Virginia 22313-1450.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

TRANSMIT THIS FORM WITH FEE(S)



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

Table with columns: APPLICATION NO., FILING DATE, FIRST NAMED INVENTOR, ATTORNEY DOCKET NO., CONFIRMATION NO.
09/608,126 06/30/2000 Russell S. Dietz APPT-001-3 2145
21921 7590 06/04/2004
DOV ROSENFELD
5507 COLLEGE AVE
SUITE 2
OAKLAND, CA 94618
EXAMINER
VU, THONG H
ART UNIT PAPER NUMBER
2142
DATE MAILED: 06/04/2004 14

Determination of Patent Term Adjustment under 35 U.S.C. 154 (b)
(application filed on or after May 29, 2000)

The Patent Term Adjustment to date is 655 day(s). If the issue fee is paid on the date that is three months after the mailing date of this notice and the patent issues on the Tuesday before the date that is 28 weeks (six and a half months) after the mailing date of this notice, the Patent Term Adjustment will be 655 day(s).

If a Continued Prosecution Application (CPA) was filed in the above-identified application, the filing date that determines Patent Term Adjustment is the filing date of the most recent CPA.

Applicant will be able to obtain more detailed information by accessing the Patent Application Information Retrieval (PAIR) system (http://pair.uspto.gov).

Any questions regarding the Patent Term Extension or Adjustment determination should be directed to the Office of Patent Legal Administration at (703) 305-1383. Questions relating to issue and publication fee payments should be directed to the Customer Service Center of the Office of Patent Publication at (703) 305-8283.



004 12:51 FAX 15102912985

INVENTEK

004

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: Mail Stop ISSUE FEE Commissioner for Patents P.O. Box 1450 Alexandria, Virginia 22313-1450 or Fax (703) 746-4000

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 4 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Legibly mark-up with any corrections or use Block 1)

21921 7590 06/04/2004 DOV ROSENFELD 5507 COLLEGE AVE SUITE 2 OAKLAND, CA 94618

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO, on the date indicated below.

Amy Drury (Depositor's name) Amy Drury (Signature) June 23, 2004 (Date)

Table with 5 columns: APPLICATION NO., FILING DATE, FIRST NAMED INVENTOR, ATTORNEY DOCKET NO., CONFIRMATION NO. Values: 09/608,126, 06/30/2000, Russell S. Dietz, APPT-001-3, 2145

TITLE OF INVENTION: RE-USING INFORMATION FROM DATA TRANSACTIONS FOR MAINTAINING STATISTICS IN NETWORK MONITORING

Table with 6 columns: APPLN. TYPE, SMALL ENTITY, ISSUE FEE, PUBLICATION FEE, TOTAL FEE(S) DUE, DATE DUE. Values: nonprovisional, NO, \$1330, \$0, \$1330, 09/07/2004

Table with 3 columns: EXAMINER, ART UNIT, CLASS-SUBCLASS. Values: VU, THONG H, 2142, 709-204000

- 1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363). 2. For printing on the patent front page, list (1) the names of up to 3 registered patent attorneys or agents OR, alternatively, (2) the name of a single firm... 1. Dov Rosenfeld 2. Inventek

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type) PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. Inclusion of assignee data is only appropriate when an assignment has been previously submitted to the USPTO or is being submitted under separate cover. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE: Hi/fn, Inc. (B) RESIDENCE: (CITY AND STATE OR COUNTRY): Los Gatos, CA

Please check the appropriate assignee category or categories (will not be printed on the patent): individual corporation or other private group entity government

- 4a. The following fee(s) are enclosed: Issue Fee Publication Fee Advance Order - # of Copies 10 4b. Payment of Fee(s): A check in the amount of the fee(s) is enclosed. Payment by credit card. Form PTO-2038 is attached. The Director is hereby authorized by charge the required fee(s), or credit any overpayment, to Deposit Account Number 50-0292 (enclose an extra copy of this form).

Director for Patents is requested to apply the Issue Fee and Publication Fee (if any) or to re-apply any previously paid issue fee to the application identified above.

(Authorized Signatory) [Signature] (Date) June 23, 2004

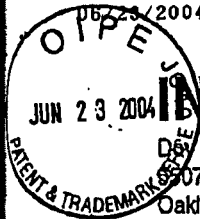
NOTE: The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant, a registered attorney or agent, or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office.

06/24/2004 AWONDAF2 00000060 09608126 01 FC:1501 1330.00 DP 02 FC:8001 30.00 DP

This collection of information is required by 37 CFR 1.311. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, Alexandria, Virginia 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, Alexandria, Virginia 22313-1450.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

TRANSMIT THIS FORM WITH FEE(S)



INVENTEK

Fax

Dov Rosenfeld
107 College Avenue, Suite 2
Oakland, CA 94618, USA
Phone: (510) 547-3378; Fax: (510) 291-2985
dov@inventek.com

OUR REF: APPT-001-3

TO: Mail Stop Issue Fee
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

FAX No.: (703) 746-4000

DATE: June 23, 2004

FROM: Dov Rosenfeld, Reg. No., 38,687

RE: Issue Fee for Application No.: 09/608,126

Number of pages including cover: 5

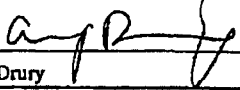
OFFICIAL COMMUNICATION
ISSUE FEE PAYMENT

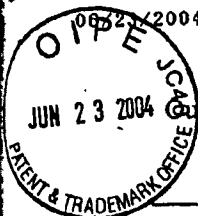
Included herewith are:

- A transmittal letter and copy
- Fee(s) Transmittal (form PTOL-85)
- Credit Card charge form for issue fee

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number (703) 746-4000 addressed to Mail Stop Issue Fee, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: June 23, 2004 Signed: 
Name: Amy Drury



Our Ref./Docket No: APPT-001-3

Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz, <i>et al.</i> Application No.: 09/608,126 Filed: June 30, 2000 Title: RE-USING INFORMATION FROM DATA TRANSACTIONS FOR MAINTAINING STATISTICS IN NETWORK MONITORING	Group Art Unit: 2142 Examiner: Vu, Thong H. Notice of Allowance Mailed: June, 4, 2004 Confirmation No: 2145
---	---

SUBMISSION OF ISSUE FEE

Mail Stop ISSUE FEE
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

Transmitted herewith is a completed "Issue Fee Transmittal" Form. Included with the form are:

- A credit card payment form for the issue fee and any advance order of copies;
- drawing corrections (with separate letter);
- formal drawings (with separate letter);

The Commissioner is hereby authorized to charge payment of the any missing fee or credit any overpayment to Deposit Account No. 50-0292
(A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED):

Respectfully Submitted,

June 23, 2004
Date

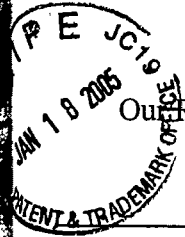
Dov Rosenfeld, Reg. No. 38687

Address for correspondence:
Dov Rosenfeld
5507 College Avenue, Suite 2,
Oakland, CA 94618
Tel. +1-510-547-3378; Fax: +1-510-291-2985

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number (703) 746-4000 addressed to Mail Stop Issue Fee, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: June 23, 2004 Signed:
Name: Amy Drury



Out. Ref./Docket No: Am 001-3

Patent

C9C

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s): Dietz, *et al.*
 Assignee: Hi/fn, Inc.
 Patent No: 6,839,751
 Issue Date: January, 4, 2005
 Application No.: 09/608,126
 Filed: June 30, 2000
 Title: RE-USING INFORMATION FROM DATA
 TRANSACTIONS FOR MAINTAINING
 STATISTICS IN NETWORK
 MONITORING

+15
 (C)
 2/16/05

Certificate
JAN 25 2005
of Correction

REQUEST FOR CERTIFICATE OF CORRECTIONS

Commissioner for Patents
 P.O. Box 1450
 Alexandria, VA 22313-1450

Dear Commissioner:

The above patent contains significant error(s) as indicated on the attached Certificate of Correction form (submitted in duplicate).

X Such error(s) arose through the fault of the Patent and Trademark Office. It is requested that the certificate be issued at no cost to the applicant.

However, if it is determined that the error(s) arose through the fault of applicant(s), please note that such error is of clerical error or minor nature and occurred in good faith and therefore issuance of the certificate of Correction is respectfully requested. The Commissioner is authorized to charge Deposit Account No. 50-0292 any required fee. A duplicate of this request is attached.

____ Such error(s) arose through the fault of applicant(s). A credit card charge form for the fee is enclosed. Each such error is of clerical error or minor nature and occurred in good faith and therefore issuance of the certificate of Correction is respectfully requested.

Certificate of Mailing under 37 CFR 1.8

I hereby certify that this response is being deposited with the United States Postal Service as first class mail in an envelope addressed to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: Jan. 12, 2005 Signed: [Signature]
 Name: Amy Drury

JAN 26 2005

Such error(s) specifically:

In column 6, line 65, kindly change "In so" to --In some--.

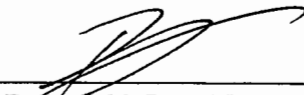
In column 13, line 23, kindly change "pattern—a signature—an be" to -- pattern—a signature—can be --.

In column 51, line 13 (the 1st line of claim 6), kindly change "claim 1" to --claim 7--.

The undersigned requests being contacted at (510) 547-3378 if there are any questions or clarifications, or if there are any problems with issuance of the Certificate of Correction.

Respectfully Submitted,

Jan. 12, 2005
Date

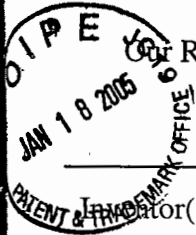


Dov Rosenfeld, Reg. No. 38687
Agent of Record.

Address for correspondence:

Dov Rosenfeld
5507 College Avenue, Suite 2,
Oakland, CA 94618
Tel. (510) 547-3378; Fax: (510) 291-2985

JAN 26 2005



Pat Ref./Docket No: At PT-001-3

Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s): Dietz, *et al.*

Assignee: Hi/fn, Inc.

Patent No: 6,839,751

Issue Date: January, 4, 2005

Application No.: 09/608,126

Filed: June 30, 2000

Title: RE-USING INFORMATION FROM DATA
TRANSACTIONS FOR MAINTAINING
STATISTICS IN NETWORK
MONITORING

REQUEST FOR CERTIFICATE OF CORRECTIONS

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

The above patent contains significant error(s) as indicated on the attached Certificate of Correction form (submitted in duplicate).

X Such error(s) arose through the fault of the Patent and Trademark Office. It is requested that the certificate be issued at no cost to the applicant.

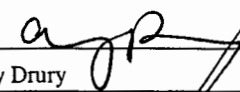
However, if it is determined that the error(s) arose through the fault of applicant(s), please note that such error is of clerical error or minor nature and occurred in good faith and therefore issuance of the certificate of Correction is respectfully requested. The Commissioner is authorized to charge Deposit Account No. 50-0292 any required fee. A duplicate of this request is attached.

 Such error(s) arose through the fault of applicant(s). A credit card charge form for the fee is enclosed. Each such error is of clerical error or minor nature and occurred in good faith and therefore issuance of the certificate of Correction is respectfully requested.

Certificate of Mailing under 37 CFR 1.8

I hereby certify that this response is being deposited with the United States Postal Service as first class mail in an envelope addressed to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: Jan. 12, 2005

Signed: 

Name: Amy Drury

Such error(s) specifically:

In column 6, line 65, kindly change "In so" to --In some--.

In column 13, line 23, kindly change "pattern—a signature—an be" to -- pattern—a signature—can be --.

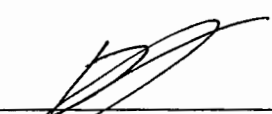
In column 51, line 13 (the 1st line of claim 6), kindly change "claim 1" to --claim 7--.

The undersigned requests being contacted at (510) 547-3378 if there are any questions or clarifications, or if there are any problems with issuance of the Certificate of Correction.

Respectfully Submitted,

Jan. 12, 2005

Date



Dov Rosenfeld, Reg. No. 38687
Agent of Record.

Address for correspondence:

Dov Rosenfeld
5507 College Avenue, Suite 2,
Oakland, CA 94618
Tel. (510) 547-3378; Fax: (510) 291-2985

JAN 26 2005

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE CERTIFICATE OF CORRECTION

PATENT NO : 6,839,751

DATED : January 4, 2005

INVENTOR(S) : Dietz, et al.

It is certified that an error appears in the above-identified patent and that said Letters Patent are hereby corrected as shown below:

In column 6, line 65, kindly change "In so" to --In some--.

In column 13, line 23, kindly change "pattern—a signature—an be" to -- pattern—a signature—
can be --.

In column 51, line 13 (the 1st line of claim 6), kindly change "claim 1" to --claim 7--.

MAILING ADDRESS OF SENDER (Atty/Agent of Record):
Dov Rosenfeld, Reg. No. 38687
5507 College Avenue, Suite 2
Oakland, CA 94618

PATENT NO: 6,839,751
No. of additional copies

JAN 26 2005

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO : 6,839,751 *bl*

DATED : January 4, 2005

INVENTOR(S) : Dietz, et al.

It is certified that an error appears in the above-identified patent and that said Letters Patent are hereby corrected as shown below:

In column 6, line 65, kindly change "In so" to --In some--.

In column 13, line 23, kindly change "pattern—a signature—an be" to -- pattern—a signature—
can be --.

In column 51, line 13 (the 1st line of claim 6), kindly change "claim 1" to --claim 7--.

cont

MAILING ADDRESS OF SENDER (Atty/Agent of Record):
Dov Rosenfeld, Reg. No. 38687
5507 College Avenue, Suite 2
Oakland, CA 94618

PATENT NO: 6,839,751
No. of additional copies

JAN 26 2005

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 6,839,751 B1
DATED : January 4, 2005
INVENTOR(S) : Dietz et al.

Page 1 of 1

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 6,

Line 65, please change "In so" to -- In some --.

Column 13,


Line 23, please change, "~~pattern—a signature—an be~~" to -- pattern—a signature—can be --.

Column 51,

Line 13, please change "claim 1" to -- claim 7 --.

Signed and Sealed this

Eighth Day of March, 2005



JON W. DUDAS

Director of the United States Patent and Trademark Office

	L #	Hits	Search Text	DBs	Time Stamp
1	L1	829	(lookup look adj up map\$4 match\$3 compar\$4) with (entry entries input attribute parameter address data signal packet cell frame segment datagram) with database with (present\$3 current\$2 exist\$3)	USPAT	2003/06/20 07:44
2	L3	1161	(new\$2 add\$7) with (entry entries input attribute parameter address data signal packet cell frame segment datagram) with database with (present\$3 current\$2 exist\$3)	USPAT	2003/06/20 07:38
3	L2	7	updat\$3 with statistic\$2 with (entry entries input attribute parameter address data signal packet cell frame segment datagram) with database with (present\$3 current\$2 exist\$3)	USPAT	2003/06/20 07:42
4	L4	190	1 and 3	USPAT	2003/06/20 07:42
5	L6	0	4 and 5	USPAT	2003/06/20 07:52
6	L5	27	parser and (monitor\$3 filter\$3 analyzer) and database and statistic\$2 and new\$2 adj (flow stream packet frame cell protocol session link connection) and (exist\$3 current\$2 present) and (lookup look adj up map\$4 match\$3 compar\$4) and parameter and (qos quality adj service)	USPAT	2003/06/20 09:07
7	L7	18	parser and (monitor\$3 filter\$3 analyzer) and database same statistic\$2 and new\$2 adj (flow stream packet frame cell protocol session link connection) and (exist\$3 current\$2 present) and (lookup look adj up map\$4 match\$3 compar\$4) and parameter and (qos quality adj service)	USPAT	2003/06/20 07:56

	L #	Hits	Search Text	DBs	Time Stamp
8	L8	616	(monitor\$3 filter\$3 analyzer) same database same statistic\$2	USPAT	2003/06/20 07:57
9	L10	0	9 and new\$2 adj3 (flow stream packet frame cell protocol session link connection) and (exist\$3 current\$2 present) and (lookup look adj up map\$4 match\$3 compar\$4) and parameter and (qos quality adj service)	USPAT	2003/06/20 08:00
10	L12	1	9 and new\$2 and (exist\$3 current\$2 present) and (lookup look adj up map\$4 match\$3 compar\$4) and parameter and (qos quality adj service)	USPAT	2003/06/20 08:00
11	L11	1	9 and new\$2 and (exist\$3 current\$2 present) and (lookup look adj up map\$4 match\$3 compar\$4) and parameter and (qos quality adj service)	USPAT	2003/06/20 08:01
12	L13	5	"09/093,955" or "6249675"	USPAT	2003/06/20 08:05
13	L14	5	13 and (statistic\$2 quality monitor\$3 analyz\$3 new\$2)	USPAT	2003/06/20 08:06
14	L9	33	((monitor\$3 filter\$3 analyzer) same database same statistic\$2).ab.	USPAT	2003/06/20 09:00
15	L15	203035	(monitor\$3 trac\$4 filter\$3 analyzer).ab.	USPAT	2003/06/20 09:00
16	L16	13843	(packet message datagram cell frame) with (extract\$3 filter\$3) with (identif\$4 tyep class\$9 address time protocol ID)	USPAT	2003/06/20 09:48
17	L18	48233	(different adj time (last previous) adj time (current\$2 exist\$3 present) adj time)	USPAT	2003/06/20 09:07
18	L19	81	database and statistic\$2 and new\$2 adj (flow stream packet frame cell protocol session link connection) and pattern and (qos quality adj service)	USPAT	2003/06/20 09:26
19	L20	4	15 and 16 and 18 and 19	USPAT	2003/06/20 09:08

	L #	Hits	Search Text	DBs	Time Stamp
20	L21	9188	(each every) adj (packet message datagram cell frame)	USPAT	2003/06/20 09:26
21	L22	22	database same statistic\$2 and 15 and 21	USPAT	2003/06/20 09:33
22	L23	13	22 and pattern	USPAT	2003/06/20 09:49
23	L24	260	database same statistic\$2 and pattern and 709/2\$\$	USPAT	2003/06/20 09:47
24	L25	62	18 and 24	USPAT	2003/06/20 09:34
25	L26	10	15 and 25	USPAT	2003/06/20 09:34
26	L27	41	database same statistic\$2 and pattern and 15 and (reuse\$1 re adj use\$1)	USPAT	2003/06/20 09:48
27	L28	18	(packet message datagram cell frame) with (extract\$3 filter\$3) and 27	USPAT	2003/06/20 09:49
28	L29	18	28 and pattern	USPAT	2003/06/20 09:49
29	L30	1	("5850388").PN.	USPAT	2003/06/20 10:18
30	L31	1	30 and port	USPAT	2003/06/20 10:18

	L #	Hits	Search Text	DBs	Time Stamp
1	L9	7555	(analy\$4 identif\$4 monitor\$3 detect\$3) adj10 (packet stream thread\$3 session link connect\$3 channel) with (new add\$3 addion\$2)	USPAT	2003/06/19 11:11
2	L10	830	(analy\$4 identif\$4 monitor\$3 detect\$3) adj10 (packet near5 (traffic flow) stream) with (new add\$3 addion\$2)	USPAT	2003/06/19 10:12
3	L11	25	10 and qos	USPAT	2003/06/19 09:44
4	L12	808	updat\$3 adj10 (packet near5 (traffic flow) stream)	USPAT	2003/06/19 09:45
5	L13	5	11 and 12	USPAT	2003/06/19 09:45
6	L14	5362	(analy\$4 filter\$3 monitor\$3) adj10 (packet stream thread\$3 session link connect\$3 channel) with (new add\$3 addion\$2)	USPAT	2003/06/19 09:57
7	L15	5375	monitor\$3 same statistic\$2	USPAT	2003/06/19 09:52
8	L17	10	14 and 15 and qos and updat\$3	USPAT	2003/06/19 11:12
9	L18	1685	(analy\$4 filter\$3 monitor\$3) with stream with (new add\$3 updat\$3 addion\$2)	USPAT	2003/06/19 09:59
10	L20	8	18 and qos and updat\$3	USPAT	2003/06/19 10:11
11	L21	160425	(current\$2 exist\$3) near3 (flow traffic)	USPAT	2003/06/19 10:12
12	L22	163636	(analy\$4 filter\$3 monitor\$3).ab.	USPAT	2003/06/19 10:12
13	L23	12584	((analy\$4 filter\$3 monitor\$3) same (stream isochronous\$3 multimedia media video near audio (packet cell) near3 flow)).ab.	USPAT	2003/06/19 10:15
14	L24	12	21 and 22 and 23 and qos and updat\$3	USPAT	2003/06/19 10:55
15	L25	7	24 and statistic\$2	USPAT	2003/06/19 10:20
16	L27	1	21 and 22 and 23 and database and statistic	USPAT	2003/06/19 10:56
17	L28	14	21 and 22 and database and statistic	USPAT	2003/06/19 10:57

	L #	Hits	Search Text	DBs	Time Stamp
18	L29	0	21 and 22 and database same statistic	USPAT	2003/06/19 11:12
19	L30	1	23 and database same statistic	USPAT	2003/06/19 10:58
20	L31	1	30 and (stream cells packets flow)	USPAT	2003/06/19 11:13
21	L32	0	30 and (realtime real adj time)	USPAT	2003/06/19 11:17
22	L33	281101	(analy\$4 identif\$4 monitor\$3 detect\$3).ab.	USPAT	2003/06/19 11:12
23	L34	443	((analy\$4 identif\$4 monitor\$3 detect\$3) same statistic).ab.	USPAT	2003/06/19 11:12
24	L36	0	35 and qos and updat\$3	USPAT	2003/06/19 11:12
25	L37	0	35 and qos	USPAT	2003/06/19 11:14
26	L35	18	34 and database same statistic	USPAT	2003/06/19 11:30
27	L39	1	35 and 23	USPAT	2003/06/19 11:14
28	L41	2	35 and (realtime real adj time) and (qos quality adj service)	USPAT	2003/06/19 11:21
29	L42	3543	(compar\$6 match\$3 map\$4) near3 pattern same (extract\$3 filter\$3 retriev\$3)	USPAT	2003/06/19 11:23
30	L43	8	34 and 42	USPAT	2003/06/19 11:23
31	L44	0	43 and database and statistic	USPAT	2003/06/19 11:25
32	L45	2	43 and database	USPAT	2003/06/19 11:26
33	L46	41	34 and database and pattern	USPAT	2003/06/19 11:27
34	L47	10	34 and database and (compar\$6 match\$3 map\$4) near3 pattern	USPAT	2003/06/19 11:33
35	L48	2	47 and database same statistic	USPAT	2003/06/19 11:30
36	L49	5	47 and (realtime real adj time) and (analy\$4 filter\$3)	USPAT	2003/06/19 11:34

	L #	Hits	Search Text	DBs	Time Stamp
1	L2	1	(traffic) with (monitor\$3 track\$3 analy\$4 classif\$4) same (max\$6 same min\$6) same (adjust\$4)	USPAT	2003/06/19 07:39
2	L3	2	(traffic) with (monitor\$3 track\$3 analy\$4 classif\$4) same balanc\$3 same (adjust\$4)	USPAT	2003/06/19 07:40
3	L4	14	(traffic) with (monitor\$3 track\$3 analy\$4 classif\$4) same percent\$3 same (adjust\$4)	USPAT	2003/06/19 08:04
4	L5	84	(traffic) with (monitor\$3 track\$3 analy\$4 classif\$4) same (up down max\$6 min\$6 percent\$3) same (adjust\$4 consolidat\$3)	USPAT	2003/06/19 08:14
5	L6	9	(traffic) with (monitor\$3 track\$3 analy\$4 classif\$4) same (up with down max\$6 with min\$6) same (adjust\$4 consolidat\$3)	USPAT	2003/06/19 08:26
6	L7	2	(("6208863") or ("6141686")).PN.	USPAT	2003/06/19 08:26
7	L8	1	7 and (operator administrator maunal static)	USPAT	2003/06/19 08:27

	L #	Hits	Search Text
1	L1	154	(filter\$3 analy\$5 monitor\$3) with packet with (low entry id identifier code value number indicator versionID) with (current\$2 exist\$3)
2	L2	2	1 and database and (satic\$2 histor\$4) and (state codition status)
3	L5	0	(filter\$3 analy\$5 monitor\$3) with packet and database near5 (satic\$2 histor\$4) same (state condition status) with (flow traffic stream)
4	L9	7	(filter\$3 analy\$5 monitor\$3) with (cell frame datagram segment packet) same database and (satic\$2 histor\$4) same (state condition status) with (flow traffic stream)
5	L10	165	(filter\$3 analy\$5 monitor\$3).ab. and database with (satic\$2 histor\$4) and (state condition status) with (flow traffic stream)
6	L11	36	(filter\$3 analy\$5 monitor\$3).ab. and database with (satic\$2 histor\$4) and (state condition status) with (flow traffic stream) and (flow traffic stream) with updat\$3
7	L12	12	11 and (flow traffic stream) near3 new
8	L13	5	(filter\$3 analy\$5 monitor\$3).ab. and database with (satic\$2 histor\$4) and (state condition status) with (flow traffic stream) and new adj entry and updat\$3
9	L14	5	(filter\$3 analy\$5 monitor\$3).ab. and database with (satic\$2 histor\$4) and (state condition status) with (flow traffic stream) and new adj entry and updat\$3 and (stream flow traffic packet cell frame)
10	L15	50	((filter\$3 analy\$5 monitor\$3) same packet\$3).ab. and (database satic\$2 histor\$4) and (state condition status version\$2) with (flow traffic stream) and (new adj entry updat\$3) with (stream flow traffic packet cell frame)
11	L16	0	15 and if with (extist\$3 current\$2)
12	L17	0	15 and if with new
13	L18	48	15 and (extist\$3 current\$2)
14	L19	7	15 and (extist\$3 current\$2) with new and (reuse "re-use" re adj used re adj using)

	L #	Hits	Search Text
15	L20	11	((filter\$3 analy\$5 monitor\$3) same packet\$3).ab. and (database satistic\$2 histor\$4) and (state condition status version\$2) with (flow traffic stream) and (new adj entry updat\$3) with (stream flow traffic packet cell frame) and (reuse "re-use" re adj used re adj using)
16	L21	8	((filter\$3 analy\$5 monitor\$3) same packet\$3).ab. and (new adj entry) and (reuse "re-use" re adj used re adj using)
17	L22	1	((filter\$3 analy\$5 monitor\$3) same packet\$3).ab. and (new adj entry) with (database histor\$4 satic\$4)
18	L23	3	((filter\$3 analy\$5 monitor\$3) same packet\$3).ab. and (new adj entry) same (database histor\$4 satic\$4)
19	L24	4	((filter\$3 analy\$5 monitor\$3) same packet\$3).ab. and (new adj3 entry) same (database histor\$4 satic\$4)

	L #	Hits	Search Text
1	L3	3	shared same transparent\$2 same simulat\$3
2	L4	16	shared same transparent\$2 same (plurality multiple different two more list table book database) near3 (address pointer indicator fields)
3	L5	17338	(messag\$3 email mail network shared) adj3 (queu\$3 bus buffer)
4	L6	3	4 and 5
5	L7	972	(messag\$3 email mail network shared) adj3 (queu\$3 bus buffer) same (plurality multiple different two more list table book database) near3 (address pointer indicator fields)
6	L8	200	(messag\$3 email mail network shared) adj3 (queu\$3 bus buffer) same Pars\$3
7	L9	2	(messag\$3 email mail network shared) adj3 (queu\$3 bus buffer) with Pars\$3 with (pointer address url ip)
8	L10	61	7 and 8
9	L11	30	7 and 8 and creat\$3 with (queu\$3 buffer bus shared adj memory)
10	L12	6	7 and 8 and creat\$3 with (shared)
11	L13	11	7 and 8 and network with (shared)
12	L14	31	7 and 8 and network adj (interface link connect\$3 adapter card port "i/o" input output)
13	L15	86	(nic network adj2 (interface link connect\$3 adapter card port "i/o" input output)) and multicast and management with translation
14	L16	79	15 and (bus queu\$3 buffer list table database index\$3 plurality multiple) near3 (pointer field address identifier ID)
15	L17	15	15 and (bus queu\$3 buffer) with (list table database index\$3 plurality multiple) near3 (pointer field address identifier ID)
16	L19	0	17 and hash\$3 and transparent\$2
17	L20	3	(emulat\$4 simulat\$3) same transparent\$2 and hash\$3 and multicast\$3
18	L21	1175	(processors cpus) and (local private client user) adj3 (memory cache queu\$3 busd buffer) with shared
19	L24	2	(processors cpus) and (local private client user) adj3 (memory cache queu\$3 busd buffer) with shared same first same second adj3 set
20	L25	20	shared same transparent\$2 same emulat\$3

	L #	Hits	Search Text
21	L26	12	shared same transparent\$2 with emulat\$3
22	L27	6	shared same transparent\$2 with emulat\$3 with (protocol lan atm ethernet token\$4 ring network)
23	L28	6	shared same transparent\$2 with emulat\$3 with (protocol lan atm ethernet token\$4 ring network) and (os operat\$3 adj system)
24	L29	163	(nic network adj2 (interface link connect\$3 adapter card port "i/o" input output)) and multicast and management with (translat\$3 conver\$4 transcod\$3 simulat\$3 emualt\$3 proxy redirect\$3 reformat\$4) and shared
25	L30	70	(nic network adj2 (interface link connect\$3 adapter card port "i/o" input output)) and multicast and management with (translat\$3 conver\$4 transcod\$3 simulat\$3 emualt\$3 proxy redirect\$3 reformat\$4) and shared adj memory
26	L31	44	(nic network adj2 (interface link connect\$3 adapter card port "i/o" input output)) and multicast and management with (translat\$3 conver\$4 transcod\$3 simulat\$3 emualt\$3 proxy redirect\$3 reformat\$4) and shared adj memory and extensions
27	L32	44	(nic network adj2 (interface link connect\$3 adapter card port "i/o" input output)) and multicast and management with (translat\$3 conver\$4 transcod\$3 simulat\$3 emualt\$3 proxy redirect\$3 reformat\$4) and shared adj memory and extensions and (pointer address indicator)
28	L33	42	(nic network adj2 (interface link connect\$3 adapter card port "i/o" input output)) and multicast and management with (translat\$3 conver\$4 transcod\$3 simulat\$3 emualt\$3 proxy redirect\$3 reformat\$4) and shared adj memory and extensions and (pointer address indicator) near3 (list table index\$3 plurality multiple database)
29	L34	40	(nic network adj2 (adapter card port "i/o" input output)) and multicast and management with (translat\$3 conver\$4 transcod\$3 simulat\$3 emualt\$3 proxy redirect\$3 reformat\$4) and shared adj memory and extensions and (pointer address indicator) near3 (list table index\$3 plurality multiple database)
30	L35	39	34 not ("2000" "2001" "2002" "2003").ay.

	L #	Hits	Search Text
31	L39	136	(nic network adj2 (adapter card port "i/o" input output) lan vlan ethernet ring token) with (translat\$3 conver\$4 transcod\$3 simulat\$3 emulat\$3 proxy redirect\$3 reformat\$4) with function with (instruciones program code driver command service)
32	L40	6	(nic network adj2 (adapter card port "i/o" input output) lan vlan ethernet ring token) with (translat\$3 conver\$4 transcod\$3 simulat\$3 emulat\$3 proxy redirect\$3 reformat\$4) with function with (instructions)
33	L41	80	mbuf
34	L42	42	mbuf and shared and pointer
35	L43	6	mbuf and (network lan vlan) with shared and pointer
36	L44	205	message adj (bufer queue bus) and (network lan vlan) with shared and pointer
37	L46	2	message adj (bufer queue bus) and transparent\$2 with (simulat\$3 emulat\$3) and shared adj memory and pointer
38	L47	11	message adj (bufer queue bus) and transparent\$2 with (simulat\$3 emulat\$3 proxy\$3 reformat\$4 conver\$\$) with protocol
39	L48	20	message adj (buffer queue bus) and transparent\$2 with (simulat\$3 emulat\$3 proxy\$3 reformat\$4 conver\$\$) with protocol
40	L49	15	message adj (buffer queue bus) and shared and transparent\$2 with (simulat\$3 emulat\$3 proxy\$3 reformat\$4 conver\$\$) with protocol
41	L50	202	(processors cpus) and shared same first adj3 set same second adj3 set
42	L51	158	(processors cpus) and shared same first adj3 (addresses pointers) same second adj3 (addresses pointers)
43	L52	27	(processors cpus) and shared adj memory same first adj3 (addresses pointers) same second adj3 (addresses pointers)
44	L53	3	(processors cpus) and private adj memory same first adj3 (addresses pointers) same second adj3 (addresses pointers)
45	L54	4	creat\$3 with (buffer queu\$3 bus) with (shared adj node server) same (pars\$3 multicast\$3 broadcast\$3)
46	L55	35	creat\$3 near5 (buffer queu\$3 bus) with (shared adj node server) and (pars\$3 multicast\$3 broadcast\$3) with (pointer address)

	L #	Hits	Search Text
47	L56	1	creat\$3 near5 (buffer queu\$3 bus) with (shared adj node server) and (pars\$3 multicast\$3 broadcast\$3 idstribut\$3) adj3 (buffer queu\$3 bus) with (pointer address)
48	L58	3	creat\$3 adj10 (buffer queu\$3 bus port shared) with (shared adj node server) and (pars\$3 multicast\$3 broadcast\$3 distribut\$3) adj3 (buffer queu\$3 bus port shared) with (pointer address)
49	L59	15	(establish\$3 initia\$4 creat\$3) adj10 (buffer queu\$3 bus port shared) with (shared adj node server) and (pars\$3 multicast\$3 broadcast\$3 distribut\$3) adj3 (buffer queu\$3 bus port shared) with (pointer address indicator ip url)
50	L60	997	(assign\$4 identif\$4 select\$3) adj10 (buffer bus queu\$3 space shared adj (memory port "i/o" channel)) same (broadcast\$3 multicast\$3)
51	L61	52	(assign\$4 identif\$4 select\$3) adj10 messag\$3 adj3 (buffer bus queu\$3 space shared adj (memory port "i/o" channel)) same (broadcast\$3 multicast\$3)
52	L62	10	61 and network with shared
53	L63	0	(establish\$3 initia\$4 creat\$3) adj10 (buffer queu\$3 bus port shared) with (shared adj node server host) same (pars\$3 multicast\$3 broadcast\$3 distribut\$3) with (shared adj (buffer queu\$3 bus port memory)) with (pointer address indicator ip url)
54	L64	0	(establish\$3 initia\$4 creat\$3) adj10 (buffer queu\$3 bus port shared) with (shared adj node server host) same (pars\$3 multicast\$3 broadcast\$3 distribut\$3) with (shared adj (buffer queu\$3 bus port memory))
55	L65	54	(establish\$3 initia\$4 creat\$3) adj10 (buffer queu\$3 bus port shared) with (shared adj node server host) same (shared adj (buffer queu\$3 bus port memory))
56	L66	28	(nic network adj2 (interface link connect\$3 adapter card port "i/o" input output)) and 65
57	L67	25	66 not ("2000" "2001" "2002" "2003").ay.
58	L69	10	66 and (broadcast\$3 multicast\$3 pars\$3) not ("2000" "2001" "2002" "2003").ay.
59	L71	0	(establish\$3 initia\$4 creat\$3) adj10 (buffer queu\$3 bus port shared) with (shared adj node server host) same (shared adj (buffer queu\$3 bus port memory) same (broadcast\$3 multicast\$3 pars\$3) not ("2000" "2001" "2002" "2003").ay.)

	L #	Hits	Search Text
60	L70	22	(establish\$3 initia\$4 creat\$3) adj10 (buffer queu\$3 bus port shared) with (shared adj node server host) same (shared adj (buffer queu\$3 bus port memory) same broadcast\$3 multicast\$3 pars\$3) not ("2000" "2001" "2002" "2003").ay.
61	L72	2	(generat\$4 creat\$3) adj10 (buffer queu\$3 bus port shared) with (shared adj node server host) same shared adj3 (buffer queu\$3 bus port memory) same (broadcast\$3 multicast\$3 pars\$3)
62	L73	19	(generat\$4 creat\$3) adj10 (buffer queu\$3 bus port shared) with (shared adj node server host) same shared adj3 (buffer queu\$3 bus port memory cache) and (broadcast\$3 multicast\$3 pars\$3)

	<u>L #</u>	Hits	Search Text	DBs	Time Stamp
1	L1	19209	709/2\$\$	USPAT	2003/05/08 12:47
2	L10	753	(traffic flow) adj3 (database data adj base)	USPAT	2003/05/08 13:47
3	L11	3	(traffic flow) adj3 (database data adj base) same (last previous\$2) with (compar\$6 match\$3 map\$4)	USPAT	2003/05/08 13:48
4	L12	165	(traffic flow) adj3 (database data adj base) same (updat\$3 add\$3 edit\$3)	USPAT	2003/05/08 13:47
5	L13	165	10 and 12	USPAT	2003/05/08 13:49
6	L14	2	11 and 13	USPAT	2003/05/08 13:48
7	L15	91	(traffic flow) adj3 (database data adj base) same ((last previous\$2) with (compar\$6 match\$3 map\$4) statistic\$2 analy\$4)	USPAT	2003/05/08 13:49
8	L16	30	15 and 12	USPAT	2003/05/08 13:49
9	L17	11	16 and (id identif\$4) near5 (packet cell frame segment)	USPAT	2003/05/08 13:51
10	L2	4774	(traffic flow) with (database data adj base)	USPAT	2003/05/08 13:43
11	L3	819	1 and 2	USPAT	2003/05/08 12:49
12	L4	30139	analy\$4 with (flow traffic)	USPAT	2003/05/08 12:50
13	L5	64	3 and 4 and statistic\$3	USPAT	2003/05/08 12:51
14	L6	9	5 and (id identif\$4) near5 (packet cell fram segment)	USPAT	2003/05/08 13:49
15	L7	3829	flow adj (entry input)	USPAT	2003/05/08 13:25
16	L8	14	stream same flow same packet same database	USPAT	2003/05/08 13:29
17	L9	3	4 and 8	USPAT	2003/05/08 13:43

	L #	Hits	Search Text
1	L1	8016	(identif\$4 determin\$3 verif\$4 filter\$3 confirm\$3 monitor\$3 analyz\$4) with (signal packet cell frame segment datagram portion) with (flow stream exist current\$2) with (new updat\$3 prior previous old)
2	L2	17393	monitor\$3 with (statistic\$4 database record history catalog)
3	L3	387	1 and 2
4	L4	1140	monitor\$3 with (statistic\$4 database record history catalog).ab.
5	L5	16	1 and 4
6	L6	155	(identif\$4 determin\$3 verif\$4 filter\$3 confirm\$3 monitor\$3 analyz\$4) with (signal packet cell frame segment datagram portion) with (flow stream) near5 (exist current\$2) with (new updat\$3 prior previous old) and monitor\$3 near5 (statistic\$4 database record history catalog)
7	L7	2	(identif\$4 determin\$3 verif\$4 filter\$3 confirm\$3 monitor\$3 analyz\$4) with (signal packet cell frame segment datagram portion) with (flow stream) near5 (exist current\$2) with (new updat\$3 prior previous old) and monitor\$3 near5 (statistic\$4 database record history catalog)
8	L8	17	(identif\$4 determin\$3 verif\$4 filter\$3 monitor\$3 analyz\$4) near5 (signal packet cell frame segment datagram portion) and (flow stream) near5 (exist current\$2) and (flow stream) near5 (new updat\$3) and (flow stream) near5 (prior previous old) and monitor\$3 near5 (statistic\$4 database record history catalog)
9	L9	15	(identif\$4 determin\$3 verif\$4 filter\$3 monitor\$3 analyz\$4) near5 (packet cell frame segment datagram portion header message) and (flow stream) near5 (exist current\$2) and (flow stream) near5 (new updat\$3) and (flow stream) near5 (prior previous old) and monitor\$3 near5 (statistic\$4 database record history catalog)
10	L10	1925	(identif\$4 determin\$3 verif\$4 filter\$3 monitor\$3 analyz\$4) near5 (packet cell frame segment datagram portion header message) and pattern and monitor\$3 near5 (statistic\$4 database record history catalog)
11	L11	1068	10 and (current exist\$3) near3 (flow stream data information packet cell frame datagram)
12	L12	372	10 and (current exist\$3) near3 (flow packet cell frame datagram)
13	L13	248	12 and (realtime real adj time)
14	L14	17	13 and flow near3 (insert\$3 entry input\$4)

	L #	Hits	Search Text
15	L15	445	(flow\$3 entry) near10 database same (identif\$4 determin\$3 verif\$4 filter\$3 monitor\$3 analyz\$4) with (packet cell frame segment datagram messag\$3)
16	L16	156253	(exist\$3 current\$2) with (packet cell frame segment datagram messag\$3)
17	L17	235052	(new add\$2 addition) with (packet cell frame segment datagram messag\$3)
18	L18	258	15 and 16 and 17
19	L19	46	15 and 16 and 17 and statistic\$2
20	L20	168591	(previous prior old\$3 last latest) with (packet cell frame segment datagram messag\$3)
21	L21	34	19 and 20
22	L22	8	19 and 20 and updat\$3 with flow
23	L23	455017	(identif\$4 determin\$3 verif\$4 filter\$3 monitor\$3 analyz\$4) with (packet cell frame segment datagram portion header message)
24	L24	555900	(identif\$4 determin\$3 verif\$4 filter\$3 monitor\$3 analyz\$4 detect\$4 discover\$3) with (packet cell frame segment datagram portion header message)
25	L25	273	(data information verison traffic stream entry) near3 flow same (previous prior old\$3 last\$3 late\$3) same (new added addition\$2) same (database history statistic\$4 record catalog log index\$3)
26	L26	184	24 and 25
27	L27	50	(data information verison traffic stream entry) near3 flow same (previous prior old\$3 last\$3 late\$3) same (new added addition\$2) same updat\$3 same (database history statistic\$4 record catalog log index\$3)
28	L28	36	27 not ("2000" "2001" "2002" "2003" "2004").ay.
29	L29	36	28 and 25
30	L30	27	28 and 24
31	L31	15	(identif\$4 determin\$3 verif\$4 filter\$3 monitor\$3 analyz\$4 detect\$4 discover\$3) with (packet cell frame segment datagram portion header message) same (data information version traffic entry) near5 flow same (previous prior old\$3 last\$3 late\$3) same (new added addition\$2) same updat\$3 same (database history statistic\$4 record catalog log index\$3)

	L #	Hits	Search Text
32	L32	108	(identif\$4 determin\$3 verif\$4 filter\$3 monitor\$3 analyz\$4 detect\$4 discover\$3) same (packet cell frame segment datagram portion header message) same flow same (previous prior old\$3 last\$3 late\$3) same (new added addition\$2) same updat\$3 same (database history statistic\$4 record catalog log index\$3)
33	L33	467588	(identif\$4 determin\$3 verif\$4 filter\$3 monitor\$3 analyz\$4 detect\$4 discover\$3).ab.
34	L34	58	32 and 33
35	L35	45	34 not ("2000" "2001" "2002" "2003" "2004").ay.
36	L36	11	(identif\$4 determin\$3 verif\$4 filter\$3 monitor\$3 analyz\$4 detect\$4 discover\$3) same (packet cell frame segment datagram portion header message) same flow with (entry input) same (previous prior old\$3 last\$3 late\$3) same (new added addition\$2) same updat\$3 same (database history statistic\$4 record catalog log index\$3)
37	L37	634	(identif\$4 determin\$3 verif\$4 filter\$3 monitor\$3 analyz\$4 detect\$4 discover\$3) with (packet cell frame segment datagram portion header message flow) and flow with (entry input) and (previous prior old\$3 last\$3 late\$3) same (new added addition\$2) same updat\$3 same (database history statistic\$4 record catalog log index\$3)
38	L38	240	(identif\$4 determin\$3 verif\$4 filter\$3 monitor\$3 analyz\$4 detect\$4 discover\$3) with (packet cell frame segment datagram portion header message flow) and flow with (entry) and (previous prior old\$3 last\$3 late\$3) same (new added addition\$2) same updat\$3 same (database history statistic\$4 record catalog log index\$3)
39	L39	86	(identif\$4 determin\$3 verif\$4 filter\$3 monitor\$3 analyz\$4 detect\$4 discover\$3) with (packet cell frame segment datagram portion header message flow) and flow with (entry) with (database history statistic\$4 record catalog log index\$3) and (previous prior old\$3 last\$3 late\$3) same (new added addition\$2) same updat\$3
40	L40	61	(identif\$4 determin\$3 verif\$4 filter\$3 monitor\$3 analyz\$4 detect\$4 discover\$3) with (packet cell frame segment datagram portion header message flow) and flow near10 (entry) with (database history statistic\$4 record catalog log index\$3) and (previous prior old\$3 last\$3 late\$3) same (new added addition\$2) same updat\$3
41	L41	55	40 not ("2000" "2001" "2002" "2003" "2004").ay.

	L #	Hits	Search Text
42	L42	24	(identif\$4 determin\$3 verif\$4 filter\$3 monitor\$3 analyz\$4 detect\$4 discover\$3) with (packet cell frame segment datagram portion header message flow) and flow near10 (entry) with (database history statistic\$4 record catalog log index\$3) and (previous prior old\$3 last\$3 late\$3) same (new added addition\$2) same updat\$3 with (database history statistic\$4 record catalog log index\$3)
43	L43	22	42 not ("2000" "2001" "2002" "2003" "2004").ay.
44	L44	24	(identif\$4 determin\$3 verif\$4 filter\$3 monitor\$3 analyz\$4 detect\$4 discover\$3) with (packet cell frame segment datagram portion header message flow) and flow near10 (entry) with (database history statistic\$4 record catalog log index\$3) and (packet cell frame segment datagram portion header message flow) with (previous prior old\$3 last\$3 late\$3) and (packet cell frame segment datagram portion header message flow) with (new added addition\$2) same updat\$3 with (database history statistic\$4 record catalog log index\$3)
45	L45	5	(identif\$4 determin\$3 verif\$4 filter\$3 monitor\$3 analyz\$4 detect\$4 discover\$3) with (packet cell frame segment datagram portion header message) same flow with (entry input) same (previous prior old\$3 last\$3 late\$3) same (new added addition\$2) and updat\$3 with (database history record catalog log index\$3 repository) and statistic\$4

VALID

Pat. No. 06014869 - 9
Issue Date: 09/27/04

Group ID: E Page 1
User ID: sjvaugh KS: 1,580

Warning [Pages Of Foreign References:]

page 1 has no references

page 2 has no references

page 3 has no references

Warning [Pages Of Other References:]

page 3 has no references

A large, bold, black handwritten letter 'E' is positioned in the upper right quadrant of the page. The letter is slightly tilted and has a thick, brush-like appearance.

Pat. No. 06014869 - 9
Issue Date: 09/27/04

Group ID: E
User ID: sjvaugh

CHECK LIST

<u>Rule 47</u>	<u>Continuing Data</u>	<u>PCT</u>	<u>Disclaimer</u>
No	Yes	No	NO
<u>Microfiche Appendix</u>		<u>CPA tag</u>	
No		No	

Foreign Priority Claimed: No
Acknowledged: No

State Code: CA Country Code:

Text Endorsement: 09608126.063000

=====

JACKET

<u>SERIAL NUMBER</u>	<u>FILING DATE</u>	<u>CLASS</u>	<u>SUBCLASS</u>	<u>GAU</u>
09/608,126	06/30/00	709	224	2142

FOREIGN PRIORITY

<u>Country</u>	<u>Document Number</u>	<u>Date</u>
----------------	------------------------	-------------

DISCLAIMER

/ /

TITLE

Re-using information from data transactions for maintaining statistics
in network monitoring

MICROFICHE APPENDIX

ASSISTANT EXAMINER:

<u>First:</u>	<u>Middle:</u>	<u>Last:</u>
---------------	----------------	--------------

PRIMARY EXAMINER:

<u>First:</u>	<u>Middle:</u>	<u>Last:</u>
---------------	----------------	--------------

Thong

Vu

CLAIMS ALLOWED
Total Print

21 1

DRAWINGS
Sheets Figures Print

18 20 y

=====
BLUE SLIP INFORMATION

<u>SERIAL NUMBER</u>	<u>CLASS</u>	<u>SUBCLASS</u>	<u>GAU</u>
09/608,126	709	224	2142

INDEP. CLAIMS

1,17

TOTAL CLAIMS

21

=====
BLUE SLIP (Page 1)

INTERNATIONAL CLASSIFICATION

<u>Class</u>	<u>SubClass</u>
G06F	15/173

CROSS-REFERENCES


<u>Class</u>	<u>SubClass</u>
709	223;230

=====
TERM EXTENSION

655
FIELD OF SEARCH

<u>Class</u>	<u>SubClass</u>
709	223;224;231;232;230
370	252;231

3
0
1



Pat. No. 06014869 - 9
Issue Date: 09/27/04

Group ID: E
User ID: sjvaugh

Page 3

379 32
704 43
714 39
340 825

=====

OATH

INVENTOR NAME

First: Russell Middle: S. Last: Dietz Signed: Yes
City: San Jose
State: CA ZIP Code: Country: Foreign ZIP:

INVENTOR NAME

First: Joseph Middle: R. Last: Maixner Signed: Yes
City: Aptos
State: CA ZIP Code: Country: Foreign ZIP:

INVENTOR NAME

First: Andrew Middle: A. Last: Koppenhaver Signed: Yes
City: Littleton
State: CO ZIP Code: Country: Foreign ZIP:

=====

PCT INFO

=====

CONTINUING DATA (Page 1)

LINE CODE SERIAL NUMBER FILING DATE STATUS DOCUMENT NO. ISSUE DATE

379 32
704 43
714 39
340 825

=====
OATH
INVENTOR NAME
First: Middle: Last: Signed:
Russell S. Dietz Yes
City: San Jose
State: CA ZIP Code: Country: Foreign ZIP:

INVENTOR NAME
First: Middle: Last: Signed:
Joseph R. Maixner Yes
City: Aptos
State: CA ZIP Code: Country: Foreign ZIP:

INVENTOR NAME
First: Middle: Last: Signed:
Andrew A. Koppenhaver Yes
City: Littleton
State: CO ZIP Code: Country: Foreign ZIP:

=====
PCT INFO
=====

CONTINUING DATA (Page 1)

LINE CODE SERIAL NUMBER FILING DATE STATUS DOCUMENT NO. ISSUE DATE

104 68 60/141,903 06/30/1999 / /

=====

REFERENCES (Page 1) SERIAL NUMBER: 09/608,126
FORM 892

U.S. REFERENCES

<u>U.S. Pat No.</u>	<u>Date</u>	<u>Patentee</u>	<u>Class</u>	<u>SubClass</u>
*6,625,657	09/2003	Bullard	709	237
No issue date available.				
*6,330,226	12/2001	Chapman et al.	370	232
No issue date available.				
*6,651,099	11/2003	Dietz et al.	709	224
No issue date available.				
*6,424,624	07/2002	Galand et al.	370	231
No issue date available.				
*6,279,113	08/2001	Vaidya	713	201
No issue date available.				
*6,363,056	03/2002	Beigi et al.	370	252
No issue date available.				
*6,115,393	09/2000	Engel et al.	370	469
No issue date available.				
4,972,453	11/1990	Daniel, III et al.	379	9.03
5,535,338	07/1996	Krause et al.	709	222
5,802,054	09/1998	Bellenger	370	401
5,720,032	02/1998	Picazo, Jr. et al.	709	250

FOREIGN REFERENCES

<u>Foreign Doc No.</u>	<u>Date</u>	<u>Country</u>	<u>Class</u>	<u>SubClass</u>
------------------------	-------------	----------------	--------------	-----------------

OTHER REFERENCE CITATIONS (incl. Author, Title, Date, Pertinent Pages, etc.)

NOV94: Packet Filtering in the SNMP Remote Monitor ;

www.skrymir.com/dobbs/articles/1994/9411/9411h/9411h.htm.

GTrace+13 A Graphical Traceroute Tool authored by Ram Periakaruppan,

Evi Nemeth ;

<http://www.caida.org/outreach/papers/1999/GTrace/index.xml>.

=====

REFERENCES (Page 2) SERIAL NUMBER: 09/608,126
FORM 892

U.S. REFERENCES

<u>U.S. Pat No.</u>	<u>Date</u>	<u>Patentee</u>	<u>Class</u>	<u>SubClass</u>
5,850,388	12/1998	Anderson et al.	370	252
*6,097,699	08/2000	Chen et al.	370	231
No issue date available.				
*6,269,330	07/2001	Cidon et al.	704	43
No issue date available.				
*6,453,345	09/2002	Trcka et al.	709	224
No issue date available.				
*6,381,306	04/2002	Lawson et al.	379	32
No issue date available.				
*6,282,570	08/2001	Leung et al.	709	224
No issue date available.				
5,761,429	06/1998	Thompson	709	224
5,799,154	08/1998	Kuriyan	709	223

FOREIGN REFERENCES

Foreign Doc No. Date Country Class SubClass

OTHER REFERENCE CITATIONS (incl. Author, Title, Date, Pertinent Pages, etc.)

Advanced Methods for Storage and Retrieval in Image ;

<http://www.cs.tulane.edu/www/Prototype/proposal.html>; 1998.

Measurement and analysis of the digital DECT propagation channel; IEEE
1998.

=====

REFERENCES (Page 3) SERIAL NUMBER: 09/608,126

Pat. No. 06014869 - 9
Issue Date: 09/27/04

Group ID: E
User ID: sjvaugh

Page 6

FORM 1449

U.S. REFERENCES

<u>U.S. Pat No.</u>	<u>Date</u>	<u>Patentee</u>	<u>Class</u>	<u>SubClass</u>
5,703,877	12/1997	Nuber et al.	370	395
5,892,754	04/1999	Kompella et al.	370	236

FOREIGN REFERENCES

<u>Foreign Doc No.</u>	<u>Date</u>	<u>Country</u>	<u>Class</u>	<u>SubClass</u>
------------------------	-------------	----------------	--------------	-----------------

OTHER REFERENCE CITATIONS (incl. Author, Title, Date, Pertinent Pages, etc.)

=====

Paser 1-12
Mission

Page 3

MISSING

PASER ARE

MISS 14.

Page 2

MISSING

Pasar 3
Miss.