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 (1) OF (3) PART(S)

09/608126
06/30/00

ISSUE CLASSIFICATION	
Class	Subclass
709	224

6839751

U.S. UTILITY Patent Application

FF O.I.P.E. SCANNED BK③ Q.A. CC	PATENT DATE JAN 0 4 2005

APPLICATION NO.	CONT/PRIOR	CLASS	SUBCLASS	ART UNIT	EXAMINER
09/608126	D	709	~~[illegible]~~ 224	~~2151~~ 2142	THONG VU

APPLICANTS
Russell Dietz
Joseph Maixner
Andrew Koppenhaver

TITLE
Re-using information from data transactions for maintaining statistics in network monitoring

Certificate
MAR 0 8 2005
of Correction

PTO-2040
12/99

ISSUING CLASSIFICATION

ORIGINAL		CROSS REFERENCE(S)						
CLASS	SUBCLASS	CLASS	SUBCLASS (ONE SUBCLASS PER BLOCK)					
709	224	709	223	230				
INTERNATIONAL CLASSIFICATION								
G06F	15/173							

□ Continued on Issue Slip Inside File Jacket

Formal Drawings (18 shts) set 11-30-04

Formal Drawings (shts) set 6-30-00

□ TERMINAL DISCLAIMER	DRAWINGS			CLAIMS ALLOWED	
	Sheets Drwg.	Figs. Drwg.	Print Fig.	Total Claims	Print Claim for O.G.
	18	20	1	21	1

□ The term of this patent subsequent to ____________ (date) has been disclaimed.

(Assistant Examiner) (Date)

NOTICE OF ALLOWANCE MAILED
6-4-04

□ The term of this patent shall not extend beyond the expiration date of U.S Patent. No. ____________

(Primary Examiner) (Date)

ISSUE FEE

Amount Due	Date Paid
$1330.00	6 23 04 JPM

□ The terminal ____ months of this patent have been disclaimed.

(Legal Instruments Examiner) (Date)

ISSUE BATCH NUMBER

WARNING:
The information disclosed herein may be restricted. Unauthorized disclosure may be prohibited by the United States Code Title 35, Sections 122, 181 and 368. Possession outside the U.S. Patent & Trademark Office is restricted to authorized employees and contractors only.

Form PTO-436A
(Rev. 6/99)

FILED WITH: □ DISK (CRF) □ FICHE □ CD-ROM
(Attached in pocket on right inside flap)

ISSUE FEE IN FILE

(FACE)

UNITED STATES PATENT AND TRADEMARK OFFICE

COMMISSIONER FOR PATENTS
UNITED STATES PATENT AND TRADEMARK OFFICE
WASHINGTON, D.C. 20231
www.uspto.gov

Bib Data Sheet

SERIAL NUMBER 09/608,126	**FILING DATE** 06/30/2000 **RULE** _	**CLASS** 709	**GROUP ART UNIT** 2755	**ATTORNEY DOCKET NO.** APPT-001-3

APPLICANTS

Russell S. Dietz, San Jose, CA ;
Joseph R. Maixner, Aptos, CA ;
Andrew A. Koppenhaver, Fairfax, VA ;

** **CONTINUING DATA** *************************

THIS APPLN CLAIMS BENEFIT OF 60/141,903 06/30/1999

** **FOREIGN APPLICATIONS** ********************

IF REQUIRED, FOREIGN FILING LICENSE GRANTED ** 08/21/2000

Foreign Priority claimed □ yes ☑ no 35 USC 119 (a-d) conditions met □ yes ☒ no □ Met after Allowance Verified and Acknowledged ______ Examiner's Signature Initials	**STATE OR COUNTRY** CA	**SHEETS DRAWING** 18	**TOTAL CLAIMS** 21	**INDEPENDENT CLAIMS** 2

ADDRESS

Dov Rosenfeld
Suite 2
5507 College Avenue
Oakland ,CA 94618

TITLE

Re-using information from data transactions for maintaining statistics in network monitoring

FILING FEE RECEIVED 858	FEES: Authority has been given in Paper No. ________ to charge/credit DEPOSIT ACCOUNT No. ________ for following:	□ All Fees □ 1.16 Fees (Filing) □ 1.17 Fees (Processing Ext. of time) □ 1.18 Fees (Issue) □ Other ____________ □ Credit

Our Ref./Docket No.: APPT-001-3

RE-USING INFORMATION FROM DATA TRANSACTIONS FOR MAINTAINING STATISTICS IN NETWORK MONITORING

Inventor(s):

DIETZ, Russell S.
San Jose, CA

MAIXNER, Joseph R.
Aptos, CA

KOPPENHAVER, Andrew A.
Fairfax, VA

Certificate of Mailing under 37 CFR 1.10

I hereby certify that this application and all attachments are being deposited with the United States Postal Service as Express Mail (Express Mail Label: EI417961927US in an envelope addressed to Box Patent Application, Assistant Commissioner for Patents, Washington, D.C. 20231 on.

Date: June 30, 2000

Signed: ______________________

Name: Dov Rosenfeld, Reg. No. 38687

RE-USING INFORMATION FROM DATA TRANSACTIONS FOR MAINTAINING STATISTICS IN NETWORK MONITORING

CROSS-REFERENCE TO RELATED APPLICATION

This application claims the benefit of U.S. Provisional Patent Application Serial No.: 60/141,903 for METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK to inventors Dietz, et al., filed June 30, 1999, the contents of which are incorporated herein by reference.

This application is related to the following U.S. patent applications, each filed concurrently with the present application, and each assigned to Apptitude, Inc., the assignee of the present invention:

U.S. Patent Application Serial No. 09/608237 for METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK, to inventors Dietz, et al., filed June 30, 2000, Attorney/Agent Reference Number APPT-001-1, and incorporated herein by reference.

U.S. Patent Application Serial No. 09/609179 for PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE, to inventors Koppenhaver, et al., filed June 30, 2000, Attorney/Agent Reference Number APPT-001-2, and incorporated herein by reference.

U.S. Patent Application Serial No. 09/608266 for ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR, to inventors Sarkissian, et al., filed June 30, 2000, Attorney/Agent Reference Number APPT-001-4, and incorporated herein by reference.

U.S. Patent Application Serial No. 09/608267 for STATE PROCESSOR FOR PATTERN MATCHING IN A NETWORK MONITOR DEVICE, to inventors Sarkissian, et al., filed June 30, 2000, Attorney/Agent Reference Number APPT-001-5, and incorporated herein by reference.

FIELD OF INVENTION

The present invention relates to computer networks, specifically to the real-time elucidation of packets communicated within a data network, including classification according to protocol and application program.

COPYRIGHT NOTICE

A portion of the disclosure of this patent document contains material that is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

BACKGROUND

There has long been a need for network activity monitors. This need has become especially acute, however, given the recent popularity of the Internet and other interconnected networks. In particular, there is a need for a real-time network monitor that can provide details as to the application programs being used. Such a monitor should enable non-intrusive, remote detection, characterization, analysis, and capture of all information passing through any point on the network (*i.e.*, of all packets and packet streams passing through any location in the network). Not only should all the packets be detected and analyzed, but for each of these packets the network monitor should determine the protocol (*e.g.*, http, ftp, H.323, VPN, etc.), the application/use within the protocol (*e.g.*, voice, video, data, real-time data, etc.), and an end user's pattern of use within each application or the application context (*e.g.*, options selected, service delivered, duration, time of day, data requested, etc.). Also, the network monitor should not be reliant upon server resident information such as log files. Rather, it should allow a user such as a network administrator or an Internet service provider (ISP) the means to measure and analyze network activity objectively; to customize the type of data that is collected and analyzed; to undertake real time analysis; and to receive timely notification of network problems.

Related and incorporated by reference U.S. Patent application 09/608237 for *METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK*, to inventors Dietz, et al, Attorney/Agent Docket APPT-001-1, describes a network monitor that includes carrying out protocol specific operations on individual packets including extracting information from header fields in the packet to use for building a signature for identifying the conversational flow of the packet and for recognizing future packets as belonging to a previously encountered flow. A parser subsystem includes a parser for recognizing different patterns in the packet that identify the protocols used. For each protocol recognized, a slicer extracts important packet elements from the packet. These form a signature (*i.e.*, key) for the packet. The slicer also preferably generates a hash for rapidly identifying a flow that may have this signature from a database of known flows.

The flow signature of the packet, the hash and at least some of the payload are passed to an analyzer subsystem. In a hardware embodiment, the analyzer subsystem includes a unified flow key buffer (UFKB) for receiving parts of packets from the parser subsystem and for storing signatures in process, a lookup/update engine (LUE) to lookup a database of flow records for previously encountered conversational flows to determine whether a signature is from an existing flow, a state processor (SP) for performing state processing, a flow insertion and deletion engine (FIDE) for inserting new flows into the database of flows, a memory for storing the database of flows, and a cache for speeding up access to the memory containing the flow database. The LUE, SP, and FIDE are all coupled to the UFKB, and to the cache.

Each flow-entry includes one or more statistical measures, e.g., the packet count related to the flow, the time of arrival of a packet, the time differential.

In the preferred hardware embodiment, each of the LUE, state processor, and FIDE operate independently from the other two engines. The state processor performs one or more operations specific to the state of the flow.

It is advantageous to collect statistics on packets passing through a point in a network rather than to simply count each and every packet. By maintaining statistical measures in the flow-entries related to a conversational flow, embodiments of the present invention enable specific metrics to be collected in real-time that otherwise would not be possible. For example, it is desirable to maintain metrics related to bi-directional conversations based on the entire flow for each exchange in the conversation. By maintaining the state of flow, embodiments of the present invention also enable certain metrics related to the states of flows to be determined.

Most prior-art network traffic monitors that use statistical metrics collect only end-point and end-of-session related statistics. Examples of such commonly used metrics include packet counts, byte counts, session connection time, session timeouts, session and transport response times and others. All of these deal with events that can be directly related to an event in a single packet. These prior-art systems cannot collect some important performance metrics that are related to a complete sequence of packets of a flow or to several disjointed sequences of the same flow in a network.

Time based metrics on application data packets are important. Such metrics could be determined if all the timestamps and related data could be stored and forwarded for later analysis. However when faced with thousands or millions of conversations per second on ever faster networks, storing all the data, even if compressed, would take too much processing, memory, and manager down load time to be practical.

Thus there is a need for maintaining and reporting time-base metrics from statistical measures accumulated from packets in a flow.

Network data is properly modeled as a population and not a sample. Thus, all the data needs to be processed. Because of the nature of application protocols, just sampling some of the packets may not give good measured related to flows. Missing just one critical packet, such as one the specified an additional port that data will be transmitted on, or what application will be run, can cause valid data to be lost.

Thus there is also a need for maintaining and reporting time-base metrics from statistical measures accumulated from *every* packet in a flow.

There also is a need to determine metrics related to a sequence of events. A good example is relative jitter. Measuring the time from the end of one packet in one direction to another packet with the same signature in the same direction collects data that relates normal jitter. This type of jitter metric is good for measuring broad signal quality in a packet network. However, it is not specific to the payload or data item being transported in a cluster of packets.

Using the state processing described herein, because the state processor can search for specific data payloads, embodiments of monitor 300 can be programmed to collect the same jitter metric for a group of packets in a flow that are all related to a specific data payload. This allows the inventive system to provide metrics more focused on the type of quality related to a set of packets. This in general is more desirable than metrics related to single packets when evaluating the performance of a system in a network.

Specifically, the monitor system 300 can be programmed to maintain any type of metric at any state of a conversational flow. Also the system 300 can have the actual statistics programmed into the state at any point. This enables embodiments of the monitor system to collect metrics related to network usage and performance, as well as metrics related to specific states or sequences of packets.

Some of the specific metrics that can be collected only with states are events related to a group of traffic in one direction, events related to the status of a communication sequence in one or both directions, events related to the exchange of packets for a specific application in a specific sequence. This is only a small sample of the metrics that requires an engine that can relate the state of a flow to a set of metrics.

In addition, because the monitor 300 provides greater visibility to the specific application in a conversation or flow, the monitor 300 can be programmed to collect metrics that may be specific to that type of application or service. In other word, if a flow is for an Oracle Database server, an embodiment of monitor 300 could collect the number of packets required to complete a transaction. Only with both state and application classification can this type of metric be derived from the network.

Because the monitor 300 can be programmed to collect a diverse set of metrics, the system can be used as a data source for metrics required in a number of environments. In particular, the metrics may be used to monitor and analyze the quality and performance of traffic flows related to a specific set of applications. Other implementation could include metrics related to billing and charge-back for specific traffic flow and events with the traffic flows. Yet other implementations could be programmed to provide metrics useful for troubleshooting and capacity planning and related directly to a focused application and service.

SUMMARY

Another aspect of the invention is determining quality of service metrics based on each and every packet. A method of and monitor apparatus for analyzing a flow of packets passing through a connection point on a computer network are disclosed that may include such quality of service metrics. The method includes receiving a packet from a packet acquisition device, and looking up a flow-entry database containing flow-entries for previously encountered conversational flows. The looking up to determine if the received packet is of an existing flow. Each and every packet is processed. If the packet is of an existing flow, the method updates the flow-entry of the existing flow, including storing one or more statistical measures kept in the flow-entry. If the packet is of a new flow, the method stores 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. The statistical measures are used to determine metrics related to the flow. The metrics may be base metrics from which quality of service metrics are determined, or may be the quality of service metrics.

BRIEF DESCRIPTION OF THE DRAWINGS

Although the present invention is better understood by referring to the detailed preferred embodiments, these should not be taken to limit the present invention to any specific embodiment because such embodiments are provided only for the purposes of explanation. The embodiments, in turn, are explained with the aid of the following figures.

FIG. 1 is a functional block diagram of a network embodiment of the present invention in which a monitor is connected to analyze packets passing at a connection point.

FIG. 2 is a diagram representing an example of some of the packets and their formats that might be exchanged in starting, as an illustrative example, a conversational flow between a client and server on a network being monitored and analyzed. A pair of flow signatures particular to this example and to embodiments of the present invention is also illustrated. This represents some of the possible flow signatures that can be generated and used in the process of analyzing packets and of recognizing the particular server applications that produce the discrete application packet exchanges.

FIG. 3 is a functional block diagram of a process embodiment of the present invention that can operate as the packet monitor shown in FIG. 1. This process may be implemented in software or hardware.

FIG. 4 is a flowchart of a high-level protocol language compiling and optimization process, which in one embodiment may be used to generate data for monitoring packets according to versions of the present invention.

FIG. 5 is a flowchart of a packet parsing process used as part of the parser in an embodiment of the inventive packet monitor.

FIG. 6 is a flowchart of a packet element extraction process that is used as part of the parser in an embodiment of the inventive packet monitor.

FIG. 7 is a flowchart of a flow-signature building process that is used as part of the parser in the inventive packet monitor.

FIG. 8 is a flowchart of a monitor lookup and update process that is used as part of the analyzer in an embodiment of the inventive packet monitor.

FIG. 9 is a flowchart of an exemplary Sun Microsystems Remote Procedure Call application than may be recognized by the inventive packet monitor.

FIG. 10 is a functional block diagram of a hardware parser subsystem including the pattern recognizer and extractor that can form part of the parser module in an embodiment of the inventive packet monitor.

FIG. 11 is a functional block diagram of a hardware analyzer including a state processor that can form part of an embodiment of the inventive packet monitor.

FIG. 12 is a functional block diagram of a flow insertion and deletion engine process that can form part of the analyzer in an embodiment of the inventive packet monitor.

FIG. 13 is a flowchart of a state processing process that can form part of the analyzer in an embodiment of the inventive packet monitor.

FIG. 14 is a simple functional block diagram of a process embodiment of the present invention that can operate as the packet monitor shown in FIG. 1. This process may be implemented in software.

FIG. 15 is a functional block diagram of how the packet monitor of FIG. 3 (and FIGS. 10 and 11) may operate on a network with a processor such as a microprocessor.

FIG. 16 is an example of the top (MAC) layer of an Ethernet packet and some of the elements that may be extracted to form a signature according to one aspect of the invention.

FIG. 17A is an example of the header of an Ethertype type of Ethernet packet of FIG. 16 and some of the elements that may be extracted to form a signature according to one aspect of the invention.

FIG. 17B is an example of an IP packet, for example, of the Ethertype packet shown in FIGs. 16 and 17A, and some of the elements that may be extracted to form a signature according to one aspect of the invention.

FIG. 18A is a three dimensional structure that can be used to store elements of the pattern, parse and extraction database used by the parser subsystem in accordance to one embodiment of the invention.

FIG. 18B is an alternate form of storing elements of the pattern, parse and extraction database used by the parser subsystem in accordance to another embodiment of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Note that this document includes hardware diagrams and descriptions that may include signal names. In most cases, the names are sufficiently descriptive, in other cases however the signal names are not needed to understand the operation and practice of the invention.

Operation in a Network

FIG. 1 represents a system embodiment of the present invention that is referred to herein by the general reference numeral 100. The system 100 has a computer network

102 that communicates packets (*e.g.*, IP datagrams) between various computers, for example between the clients 104–107 and servers 110 and 112. The network is shown schematically as a cloud with several network nodes and links shown in the interior of the cloud. A monitor 108 examines the packets passing in either direction past its connection point 121 and, according to one aspect of the invention, can elucidate what application programs are associated with each packet. The monitor 108 is shown examining packets (*i.e.*, datagrams) between the network interface 116 of the server 110 and the network. The monitor can also be placed at other points in the network, such as connection point 123 between the network 102 and the interface 118 of the client 104, or some other location, as indicated schematically by connection point 125 somewhere in network 102. Not shown is a network packet acquisition device at the location 123 on the network for converting the physical information on the network into packets for input into monitor 108. Such packet acquisition devices are common.

Various protocols may be employed by the network to establish and maintain the required communication, *e.g.*, TCP/IP, etc. Any network activity—for example an application program run by the client 104 (CLIENT 1) communicating with another running on the server 110 (SERVER 2)—will produce an exchange of a sequence of packets over network 102 that is characteristic of the respective programs and of the network protocols. Such characteristics may not be completely revealing at the individual packet level. It may require the analyzing of many packets by the monitor 108 to have enough information needed to recognize particular application programs. The packets may need to be parsed then analyzed in the context of various protocols, for example, the transport through the application session layer protocols for packets of a type conforming to the ISO layered network model.

Communication protocols are layered, which is also referred to as a protocol stack. The ISO (International Standardization Organization) has defined a general model that provides a framework for design of communication protocol layers. This model, shown in table form below, serves as a basic reference for understanding the functionality of existing communication protocols.

ISO MODEL

Layer	Functionality	Example
7	Application	Telnet, NFS, Novell NCP, HTTP, H.323
6	Presentation	XDR
5	Session	RPC, NETBIOS, SNMP, *etc.*
4	Transport	TCP, Novel SPX, UDP, *etc.*
3	Network	IP, Novell IPX, VIP, AppleTalk, *etc.*
2	Data Link	Network Interface Card (Hardware Interface). MAC layer
1	Physical	Ethernet, Token Ring, Frame Relay, ATM, T1 (Hardware Connection)

Different communication protocols employ different levels of the ISO model or may use a layered model that is similar to but which does not exactly conform to the ISO model. A protocol in a certain layer may not be visible to protocols employed at other layers. For example, an application (Level 7) may not be able to identify the source computer for a communication attempt (Levels 2–3).

In some communication arts, the term "frame" generally refers to encapsulated data at OSI layer 2, including a destination address, control bits for flow control, the data or payload, and CRC (cyclic redundancy check) data for error checking. The term "packet" generally refers to encapsulated data at OSI layer 3. In the TCP/IP world, the term "datagram" is also used. In this specification, the term "packet" is intended to encompass packets, datagrams, frames, and cells. In general, a packet format or frame format refers to how data is encapsulated with various fields and headers for transmission across a network. For example, a data packet typically includes an address destination field, a length field, an error correcting code (ECC) field, or cyclic redundancy check (CRC) field, as well as headers and footers to identify the beginning and end of the packet. The terms “packet format” and “frame format,” also referred to as “cell format,” are generally synonymous.

Monitor 108 looks at every packet passing the connection point 121 for analysis. However, not every packet carries the same information useful for recognizing all levels of the protocol. For example, in a conversational flow associated with a particular application, the application will cause the server to send a type-A packet, but so will another. If, though, the particular application program always follows a type-A packet with the sending of a type-B packet, and the other application program does not, then in order to recognize packets of that application’s conversational flow, the monitor can be available to recognize packets that match the type-B packet to associate with the type-A packet. If such is recognized after a type-A packet, then the particular application program’s conversational flow has started to reveal itself to the monitor 108.

Further packets may need to be examined before the conversational flow can be identified as being associated with the application program. Typically, monitor 108 is simultaneously also in partial completion of identifying other packet exchanges that are parts of conversational flows associated with other applications. One aspect of monitor 108 is its ability to maintain the state of a flow. The state of a flow is an indication of all previous events in the flow that lead to recognition of the content of all the protocol levels, *e.g.,* the ISO model protocol levels. Another aspect of the invention is forming a signature of extracted characteristic portions of the packet that can be used to rapidly identify packets belonging to the same flow.

In real-world uses of the monitor 108, the number of packets on the network 102 passing by the monitor 108’s connection point can exceed a million per second. Consequently, the monitor has very little time available to analyze and type each packet and identify and maintain the state of the flows passing through the connection point. The monitor 108 therefore masks out all the unimportant parts of each packet that will not contribute to its classification. However, the parts to mask-out will change with each packet depending on which flow it belongs to and depending on the state of the flow.

The recognition of the packet type, and ultimately of the associated application programs according to the packets that their executions produce, is a multi-step process within the monitor 108. At a first level, for example, several application programs will all produce a first kind of packet. A first "signature" is produced from selected parts of a packet that will allow monitor 108 to identify efficiently any packets that belong to the same flow. In some cases, that packet type may be sufficiently unique to enable the monitor to identify the application that generated such a packet in the conversational flow. The signature can then be used to efficiently identify all future packets generated in traffic related to that application.

In other cases, that first packet only starts the process of analyzing the conversational flow, and more packets are necessary to identify the associated application program. In such a case, a subsequent packet of a second type—but that potentially belongs to the same conversational flow—is recognized by using the signature. At such a second level, then, only a few of those application programs will have conversational flows that can produce such a second packet type. At this level in the process of classification, all application programs that are not in the set of those that lead to such a sequence of packet types may be excluded in the process of classifying the conversational flow that includes these two packets. Based on the known patterns for the protocol and for the possible applications, a signature is produced that allows recognition of any future packets that may follow in the conversational flow.

It may be that the application is now recognized, or recognition may need to proceed to a third level of analysis using the second level signature. For each packet, therefore, the monitor parses the packet and generates a signature to determine if this signature identified a previously encountered flow, or shall be used to recognize future packets belonging to the same conversational flow. In real time, the packet is further analyzed in the context of the sequence of previously encountered packets (the state), and of the possible future sequences such a past sequence may generate in conversational flows associated with different applications. A new signature for recognizing future packets may also be generated. This process of analysis continues until the applications are identified. The last generated signature may then be used to efficiently recognize future packets associated with the same conversational flow. Such an arrangement makes it possible for the monitor 108 to cope with millions of packets per second that must be inspected.

Another aspect of the invention is adding Eavesdropping. In alternative embodiments of the present invention capable of eavesdropping, once the monitor 108 has recognized the executing application programs passing through some point in the network 102 (for example, because of execution of the applications by the client 105 or server 110), the monitor sends a message to some general purpose processor on the network that can input the same packets from the same location on the network, and the processor then loads its own executable copy of the application program and uses it to read the content being exchanged over the network. In other words, once the monitor 108 has accomplished recognition of the application program, eavesdropping can commence.

The Network Monitor

FIG. 3 shows a network packet monitor 300, in an embodiment of the present invention that can be implemented with computer hardware and/or software. The system 300 is similar to monitor 108 in FIG. 1. A packet 302 is examined, *e.g.,* from a packet acquisition device at the location 121 in network 102 (FIG. 1), and the packet evaluated, for example in an attempt to determine its characteristics, *e.g.,* all the protocol information in a multilevel model, including what server application produced the packet.

The packet acquisition device is a common interface that converts the physical signals and then decodes them into bits, and into packets, in accordance with the particular network (Ethernet, frame relay, ATM, *etc.*). The acquisition device indicates to the monitor 108 the type of network of the acquired packet or packets.

Aspects shown here include: (1) the initialization of the monitor to generate what operations need to occur on packets of different types—accomplished by compiler and optimizer 310, (2) the processing—parsing and extraction of selected portions—of packets to generate an identifying signature—accomplished by parser subsystem 301, and (3) the analysis of the packets—accomplished by analyzer 303.

The purpose of compiler and optimizer 310 is to provide protocol specific information to parser subsystem 301 and to analyzer subsystem 303. The initialization occurs prior to operation of the monitor, and only needs to re-occur when new protocols are to be added.

A flow is a stream of packets being exchanged between any two addresses in the network. For each protocol there are known to be several fields, such as the destination (recipient), the source (the sender), and so forth, and these and other fields are used in monitor 300 to identify the flow. There are other fields not important for identifying the flow, such as checksums, and those parts are not used for identification.

Parser subsystem 301 examines the packets using pattern recognition process 304 that parses the packet and determines the protocol types and associated headers for each protocol layer that exists in the packet 302. An extraction process 306 in parser subsystem 301 extracts characteristic portions (signature information) from the packet 302. Both the pattern information for parsing and the related extraction operations, *e.g.*, extraction masks, are supplied from a parsing-pattern-structures and extraction-operations database (parsing/extractions database) 308 filled by the compiler and optimizer 310.

The protocol description language (PDL) files 336 describes both patterns and states of all protocols that an occur at any layer, including how to interpret header information, how to determine from the packet header information the protocols at the next layer, and what information to extract for the purpose of identifying a flow, and ultimately, applications and services. The layer selections database 338 describes the particular layering handled by the monitor. That is, what protocols run on top of what protocols at any layer level. Thus 336 and 338 combined describe how one would decode, analyze, and understand the information in packets, and, furthermore, how the information is layered. This information is input into compiler and optimizer 310.

When compiler and optimizer 310 executes, it generates two sets of internal data structures. The first is the set of parsing/extraction operations 308. The pattern structures include parsing information and describe what will be recognized in the headers of packets; the extraction operations are what elements of a packet are to be extracted from the packets based on the patterns that get matched. Thus, database 308 of parsing/extraction operations includes information describing how to determine a set of one or more protocol dependent extraction operations from data in the packet that indicate a protocol used in the packet.

The other internal data structure that is built by compiler 310 is the set of state patterns and processes 326. These are the different states and state transitions that occur in different conversational flows, and the state operations that need to be performed (*e.g.*, patterns that need to be examined and new signatures that need to be built) during any state of a conversational flow to further the task of analyzing the conversational flow.

Thus, compiling the PDL files and layer selections provides monitor 300 with the information it needs to begin processing packets. In an alternate embodiment, the contents of one or more of databases 308 and 326 may be manually or otherwise generated. Note that in some embodiments the layering selections information is inherent rather than explicitly described. For example, since a PDL file for a protocol includes the child protocols, the parent protocols also may be determined.

In the preferred embodiment, the packet 302 from the acquisition device is input into a packet buffer. The pattern recognition process 304 is carried out by a pattern analysis and recognition (PAR) engine that analyzes and recognizes patterns in the packets. In particular, the PAR locates the next protocol field in the header and determines the length of the header, and may perform certain other tasks for certain types of protocol headers. An example of this is type and length comparison to distinguish an IEEE 802.3 (Ethernet) packet from the older type 2 (or Version 2) Ethernet packet, also called a DIGITAL-Intel-Xerox (DIX) packet. The PAR also uses the pattern structures and extraction operations database 308 to identify the next protocol and parameters associated with that protocol that enables analysis of the next protocol layer. Once a pattern or a set of patterns has been identified, it/they will be associated with a set of none or more extraction operations. These extraction operations (in the form of commands and associated parameters) are passed to the extraction process 306 implemented by an extracting and information identifying (EII) engine that extracts selected parts of the packet, including identifying information from the packet as required for recognizing this packet as part of a flow. The extracted information is put in sequence and then processed in block 312 to build a unique flow signature (also called a "key") for this flow. A flow signature depends on the protocols used in the packet. For some protocols, the extracted components may include source and destination addresses. For example, Ethernet frames have end-point addresses that are useful in building a better flow signature. Thus, the signature typically includes the client and server address pairs. The signature is used to recognize further packets that are or may be part of this flow.

In the preferred embodiment, the building of the flow key includes generating a hash of the signature using a hash function. The purpose if using such a hash is conventional—to spread flow-entries identified by the signature across a database for efficient searching. The hash generated is preferably based on a hashing algorithm and such hash generation is known to those in the art.

In one embodiment, the parser passes data from the packet—a parser record—that includes the signature (i.e., selected portions of the packet), the hash, and the packet itself to allow for any state processing that requires further data from the packet. An improved embodiment of the parser subsystem might generate a parser record that has some predefined structure and that includes the signature, the hash, some flags related to some of the fields in the parser record, and parts of the packet's payload that the parser subsystem has determined might be required for further processing, e.g., for state processing.

Note that alternate embodiments may use some function other than concatenation of the selected portions of the packet to make the identifying signature. For example, some "digest function" of the concatenated selected portions may be used.

The parser record is passed onto lookup process 314 which looks in an internal data store of records of known flows that the system has already encountered, and decides (in 316) whether or not this particular packet belongs to a known flow as indicated by the presence of a flow-entry matching this flow in a database of known flows 324. A record in database 324 is associated with each encountered flow.

The parser record enters a buffer called the unified flow key buffer (UFKB). The UFKB stores the data on flows in a data structure that is similar to the parser record, but that includes a field that can be modified. In particular, one or the UFKB record fields stores the packet sequence number, and another is filled with state information in the form of a program counter for a state processor that implements state processing 328.

The determination (316) of whether a record with the same signature already exists is carried out by a lookup engine (LUE) that obtains new UFKB records and uses the hash in the UFKB record to lookup if there is a matching known flow. In the particular embodiment, the database of known flows 324 is in an external memory. A cache is associated with the database 324. A lookup by the LUE for a known record is carried out by accessing the cache using the hash, and if the entry is not already present in the cache, the entry is looked up (again using the hash) in the external memory.

The flow-entry database 324 stores flow-entries that include the unique flow-signature, state information, and extracted information from the packet for updating flows, and one or more statistical about the flow. Each entry completely describes a flow. Database 324 is organized into bins that contain a number, denoted N, of flow-entries (also called flow-entries, each a bucket), with N being 4 in the preferred embodiment. Buckets (i.e., flow-entries) are accessed via the hash of the packet from the parser subsystem 301 (i.e., the hash in the UFKB record). The hash spreads the flows across the database to allow for fast lookups of entries, allowing shallower buckets. The designer selects the bucket depth N based on the amount of memory attached to the monitor, and the number of bits of the hash data value used. For example, in one embodiment, each flow-entry is 128 bytes long, so for 128K flow-entries, 16 Mbytes are required. Using a 16-bit hash gives two flow-entries per bucket. Empirically, this has been shown to be more than adequate for the vast majority of cases. Note that another embodiment uses flow-entries that are 256 bytes long.

Herein, whenever an access to database 324 is described, it is to be understood that the access is via the cache, unless otherwise stated or clear from the context.

If there is no flow-entry found matching the signature, i.e., the signature is for a new flow, then a protocol and state identification process 318 further determines the state and protocol. That is, process 318 determines the protocols and where in the state sequence for a flow for this protocol's this packet belongs. Identification process 318 uses the extracted information and makes reference to the database 326 of state patterns and processes. Process 318 is then followed by any state operations that need to be executed on this packet by a state processor 328.

If the packet is found to have a matching flow-entry in the database 324 (e.g., in the cache), then a process 320 determines, from the looked-up flow-entry, if more classification by state processing of the flow signature is necessary. If not, a process 322 updates the flow-entry in the flow-entry database 324 (e.g., via the cache). Updating includes updating one or more statistical measures stored in the flow-entry. In our embodiment, the statistical measures are stored in counters in the flow-entry.

If state processing is required, state process 328 is commenced. State processor 328 carries out any state operations specified for the state of the flow and updates the state to the next state according to a set of state instructions obtained form the state pattern and processes database 326.

The state processor 328 analyzes both new and existing flows in order to analyze all levels of the protocol stack, ultimately classifying the flows by application (level 7 in the ISO model). It does this by proceeding from state-to-state based on predefined state transition rules and state operations as specified in state processor instruction database 326. A state transition rule is a rule typically containing a test followed by the next-state to proceed to if the test result is true. An operation is an operation to be performed while the state processor is in a particular state—for example, in order to evaluate a quantity needed to apply the state transition rule. The state processor goes through each rule and each state process until the test is true, or there are no more tests to perform.

In general, the set of state operations may be none or more operations on a packet, and carrying out the operation or operations may leave one in a state that causes exiting the system prior to completing the identification, but possibly knowing more about what state and state processes are needed to execute next, *i.e.,* when a next packet of this flow is encountered. As an example, a state process (set of state operations) at a particular state may build a new signature for future recognition packets of the next state.

By maintaining the state of the flows and knowing that new flows may be set up using the information from previously encountered flows, the network traffic monitor 300 provides for (a) single-packet protocol recognition of flows, and (b) multiple-packet protocol recognition of flows. Monitor 300 can even recognize the application program from one or more disjointed sub-flows that occur in server announcement type flows. What may seem to prior art monitors to be some unassociated flow, may be recognized by the inventive monitor using the flow signature to be a sub-flow associated with a previously encountered sub-flow.

Thus, state processor 328 applies the first state operation to the packet for this particular flow-entry. A process 330 decides if more operations need to be performed for this state. If so, the analyzer continues looping between block 330 and 328 applying additional state operations to this particular packet until all those operations are completed—that is, there are no more operations for this packet in this state. A process 332 decides if there are further states to be analyzed for this type of flow according to the state of the flow and the protocol, in order to fully characterize the flow. If not, the conversational flow has now been fully characterized and a process 334 finalizes the classification of the conversational flow for the flow.

In the particular embodiment, the state processor 328 starts the state processing by using the last protocol recognized by the parser as an offset into a jump table (jump vector). The jump table finds the state processor instructions to use for that protocol in the state patterns and processes database 326. Most instructions test something in the unified flow key buffer, or the flow-entry in the database of known flows 324, if the entry exists. The state processor may have to test bits, do comparisons, add, or subtract to perform the test. For example, a common operation carried out by the state processor is searching for one or more patterns in the payload part of the UFKB.

Thus, in 332 in the classification, the analyzer decides whether the flow is at an end state. If not at an end state, the flow-entry is updated (or created if a new flow) for this flow-entry in process 322.

Furthermore, if the flow is known and if in 332 it is determined that there are further states to be processed using later packets, the flow-entry is updated in process 322.

The flow-entry also is updated after classification finalization so that any further packets belonging to this flow will be readily identified from their signature as belonging to this fully analyzed conversational flow.

After updating, database 324 therefore includes the set of all the conversational flows that have occurred.

Thus, the embodiment of present invention shown in FIG. 3 automatically maintains flow-entries, which in one aspect includes storing states. The monitor of

FIG. 3 also generates characteristic parts of packets—the signatures—that can be used to recognize flows. The flow-entries may be identified and accessed by their signatures. Once a packet is identified to be from a known flow, the state of the flow is known and this knowledge enables state transition analysis to be performed in real time for each different protocol and application. In a complex analysis, state transitions are traversed as more and more packets are examined. Future packets that are part of the same conversational flow have their state analysis continued from a previously achieved state. When enough packets related to an application of interest have been processed, a final recognition state is ultimately reached, *i.e.,* a set of states has been traversed by state analysis to completely characterize the conversational flow. The signature for that final state enables each new incoming packet of the same conversational flow to be individually recognized in real time.

In this manner, one of the great advantages of the present invention is realized. Once a particular set of state transitions has been traversed for the first time and ends in a final state, a short-cut recognition pattern—a signature—can be generated that will key on every new incoming packet that relates to the conversational flow. Checking a signature involves a simple operation, allowing high packet rates to be successfully monitored on the network.

In improved embodiments, several state analyzers are run in parallel so that a large number of protocols and applications may be checked for. Every known protocol and application will have at least one unique set of state transitions, and can therefore be uniquely identified by watching such transitions.

When each new conversational flow starts, signatures that recognize the flow are automatically generated on-the-fly, and as further packets in the conversational flow are encountered, signatures are updated and the states of the set of state transitions for any potential application are further traversed according to the state transition rules for the flow. The new states for the flow—those associated with a set of state transitions for one or more potential applications—are added to the records of previously encountered states for easy recognition and retrieval when a new packet in the flow is encountered.

Detailed operation

FIG. 4 diagrams an initialization system 400 that includes the compilation process. That is, part of the initialization generates the pattern structures and extraction operations database 308 and the state instruction database 328. Such initialization can occur off-line or from a central location.

The different protocols that can exist in different layers may be thought of as nodes of one or more trees of linked nodes. The packet type is the root of a tree (called level 0). Each protocol is either a parent node or a terminal node. A parent node links a protocol to other protocols (child protocols) that can be at higher layer levels. Thus a protocol may have zero or more children. Ethernet packets, for example, have several variants, each having a basic format that remains substantially the same. An Ethernet packet (the root or level 0 node) may be an Ethertype packet—also called an Ethernet Type/Version 2 and a DIX (DIGITAL-Intel-Xerox packet)—or an IEEE 803.2 packet. Continuing with the IEEE 802.3 packet, one of the children nodes may be the IP protocol, and one of the children of the IP protocol may be the TCP protocol.

FIG. 16 shows the header 1600 (base level 1) of a complete Ethernet frame (*i.e.,* packet) of information and includes information on the destination media access control address (Dst MAC 1602) and the source media access control address (Src MAC 1604). Also shown in FIG. 16 is some (but not all) of the information specified in the PDL files for extraction the signature.

FIG. 17A now shows the header information for the next level (level-2) for an Ethertype packet 1700. For an Ethertype packet 1700, the relevant information from the packet that indicates the next layer level is a two-byte type field 1702 containing the child recognition pattern for the next level. The remaining information 1704 is shown hatched because it not relevant for this level. The list 1712 shows the possible children for an Ethertype packet as indicated by what child recognition pattern is found offset 12. FIG. 17B shows the structure of the header of one of the possible next levels, that of the IP protocol. The possible children of the IP protocol are shown in table 1752.

The pattern, parse, and extraction database (pattern recognition database, or PRD) 308 generated by compilation process 310, in one embodiment, is in the form of a three dimensional structure that provides for rapidly searching packet headers for the next protocol. FIG. 18A shows such a 3-D representation 1800 (which may be considered as an indexed set of 2-D representations). A compressed form of the 3-D structure is preferred.

An alternate embodiment of the data structure used in database 308 is illustrated in FIG. 18B. Thus, like the 3-D structure of FIG. 18A, the data structure permits rapid searches to be performed by the pattern recognition process 304 by indexing locations in a memory rather than performing address link computations. In this alternate embodiment, the PRD 308 includes two parts, a single protocol table 1850 (PT) which has an entry for each protocol known for the monitor, and a series of Look Up Tables 1870 (LUT's) that are used to identify known protocols and their children. The protocol table includes the parameters needed by the pattern analysis and recognition process 304 (implemented by PRE 1006) to evaluate the header information in the packet that is associated with that protocol, and parameters needed by extraction process 306 (implemented by slicer 1007) to process the packet header. When there are children, the PT describes which bytes in the header to evaluate to determine the child protocol. In particular, each PT entry contains the header length, an offset to the child, a slicer command, and some flags.

The pattern matching is carried out by finding particular "child recognition codes" in the header fields, and using these codes to index one or more of the LUT's. Each LUT entry has a node code that can have one of four values, indicating the protocol that has been recognized, a code to indicate that the protocol has been partially recognized (more LUT lookups are needed), a code to indicate that this is a terminal node, and a null node to indicate a null entry. The next LUT to lookup is also returned from a LUT lookup.

Compilation process is described in FIG. 4. The source-code information in the form of protocol description files is shown as 402. In the particular embodiment, the high level decoding descriptions includes a set of protocol description files 336, one for each protocol, and a set of packet layer selections 338, which describes the particular layering (sets of trees of protocols) that the monitor is to be able to handle.

A compiler 403 compiles the descriptions. The set of packet parse-and-extract operations 406 is generated (404), and a set of packet state instructions and operations 407 is generated (405) in the form of instructions for the state processor that implements state processing process 328. Data files for each type of application and protocol to be recognized by the analyzer are downloaded from the pattern, parse, and extraction database 406 into the memory systems of the parser and extraction engines. (See the parsing process 500 description and FIG. 5; the extraction process 600 description and FIG. 6; and the parsing subsystem hardware description and FIG. 10). Data files for each type of application and protocol to be recognized by the analyzer are also downloaded from the state-processor instruction database 407 into the state processor. (see the state processor 1108 description and FIG. 11.).

Note that generating the packet parse and extraction operations builds and links the three dimensional structure (one embodiment) or the or all the lookup tables for the PRD.

Because of the large number of possible protocol trees and subtrees, the compiler process 400 includes optimization that compares the trees and subtrees to see which children share common parents. When implemented in the form of the LUT's, this process can generate a single LUT from a plurality of LUT's. The optimization process further includes a compaction process that reduces the space needed to store the data of the PRD.

As an example of compaction, consider the 3-D structure of FIG. 18A that can be thought of as a set of 2-D structures each representing a protocol. To enable saving space by using only one array per protocol which may have several parents, in one embodiment, the pattern analysis subprocess keeps a “current header” pointer. Each location (offset) index for each protocol 2-D array in the 3-D structure is a relative location starting with the start of header for the particular protocol. Furthermore, each of the two-dimensional arrays is sparse. The next step of the optimization, is checking all the 2-D arrays against all the other 2-D arrays to find out which ones can share memory. Many of these 2-D arrays are often sparsely populated in that they each have only a small number of valid entries. So, a process of "folding" is next used to combine two or more 2-D arrays together into one physical 2-D array without losing the identity of any of the original 2-D arrays (i.e., all the 2-D arrays continue to exist logically). Folding can occur between any 2-D arrays irrespective of their location in the tree as long as certain conditions are met. Multiple arrays may be combined into a single array as long as the individual entries do not conflict with each other. A fold number is then used to associate each element with its original array. A similar folding process is used for the set of LUTs 1850 in the alternate embodiment of FIG. 18B.

In 410, the analyzer has been initialized and is ready to perform recognition.

FIG. 5 shows a flowchart of how actual parser subsystem 301 functions. Starting at 501, the packet 302 is input to the packet buffer in step 502. Step 503 loads the next (initially the first) packet component from the packet 302. The packet components are extracted from each packet 302 one element at a time. A check is made (504) to determine if the load-packet-component operation 503 succeeded, indicating that there was more in the packet to process. If not, indicating all components have been loaded, the parser subsystem 301 builds the packet signature (512)—the next stage (FIG 6).

If a component is successfully loaded in 503, the node and processes are fetched (505) from the pattern, parse and extraction database 308 to provide a set of patterns and processes for that node to apply to the loaded packet component. The parser subsystem 301 checks (506) to determine if the fetch pattern node operation 505 completed successfully, indicating there was a pattern node that loaded in 505. If not, step 511 moves to the next packet component. If yes, then the node and pattern matching process are applied in 507 to the component extracted in 503. A pattern match obtained in 507 (as indicated by test 508) means the parser subsystem 301 has found a node in the parsing elements; the parser subsystem 301 proceeds to step 509 to extract the elements.

If applying the node process to the component does not produce a match (test 508), the parser subsystem 301 moves (510) to the next pattern node from the pattern database 308 and to step 505 to fetch the next node and process. Thus, there is an "applying patterns" loop between 508 and 505. Once the parser subsystem 301 completes all the patterns and has either matched or not, the parser subsystem 301 moves to the next packet component (511).

Once all the packet components have been the loaded and processed from the input packet 302, then the load packet will fail (indicated by test 504), and the parser subsystem 301 moves to build a packet signature which is described in FIG. 6

FIG. 6 is a flow chart for extracting the information from which to build the packet signature. The flow starts at 601, which is the exit point 513 of FIG. 5. At this point parser subsystem 301 has a completed packet component and a pattern node available in a buffer (602). Step 603 loads the packet component available from the pattern analysis process of FIG. 5. If the load completed (test 604), indicating that there was indeed another packet component, the parser subsystem 301 fetches in 605 the extraction and process elements received from the pattern node component in 602. If the fetch was successful (test 606), indicating that there are extraction elements to apply, the parser subsystem 301 in step 607 applies that extraction process to the packet component based on an extraction instruction received from that pattern node. This removes and saves an element from the packet component.

In step 608, the parser subsystem 301 checks if there is more to extract from this component, and if not, the parser subsystem 301 moves back to 603 to load the next packet component at hand and repeats the process. If the answer is yes, then the parser subsystem 301 moves to the next packet component ratchet. That new packet component is then loaded in step 603. As the parser subsystem 301 moved through the loop between 608 and 603, extra extraction processes are applied either to the same packet component if there is more to extract, or to a different packet component if there is no more to extract.

The extraction process thus builds the signature, extracting more and more components according to the information in the patterns and extraction database 308 for the particular packet. Once loading the next packet component operation 603 fails (test 604), all the components have been extracted. The built signature is loaded into the signature buffer (610) and the parser subsystem 301 proceeds to FIG. 7 to complete the signature generation process.

Referring now to FIG. 7, the process continues at 701. The signature buffer and the pattern node elements are available (702). The parser subsystem 301 loads the next pattern node element. If the load was successful (test 704) indicating there are more nodes, the parser subsystem 301 in 705 hashes the signature buffer element based on the hash elements that are found in the pattern node that is in the element database. In 706 the resulting signature and the hash are packed. In 707 the parser subsystem 301 moves on to the next packet component which is loaded in 703.

The 703 to 707 loop continues until there are no more patterns of elements left (test 704). Once all the patterns of elements have been hashed, processes 304, 306 and 312 of parser subsystem 301 are complete. Parser subsystem 301 has generated the signature used by the analyzer subsystem 303.

A parser record is loaded into the analyzer, in particular, into the UFKB in the form of a UFKB record which is similar to a parser record, but with one or more different fields.

FIG. 8 is a flow diagram describing the operation of the lookup/update engine (LUE) that implements lookup operation 314. The process starts at 801 from FIG. 7 with the parser record that includes a signature, the hash and at least parts of the payload. In 802 those elements are shown in the form of a UFKB-entry in the buffer. The LUE, the lookup engine 314 computes a "record bin number" from the hash for a flow-entry. A bin herein may have one or more "buckets" each containing a flow-entry. The preferred embodiment has four buckets per bin.

Since preferred hardware embodiment includes the cache, all data accesses to records in the flowchart of FIG. 8 are stated as being to or from the cache.

Thus, in 804, the system looks up the cache for a bucket from that bin using the hash. If the cache successfully returns with a bucket from the bin number, indicating there are more buckets in the bin, the lookup/update engine compares (807) the current signature (the UFKB-entry's signature) from that in the bucket (i.e., the flow-entry signature). If the signatures match (test 808), that record (in the cache) is marked in step 810 as "in process" and a timestamp added. Step 811 indicates to the UFKB that the UFKB-entry in 802 has a status of "found." The "found" indication allows the state processing 328 to begin processing this UFKB element. The preferred hardware embodiment includes one or more state processors, and these can operate in parallel with the lookup/update engine.

In the preferred embodiment, a set of statistical operations is performed by a calculator for every packet analyzed. The statistical operations may include one or more of counting the packets associated with the flow; determining statistics related to the size of packets of the flow; compiling statistics on differences between packets in each direction, for example using timestamps; and determining statistical relationships of timestamps of packets in the same direction. The statistical measures are kept in the flow-entries. Other statistical measures also may be compiled. These statistics may be used singly or in combination by a statistical processor component to analyze many different aspects of the flow. This may include determining network usage metrics from the statistical measures, for example to ascertain the network's ability to transfer information for this application. Such analysis provides for measuring the quality of service of a conversation, measuring how well an application is performing in the network, measuring network resources consumed by an application, and so forth.

To provide for such analyses, the lookup/update engine updates one or more counters that are part of the flow-entry (in the cache) in step 812. The process exits at 813. In our embodiment, the counters include the total packets of the flow, the time, and a differential time from the last timestamp to the present timestamp.

It may be that the bucket of the bin did not lead to a signature match (test 808). In such a case, the analyzer in 809 moves to the next bucket for this bin. Step 804 again looks up the cache for another bucket from that bin. The lookup/update engine thus continues lookup up buckets of the bin until there is either a match in 808 or operation 804 is not successful (test 805), indicating that there are no more buckets in the bin and no match was found.

If no match was found, the packet belongs to a new (not previously encountered) flow. In 806 the system indicates that the record in the unified flow key buffer for this packet is new, and in 812, any statistical updating operations are performed for this packet by updating the flow-entry in the cache. The update operation exits at 813. A flow insertion/deletion engine (FIDE) creates a new record for this flow (again via the cache).

Thus, the update/lookup engine ends with a UFKB-entry for the packet with a "new" status or a "found" status.

Note that the above system uses a hash to which more than one flow-entry can match. A longer hash may be used that corresponds to a single flow-entry. In such an embodiment, the flow chart of FIG. 8 is simplified as would be clear to those in the art.

The hardware system

Each of the individual hardware elements through which the data flows in the system are now described with reference to FIGS. 10 and 11. Note that while we are describing a particular hardware implementation of the invention embodiment of FIG. 3, it would be clear to one skilled in the art that the flow of FIG. 3 may alternatively be implemented in software running on one or more general-purpose processors, or only partly implemented in hardware. An implementation of the invention that can operate in software is shown in FIG. 14. The hardware embodiment (FIGS. 10 and 11) can operate at over a million packets per second, while the software system of FIG. 14 may be suitable for slower networks. To one skilled in the art it would be clear that more and more of the system may be implemented in software as processors become faster.

FIG. 10 is a description of the parsing subsystem (301, shown here as subsystem 1000) as implemented in hardware. Memory 1001 is the pattern recognition database memory, in which the patterns that are going to be analyzed are stored. Memory 1002 is the extraction-operation database memory, in which the extraction instructions are stored. Both 1001 and 1002 correspond to internal data structure 308 of FIG. 3. Typically, the system is initialized from a microprocessor (not shown) at which time these memories are loaded through a host interface multiplexor and control register 1005 via the internal buses 1003 and 1004. Note that the contents of 1001 and 1002 are preferably obtained by compiling process 310 of FIG. 3.

A packet enters the parsing system via 1012 into a parser input buffer memory 1008 using control signals 1021 and 1023, which control an input buffer interface controller 1022. The buffer 1008 and interface control 1022 connect to a packet acquisition device (not shown). The buffer acquisition device generates a packet start signal 1021 and the interface control 1022 generates a next packet (i.e., ready to receive data) signal 1023 to control the data flow into parser input buffer memory 1008. Once a packet starts loading into the buffer memory 1008, pattern recognition engine (PRE) 1006 carries out the operations on the input buffer memory described in block 304 of FIG. 3. That is, protocol types and associated headers for each protocol layer that exist in the packet are determined.

The PRE searches database 1001 and the packet in buffer 1008 in order to recognize the protocols the packet contains. In one implementation, the database 1001 includes a series of linked lookup tables. Each lookup table uses eight bits of addressing. The first lookup table is always at address zero. The Pattern Recognition Engine uses a base packet offset from a control register to start the comparison. It loads this value into a current offset pointer (COP). It then reads the byte at base packet offset from the parser input buffer and uses it as an address into the first lookup table.

Each lookup table returns a word that links to another lookup table or it returns a terminal flag. If the lookup produces a recognition event the database also returns a command for the slicer. Finally it returns the value to add to the COP.

The PRE 1006 includes of a comparison engine. The comparison engine has a first stage that checks the protocol type field to determine if it is an 802.3 packet and the field should be treated as a length. If it is not a length, the protocol is checked in a second stage. The first stage is the only protocol level that is not programmable. The second stage has two full sixteen bit content addressable memories (CAMs) defined for future protocol additions.

Thus, whenever the PRE recognizes a pattern, it also generates a command for the extraction engine (also called a "slicer") 1007. The recognized patterns and the commands are sent to the extraction engine 1007 that extracts information from the packet to build the parser record. Thus, the operations of the extraction engine are those carried out in blocks 306 and 312 of FIG. 3. The commands are sent from PRE 1006 to slicer 1007 in the form of extraction instruction pointers which tell the extraction engine 1007 where to a find the instructions in the extraction operations database memory (i.e., slicer instruction database) 1002.

Thus, when the PRE 1006 recognizes a protocol it outputs both the protocol identifier and a process code to the extractor. The protocol identifier is added to the flow signature and the process code is used to fetch the first instruction from the instruction database 1002. Instructions include an operation code and usually source and destination offsets as well as a length. The offsets and length are in bytes. A typical operation is the MOVE instruction. This instruction tells the slicer 1007 to copy n bytes of data unmodified from the input buffer 1008 to the output buffer 1010. The extractor contains a byte-wise barrel shifter so that the bytes moved can be packed into the flow signature. The extractor contains another instruction called HASH. This instruction tells the extractor to copy from the input buffer 1008 to the HASH generator.

Thus these instructions are for extracting selected element(s) of the packet in the input buffer memory and transferring the data to a parser output buffer memory 1010. Some instructions also generate a hash.

The extraction engine 1007 and the PRE operate as a pipeline. That is, extraction engine 1007 performs extraction operations on data in input buffer 1008 already processed by PRE 1006 while more (i.e., later arriving) packet information is being simultaneously parsed by PRE 1006. This provides high processing speed sufficient to accommodate the high arrival rate speed of packets.

Once all the selected parts of the packet used to form the signature are extracted, the hash is loaded into parser output buffer memory 1010. Any additional payload from the packet that is required for further analysis is also included. The parser output memory 1010 is interfaced with the analyzer subsystem by analyzer interface control 1011. Once all the information of a packet is in the parser output buffer memory 1010, a data ready signal 1025 is asserted by analyzer interface control. The data from the parser subsystem 1000 is moved to the analyzer subsystem via 1013 when an analyzer ready signal 1027 is asserted.

FIG. 11 shows the hardware components and dataflow for the analyzer subsystem that performs the functions of the analyzer subsystem 303 of FIG. 3. The analyzer is initialized prior to operation, and initialization includes loading the state processing information generated by the compilation process 310 into a database memory for the state processing, called state processor instruction database (SPID) memory 1109.

The analyzer subsystem 1100 includes a host bus interface 1122 using an analyzer host interface controller 1118, which in turn has access to a cache system 1115. The cache system has bi-directional access to and from the state processor of the system 1108. State processor 1108 is responsible for initializing the state processor instruction database memory 1109 from information given over the host bus interface 1122.

With the SPID 1109 loaded, the analyzer subsystem 1100 receives parser records comprising packet signatures and payloads that come from the parser into the unified flow key buffer (UFKB) 1103. UFKB is comprised of memory set up to maintain UFKB records. A UFKB record is essentially a parser record; the UFKB holds records of packets that are to be processed or that are in process. Furthermore, the UFKB provides for one or more fields to act as modifiable status flags to allow different processes to run concurrently.

Three processing engines run concurrently and access records in the UFKB 1103: the lookup/update engine (LUE) 1107, the state processor (SP) 1108, and the flow insertion and deletion engine (FIDE) 1110. Each of these is implemented by one or more finite state machines (FSM's). There is bi-directional access between each of the finite state machines and the unified flow key buffer 1103. The UFKB record includes a field that stores the packet sequence number, and another that is filled with state information in the form of a program counter for the state processor 1108 that implements state processing 328. The status flags of the UFKB for any entry includes that the LUE is done and that the LUE is transferring processing of the entry to the state processor. The LUE done indicator is also used to indicate what the next entry is for the LUE. There also is provided a flag to indicate that the state processor is done with the current flow and to indicate what the next entry is for the state processor. There also is provided a flag to indicate the state processor is transferring processing of the UFKB-entry to the flow insertion and deletion engine.

A new UFKB record is first processed by the LUE 1107. A record that has been processed by the LUE 1107 may be processed by the state processor 1108, and a UFKB record data may be processed by the flow insertion/deletion engine 1110 after being processed by the state processor 1108 or only by the LUE. Whether or not a particular engine has been applied to any unified flow key buffer entry is determined by status fields set by the engines upon completion. In one embodiment, a status flag in the UFKB-entry indicates whether an entry is new or found. In other embodiments, the LUE issues a flag to pass the entry to the state processor for processing, and the required operations for a new record are included in the SP instructions.

Note that each UFKB-entry may not need to be processed by all three engines. Furthermore, some UFKB entries may need to be processed more than once by a particular engine.

Each of these three engines also has bi-directional access to a cache subsystem 1115 that includes a caching engine. Cache 1115 is designed to have information flowing in and out of it from five different points within the system: the three engines, external memory via a unified memory controller (UMC) 1119 and a memory interface 1123, and a microprocessor via analyzer host interface and control unit (ACIC) 1118 and host interface bus (HIB) 1122. The analyzer microprocessor (or dedicated logic processor) can thus directly insert or modify data in the cache.

The cache subsystem 1115 is an associative cache that includes a set of content addressable memory cells (CAMs) each including an address portion and a pointer portion pointing to the cache memory (e.g., RAM) containing the cached flow-entries. The CAMs are arranged as a stack ordered from a top CAM to a bottom CAM. The bottom CAM's pointer points to the least recently used (LRU) cache memory entry. Whenever there is a cache miss, the contents of cache memory pointed to by the bottom CAM are replaced by the flow-entry from the flow-entry database 324. This now becomes the most recently used entry, so the contents of the bottom CAM are moved to the top CAM and all CAM contents are shifted down. Thus, the cache is an associative cache with a true LRU replacement policy.

The LUE 1107 first processes a UFKB-entry, and basically performs the operation of blocks 314 and 316 in FIG. 3. A signal is provided to the LUE to indicate that a "new" UFKB-entry is available. The LUE uses the hash in the UFKB-entry to read a matching bin of up to four buckets from the cache. The cache system attempts to obtain the matching bin. If a matching bin is not in the cache, the cache 1115 makes the request to the UMC 1119 to bring in a matching bin from the external memory.

When a flow-entry is found using the hash, the LUE 1107 looks at each bucket and compares it using the signature to the signature of the UFKB-entry until there is a match or there are no more buckets.

If there is no match, or if the cache failed to provide a bin of flow-entries from the cache, a time stamp in set in the flow key of the UFKB record, a protocol identification and state determination is made using a table that was loaded by compilation process 310 during initialization, the status for the record is set to indicate the LUE has processed the record, and an indication is made that the UFKB-entry is ready to start state processing. The identification and state determination generates a protocol identifier which in the preferred embodiment is a "jump vector" for the state processor which is kept by the UFKB for this UFKB-entry and used by the state processor to start state processing for the particular protocol. For example, the jump vector jumps to the subroutine for processing the state.

If there was a match, indicating that the packet of the UFKB-entry is for a previously encountered flow, then a calculator component enters one or more statistical measures stored in the flow-entry, including the timestamp. In addition, a time difference from the last stored timestamp may be stored, and a packet count may be updated. The state of the flow is obtained from the flow-entry is examined by looking at the protocol identifier stored in the flow-entry of database 324. If that value indicates that no more classification is required, then the status for the record is set to indicate the LUE has processed the record. In the preferred embodiment, the protocol identifier is a jump vector for the state processor to a subroutine to state processing the protocol, and no more classification is indicated in the preferred embodiment by the jump vector being zero. If the protocol identifier indicates more processing, then an indication is made that the UFKB-entry is ready to start state processing and the status for the record is set to indicate the LUE has processed the record.

The state processor 1108 processes information in the cache system according to a UFKB-entry after the LUE has completed. State processor 1108 includes a state processor program counter SPPC that generates the address in the state processor instruction database 1109 loaded by compiler process 310 during initialization. It contains an Instruction Pointer (SPIP) which generates the SPID address. The instruction pointer can be incremented or loaded from a Jump Vector Multiplexor which facilitates conditional branching. The SPIP can be loaded from one of three sources: (1) A protocol identifier from the UFKB, (2) an immediate jump vector form the currently decoded instruction, or (3) a value provided by the arithmetic logic unit (SPALU) included in the state processor.

Thus, after a Flow Key is placed in the UFKB by the LUE with a known protocol identifier, the Program Counter is initialized with the last protocol recognized by the Parser. This first instruction is a jump to the subroutine which analyzes the protocol that was decoded.

The State Processor ALU (SPALU) contains all the Arithmetic, Logical and String Compare functions necessary to implement the State Processor instructions. The main blocks of the SPALU are: The A and B Registers, the Instruction Decode & State Machines, the String Reference Memory the Search Engine, an Output Data Register and an Output Control Register

The Search Engine in turn contains the Target Search Register set, the Reference Search Register set, and a Compare block which compares two operands by exclusive-or-ing them together.

Thus, after the UFKB sets the program counter, a sequence of one or more state operations are be executed in state processor 1108 to further analyze the packet that is in the flow key buffer entry for this particular packet.

FIG. 13 describes the operation of the state processor 1108. The state processor is entered at 1301 with a unified flow key buffer entry to be processed. The UFKB-entry is new or corresponding to a found flow-entry. This UFKB-entry is retrieved from unified flow key buffer 1103 in 1301. In 1303, the protocol identifier for the UFKB-entry is used to set the state processor's instruction counter. The state processor 1108 starts the process by using the last protocol recognized by the parser subsystem 301 as an offset into a jump table. The jump table takes us to the instructions to use for that protocol. Most instructions test something in the unified flow key buffer or the flow-entry if it exists. The state processor 1108 may have to test bits, do comparisons, add or subtract to perform the test.

The first state processor instruction is fetched in 1304 from the state processor instruction database memory 1109. The state processor performs the one or more fetched operations (1304). In our implementation, each single state processor instruction is very primitive (e.g., a move, a compare, etc.), so that many such instructions need to be performed on each unified flow key buffer entry. One aspect of the state processor is its ability to search for one or more (up to four) reference strings in the payload part of the

UFKB entry. This is implemented by a search engine component of the state processor responsive to special searching instructions.

In 1307, a check is made to determine if there are any more instructions to be performed for the packet. If yes, then in 1308 the system sets the state processor instruction pointer (SPIP) to obtain the next instruction. The SPIP may be set by an immediate jump vector in the currently decoded instruction, or by a value provided by the SPALU during processing.

The next instruction to be performed is now fetched (1304) for execution. This state processing loop between 1304 and 1307 continues until there are no more instructions to be performed.

At this stage, a check is made in 1309 if the processing on this particular packet has resulted in a final state. That is, is the analyzer is done processing not only for this particular packet, but for the whole flow to which the packet belongs, and the flow is fully determined. If indeed there are no more states to process for this flow, then in 1311 the processor finalizes the processing. Some final states may need to put a state in place that tells the system to remove a flow—for example, if a connection disappears from a lower level connection identifier. In that case, in 1311, a flow removal state is set and saved in the flow-entry. The flow removal state may be a NOP (no-op) instruction which means there are no removal instructions.

Once the appropriate flow removal instruction as specified for this flow (a NOP or otherwise) is set and saved, the process is exited at 1313. The state processor 1108 can now obtain another unified flow key buffer entry to process.

If at 1309 it is determined that processing for this flow is not completed, then in 1310 the system saves the state processor instruction pointer in the current flow-entry in the current flow-entry. That will be the next operation that will be performed the next time the LRE 1107 finds packet in the UFKB that matches this flow. The processor now exits processing this particular unified flow key buffer entry at 1313.

Note that state processing updates information in the unified flow key buffer 1103 and the flow-entry in the cache. Once the state processor is done, a flag is set in the UFKB for the entry that the state processor is done. Furthermore, If the flow needs to be inserted or deleted from the database of flows, control is then passed on to the flow insertion/deletion engine 1110 for that flow signature and packet entry. This is done by the state processor setting another flag in the UFKB for this UFKB-entry indicating that the state processor is passing processing of this entry to the flow insertion and deletion engine.

The flow insertion and deletion engine 1110 is responsible for maintaining the flow-entry database. In particular, for creating new flows in the flow database, and deleting flows from the database so that they can be reused.

The process of flow insertion is now described with the aid of FIG. 12. Flows are grouped into bins of buckets by the hash value. The engine processes a UFKB-entry that may be new or that the state processor otherwise has indicated needs to be created. FIG. 12 shows the case of a new entry being created. A conversation record bin (preferably containing 4 buckets for four records) is obtained in 1203. This is a bin that matches the hash of the UFKB, so this bin may already have been sought for the UFKB-entry by the LUE. In 1204 the FIDE 1110 requests that the record bin/bucket be maintained in the cache system 1115. If in 1205 the cache system 1115 indicates that the bin/bucket is empty, step 1207 inserts the flow signature (with the hash) into the bucket and the bucket is marked "used" in the cache engine of cache 1115 using a timestamp that is maintained throughout the process. In 1209, the FIDE 1110 compares the bin and bucket record flow signature to the packet to verify that all the elements are in place to complete the record. In 1211 the system marks the record bin and bucket as "in process" and as "new" in the cache system (and hence in the external memory). In 1212, the initial statistical measures for the flow-record are set in the cache system. This in the preferred embodiment clears the set of counters used to maintain statistics, and may perform other procedures for statistical operations requires by the analyzer for the first packet seen for a particular flow.

Back in step 1205, if the bucket is not empty, the FIDE 1110 requests the next bucket for this particular bin in the cache system. If this succeeds, the processes of 1207, 1209, 1211 and 1212 are repeated for this next bucket. If at 1208, there is no valid bucket, the unified flow key buffer entry for the packet is set as "drop," indicating that the system cannot process the particular packet because there are no buckets left in the system. The process exits at 1213. The FIDE 1110 indicates to the UFKB that the flow insertion and deletion operations are completed for this UFKB-entry. This also lets the UFKB provide the FIDE with the next UFKB record.

Once a set of operations is performed on a unified flow key buffer entry by all of the engines required to access and manage a particular packet and its flow signature, the unified flow key buffer entry is marked as "completed." That element will then be used by the parser interface for the next packet and flow signature coming in from the parsing and extracting system.

All flow-entries are maintained in the external memory and some are maintained in the cache 1115. The cache system 1115 is intelligent enough to access the flow database and to understand the data structures that exists on the other side of memory interface 1123. The lookup/update engine 1107 is able to request that the cache system pull a particular flow or "buckets" of flows from the unified memory controller 1119 into the cache system for further processing. The state processor 1108 can operate on information found in the cache system once it is looked up by means of the lookup/update engine request, and the flow insertion/deletion engine 1110 can create new entries in the cache system if required based on information in the unified flow key buffer 1103. The cache retrieves information as required from the memory through the memory interface 1123 and the unified memory controller 1119, and updates information as required in the memory through the memory controller 1119.

There are several interfaces to components of the system external to the module of FIG. 11 for the particular hardware implementation. These include host bus interface 1122,which is designed as a generic interface that can operate with any kind of external processing system such as a microprocessor or a multiplexor (MUX) system. Consequently, one can connect the overall traffic classification system of FIGS. 11 and 12 into some other processing system to manage the classification system and to extract data gathered by the system.

The memory interface 1123 is designed to interface to any of a variety of memory systems that one may want to use to store the flow-entries. One can use different types of memory systems like regular dynamic random access memory (DRAM), synchronous DRAM, synchronous graphic memory (SGRAM), static random access memory

(SRAM), and so forth.

FIG. 10 also includes some "generic" interfaces. There is a packet input interface 1012—a general interface that works in tandem with the signals of the input buffer interface control 1022. These are designed so that they can be used with any kind of generic systems that can then feed packet information into the parser. Another generic interface is the interface of pipes 1031 and 1033 respectively out of and into host interface multiplexor and control registers 1005. This enables the parsing system to be managed by an external system, for example a microprocessor or another kind of external logic, and enables the external system to program and otherwise control the parser.

The preferred embodiment of this aspect of the invention is described in a hardware description language (HDL) such as VHDL or Verilog. It is designed and created in an HDL so that it may be used as a single chip system or, for instance, integrated into another general-purpose system that is being designed for purposes related to creating and analyzing traffic within a network. Verilog or other HDL implementation is only one method of describing the hardware.

In accordance with one hardware implementation, the elements shown in FIGS. 10 and 11 are implemented in a set of six field programmable logic arrays (FPGA's). The boundaries of these FPGA's are as follows. The parsing subsystem of FIG. 10 is implemented as two FPGAS; one FPGA, and includes blocks 1006, 1008 and 1012, parts of 1005, and memory 1001. The second FPGA includes 1002, 1007, 1013, 1011 parts of 1005. Referring to FIG. 11, the unified look-up buffer 1103 is implemented as a single FPGA. State processor 1108 and part of state processor instruction database memory 1109 is another FPGA. Portions of the state processor instruction database memory 1109 are maintained in external SRAM's. The lookup/update engine 1107 and the flow insertion/deletion engine 1110 are in another FPGA. The sixth FPGA includes the cache system 1115, the unified memory control 1119, and the analyzer host interface and control 1118.

Note that one can implement the system as one or more VSLI devices, rather than as a set of application specific integrated circuits (ASIC's) such as FPGA's. It is anticipated that in the future device densities will continue to increase, so that the complete system may eventually form a sub-unit (a "core") of a larger single chip unit.

Operation of the Invention

Fig. 15 shows how an embodiment of the network monitor 300 might be used to analyze traffic in a network 102. Packet acquisition device 1502 acquires all the packets from a connection point 121 on network 102 so that all packets passing point 121 in either direction are supplied to monitor 300. Monitor 300 comprises the parser sub-system 301, which determines flow signatures, and analyzer sub-system 303 that analyzes the flow signature of each packet. A memory 324 is used to store the database of flows that are determined and updated by monitor 300. A host computer 1504, which might be any processor, for example, a general-purpose computer, is used to analyze the flows in memory 324. As is conventional, host computer 1504 includes a memory, say RAM, shown as host memory 1506. In addition, the host might contain a disk. In one application, the system can operate as an RMON probe, in which case the host computer is coupled to a network interface card 1510 that is connected to the network 102.

The preferred embodiment of the invention is supported by an optional Simple Network Management Protocol (SNMP) implementation. Fig. 15 describes how one would, for example, implement an RMON probe, where a network interface card is used to send RMON information to the network. Commercial SNMP implementations also are available, and using such an implementation can simplify the process of porting the preferred embodiment of the invention to any platform.

In addition, MIB Compilers are available. An MIB Compiler is a tool that greatly simplifies the creation and maintenance of proprietary MIB extensions.

Examples of Packet Elucidation

Monitor 300, and in particular, analyzer 303 is capable of carrying out state analysis for packet exchanges that are commonly referred to as "server announcement" type exchanges. Server announcement is a process used to ease communications between a server with multiple applications that can all be simultaneously accessed from multiple clients. Many applications use a server announcement process as a means of multiplexing a single port or socket into many applications and services. With this type of exchange, messages are sent on the network, in either a broadcast or multicast approach, to announce a server and application, and all stations in the network may receive and decode these messages. The messages enable the stations to derive the appropriate connection point for communicating that particular application with the particular server. Using the server announcement method, a particular application communicates using a service channel, in the form of a TCP or UDP socket or port as in the IP protocol suite, or using a SAP as in the Novell IPX protocol suite.

The analyzer 303 is also capable of carrying out “in-stream analysis” of packet exchanges. The “in-stream analysis” method is used either as a primary or secondary recognition process. As a primary process, in-stream analysis assists in extracting detailed information which will be used to further recognize both the specific application and application component. A good example of in-stream analysis is any Web-based application. For example, the commonly used PointCast Web information application can be recognized using this process; during the initial connection between a PointCast server and client, specific key tokens exist in the data exchange that will result in a signature being generated to recognize PointCast.

The in-stream analysis process may also be combined with the server announcement process. In many cases in-stream analysis will augment other recognition processes. An example of combining in-stream analysis with server announcement can be found in business applications such as SAP and BAAN.

“Session tracking” also is known as one of the primary processes for tracking applications in client/server packet exchanges. The process of tracking sessions requires an initial connection to a predefined socket or port number. This method of communication is used in a variety of transport layer protocols. It is most commonly seen in the TCP and UDP transport protocols of the IP protocol.

During the session tracking, a client makes a request to a server using a specific port or socket number. This initial request will cause the server to create a TCP or UDP port to exchange the remainder of the data between the client and the server. The server then replies to the request of the client using this newly created port. The original port used by the client to connect to the server will never be used again during this data exchange.

One example of session tracking is TFTP (Trivial File Transfer Protocol), a version of the TCP/IP FTP protocol that has no directory or password capability. During the client/server exchange process of TFTP, a specific port (port number 69) is always used to initiate the packet exchange. Thus, when the client begins the process of communicating, a request is made to UDP port 69. Once the server receives this request, a new port number is created on the server. The server then replies to the client using the new port. In this example, it is clear that in order to recognize TFTP; network monitor 300 analyzes the initial request from the client and generates a signature for it. Monitor 300 uses that signature to recognize the reply. Monitor 300 also analyzes the reply from the server with the key port information, and uses this to create a signature for monitoring the remaining packets of this data exchange.

Network monitor 300 can also understand the current state of particular connections in the network. Connection-oriented exchanges often benefit from state tracking to correctly identify the application. An example is the common TCP transport protocol that provides a reliable means of sending information between a client and a server. When a data exchange is initiated, a TCP request for synchronization message is sent. This message contains a specific sequence number that is used to track an acknowledgement from the server. Once the server has acknowledged the synchronization request, data may be exchanged between the client and the server. When communication is no longer required, the client sends a finish or complete message to the server, and the server acknowledges this finish request with a reply containing the sequence numbers from the request. The states of such a connection-oriented exchange relate to the various types of connection and maintenance messages.

Server Announcement Example

The individual methods of server announcement protocols vary. However, the basic underlying process remains similar. A typical server announcement message is sent to one or more clients in a network. This type of announcement message has specific content, which, in another aspect of the invention, is salvaged and maintained in the database of flow-entries in the system. Because the announcement is sent to one or more stations, the client involved in a future packet exchange with the server will make an assumption that the information announced is known, and an aspect of the inventive monitor is that it too can make the same assumption.

Sun-RPC is the implementation by Sun Microsystems, Inc. (Palo Alto, California) of the Remote Procedure Call (RPC), a programming interface that allows one program to use the services of another on a remote machine. A Sun-RPC example is now used to explain how monitor 300 can capture server announcements.

A remote program or client that wishes to use a server or procedure must establish a connection, for which the RPC protocol can be used.

Each server running the Sun-RPC protocol must maintain a process and database called the port Mapper. The port Mapper creates a direct association between a Sun-RPC program or application and a TCP or UDP socket or port (for TCP or UDP implementations). An application or program number is a 32-bit unique identifier assigned by ICANN (the Internet Corporation for Assigned Names and Numbers, www.icann.org), which manages the huge number of parameters associated with Internet protocols (port numbers, router protocols, multicast addresses, *etc.*) Each port Mapper on a Sun-RPC server can present the mappings between a unique program number and a specific transport socket through the use of specific request or a directed announcement. According to ICANN, port number 111 is associated with Sun RPC.

As an example, consider a client (*e.g.*, CLIENT 3 shown as 106 in FIG. 1) making a specific request to the server (*e.g.*, SERVER 2 of FIG. 1, shown as 110) on a predefined UDP or TCP socket. Once the port Mapper process on the sun RPC server receives the request, the specific mapping is returned in a directed reply to the client.

1. A client (CLIENT 3, 106 in FIG. 1) sends a TCP packet to SERVER 2 (110 in FIG. 1) on port 111, with an RPC Bind Lookup Request (rpcBindLookup). TCP or UDP port 111 is always associated Sun RPC. This request specifies the program (as a program identifier), version, and might specify the protocol (UDP or TCP).

2. The server SERVER 2 (110 in FIG. 1) extracts the program identifier and version identifier from the request. The server also uses the fact that this packet came in using the TCP transport and that no protocol was specified, and thus will use the TCP protocol for its reply.

3. The server 110 sends a TCP packet to port number 111, with an RPC Bind Lookup Reply. The reply contains the specific port number (*e.g., port* number 'port') on which future transactions will be accepted for the specific RPC program identifier (*e.g.,* Program 'program') and the protocol (UDP or TCP) for use.

It is desired that from now on every time that port number 'port' is used, the packet is associated with the application program 'program' until the number 'port' no longer is to be associated with the program 'program'. Network monitor 300 by creating a flow-entry and a signature includes a mechanism for remembering the exchange so that future packets that use the port number 'port' will be associated by the network monitor with the application program 'program'.

In addition to the Sun RPC Bind Lookup request and reply, there are other ways that a particular program—say 'program'—might be associated with a particular port number, for example number 'port'. One is by a broadcast announcement of a particular association between an application service and a port number, called a Sun RPC portMapper Announcement. Another, is when some server—say the same SERVER 2—replies to some client—say CLIENT 1—requesting some portMapper assignment with a RPC portMapper Reply. Some other client—say CLIENT 2—might inadvertently see this request, and thus know that for this particular server, SERVER 2, port number 'port' is associated with the application service 'program'. It is desirable for the network monitor 300 to be able to associate any packets to SERVER 2 using port number 'port' with the application program 'program'.

FIG. 9 represents a dataflow 900 of some operations in the monitor 300 of FIG. 3 for Sun Remote Procedure Call. Suppose a client 106 (*e.g.,* CLIENT 3 in FIG. 1) is communicating via its interface to the network 118 to a server 110 (*e.g.,* SERVER 2 in FIG. 1) via the server's interface to the network 116. Further assume that Remote Procedure Call is used to communicate with the server 110. One path in the data flow 900 starts with a step 910 that a Remote Procedure Call bind lookup request is issued by client 106 and ends with the server state creation step 904. Such RPC bind lookup request includes values for the 'program,' 'version,' and 'protocol' to use, *e.g.,* TCP or

UDP. The process for Sun RPC analysis in the network monitor 300 includes the following aspects. :

- Process 909: Extract the 'program,' 'version,' and 'protocol' (UDP or TCP). Extract the TCP or UDP port (process 909) which is 111 indicating Sun RPC.
- Process 908: Decode the Sun RPC packet. Check RPC type field for ID. If value is portMapper, save paired socket (*i.e.,* dest for destination address, src for source address). Decode ports and mapping, save ports with socket/addr key. There may be more than one pairing per mapper packet. Form a signature (e.g., a key). A flow-entry is created in database 324. The saving of the request is now complete.

At some later time, the server (process 907) issues a RPC bind lookup reply. The packet monitor 300 will extract a signature from the packet and recognize it from the previously stored flow. The monitor will get the protocol port number (906) and lookup the request (905). A new signature (i.e., a key) will be created and the creation of the server state (904) will be stored as an entry identified by the new signature in the flow-entry database. That signature now may be used to identify packets associated with the server.

The server state creation step 904 can be reached not only from a Bind Lookup Request/Reply pair, but also from a RPC Reply portMapper packet shown as 901 or an RPC Announcement portMapper shown as 902. The Remote Procedure Call protocol can announce that it is able to provide a particular application service. Embodiments of the present invention preferably can analyze when an exchange occurs between a client and a server, and also can track those stations that have received the announcement of a service in the network.

The RPC Announcement portMapper announcement 902 is a broadcast. Such causes various clients to execute a similar set of operations, for example, saving the information obtained from the announcement. The RPC Reply portMapper step 901 could be in reply to a portMapper request, and is also broadcast. It includes all the service parameters.

Thus monitor 300 creates and saves all such states for later classification of flows that relate to the particular service 'program'.

FIG. 2 shows how the monitor 300 in the example of Sun RPC builds a signature and flow states. A plurality of packets 206-209 are exchanged, *e.g.*, in an exemplary Sun Microsystems Remote Procedure Call protocol. A method embodiment of the present invention might generate a pair of flow signatures, “signature-1” 210 and “signature-2” 212, from information found in the packets 206 and 207 which, in the example, correspond to a Sun RPC Bind Lookup request and reply, respectively.

Consider first the Sun RPC Bind Lookup request. Suppose packet 206 corresponds to such a request sent from CLIENT 3 to SERVER 2. This packet contains important information that is used in building a signature according to an aspect of the invention. A source and destination network address occupy the first two fields of each packet, and according to the patterns in pattern database 308, the flow signature (shown as KEY1 230 in FIG. 2) will also contain these two fields, so the parser subsystem 301 will include these two fields in signature KEY 1 (230). Note that in FIG. 2, if an address identifies the client 106 (shown also as 202), the label used in the drawing is “C_1”. If such address identifies the server 110 (shown also as server 204), the label used in the drawing is “S_1”. The first two fields 214 and 215 in packet 206 are “S_1” and C_1” because packet 206 is provided from the server 110 and is destined for the client 106. Suppose for this example, “S_1” is an address numerically less than address “C_1”. A third field “p^1” 216 identifies the particular protocol being used, *e.g.*, TCP, UDP, etc.

In packet 206, a fourth field 217 and a fifth field 218 are used to communicate port numbers that are used. The conversation direction determines where the port number field is. The diagonal pattern in field 217 is used to identify a source-port pattern, and the hash pattern in field 218 is used to identify the destination-port pattern. The order indicates the client-server message direction. A sixth field denoted “i^1” 219 is an element that is being requested by the client from the server. A seventh field denoted “s_1a” 220 is the service requested by the client from server 110. The following eighth field “QA” 221 (for question mark) indicates that the client 106 wants to know what to use to access application “s_1a”. A tenth field “QP” 223 is used to indicate that the client wants the server to indicate what protocol to use for the particular application.

Packet 206 initiates the sequence of packet exchanges, *e.g.*, a RPC Bind Lookup Request to SERVER 2. It follows a well-defined format, as do all the packets, and is transmitted to the server 110 on a well-known service connection identifier (port 111 indicating Sun RPC).

Packet 207 is the first sent in reply to the client 106 from the server. It is the RPC Bind Lookup Reply as a result of the request packet 206.

Packet 207 includes ten fields 224–233. The destination and source addresses are carried in fields 224 and 225, *e.g.*, indicated "C_1" and "S_1", respectively. Notice the order is now reversed, since the client-server message direction is from the server 110 to the client 106. The protocol "p^1" is used as indicated in field 226. The request "i^1" is in field 229. Values have been filled in for the application port number, *e.g.*, in field 233 and protocol ""p^2"" in field 233.

The flow signature and flow states built up as a result of this exchange are now described. When the packet monitor 300 sees the request packet 206 from the client, a first flow signature 210 is built in the parser subsystem 301 according to the pattern and extraction operations database 308. This signature 210 includes a destination and a source address 240 and 241. One aspect of the invention is that the flow keys are built consistently in a particular order no matter what the direction of conversation. Several mechanisms may be used to achieve this. In the particular embodiment, the numerically lower address is always placed before the numerically higher address. Such least to highest order is used to get the best spread of signatures and hashes for the lookup operations. In this case, therefore, since we assume "S_1"<"C_1", the order is address "S_1" followed by client address "C_1". The next field used to build the signature is a protocol field 242 extracted from packet 206's field 216, and thus is the protocol "p^1". The next field used for the signature is field 243, which contains the destination source port number shown as a crosshatched pattern from the field 218 of the packet 206. This pattern will be recognized in the payload of packets to derive how this packet or sequence of packets exists as a flow. In practice, these may be TCP port numbers, or a combination of TCP port numbers. In the case of the Sun RPC example, the crosshatch represents a set of port numbers of UDS for p^1 that will be used to recognize this flow (*e.g., port* 111). Port 111 indicates this is Sun RPC. Some applications, such as the Sun RPC Bind Lookups, are directly determinable ("known") at the parser level. So in this case, the signature KEY-1 points to a known application denoted "a^1" (Sun RPC Bind Lookup), and a next-state that the state processor should proceed to for more complex recognition jobs, denoted as state "st_D" is placed in the field 245 of the flow-entry.

When the Sun RPC Bind Lookup reply is acquired, a flow signature is again built by the parser. This flow signature is identical to KEY-1. Hence, when the signature enters the analyzer subsystem 303 from the parser subsystem 301, the complete flow-entry is obtained, and in this flow-entry indicates state "st_D". The operations for state "st_D" in the state processor instruction database 326 instructs the state processor to build and store a new flow signature, shown as KEY-2 (212) in FIG. 2. This flow signature built by the state processor also includes the destination and a source addresses 250 and 251, respectively, for server "S_1" followed by (the numerically higher address) client "C_1". A protocol field 252 defines the protocol to be used, *e.g.*, "p^2" which is obtained from the reply packet. A field 253 contains a recognition pattern also obtained from the reply packet. In this case, the application is Sun RPC, and field 254 indicates this application "a^2". A next-state field 255 defines the next state that the state processor should proceed to for more complex recognition jobs, *e.g.*, a state "st^1". In this particular example, this is a final state. Thus, KEY-2 may now be used to recognize packets that are in any way associated with the application "a^2". Two such packets 208 and 209 are shown, one in each direction. They use the particular application service requested in the original Bind Lookup Request, and each will be recognized because the signature KEY-2 will be built in each case.

The two flow signatures 210 and 212 always order the destination and source address fields with server "S_1" followed by client "C_1". Such values are automatically filled in when the addresses are first created in a particular flow signature. Preferably, large collections of flow signatures are kept in a lookup table in a least-to-highest order for the best spread of flow signatures and hashes.

Thereafter, the client and server exchange a number of packets, *e.g.*, represented by request packet 208 and response packet 209. The client 106 sends packets 208 that have a destination and source address S_1 and C_1, in a pair of fields 260 and 261. A field 262 defines the protocol as "p^2", and a field 263 defines the destination port number.

Some network-server application recognition jobs are so simple that only a single state transition has to occur to be able to pinpoint the application that produced the packet. Others require a sequence of state transitions to occur in order to match a known and predefined climb from state-to-state.

Thus the flow signature for the recognition of application "a^2" is automatically set up by predefining what packet-exchange sequences occur for this example when a relatively simple Sun Microsystems Remote Procedure Call bind lookup request instruction executes. More complicated exchanges than this may generate more than two flow signatures and their corresponding states. Each recognition may involve setting up a complex state transition diagram to be traversed before a "final" resting state such as "st_1" in field 255 is reached. All these are used to build the final set of flow signatures for recognizing a particular application in the future.

Re-Using Information from Flows for Maintaining Metrics

The flow-entry of each flow stores a set of statistical measures for the flow, including the total number of packets in the flow, the time of arrival, and the differential time from the last arrival.

Referring again to FIG. 3, the state processing process 328 performs operations defined for the state of the flow, for example for the particular protocol so far identified for the flow. One aspect of the invention is that from time to time, a set of one or more metrics related t the flow may be determined using one or more of the statistical measures stored in the flow-entry. Such metric determining may be carried out, for example, by the state processor running instructions in the state processor instruction and pattern database 326. Such metrics may then be sent by the analyzer subsystem to a host computer connected to the monitor. Alternatively, such metric determining may be carried out by a processor connected to the flow-entry database 324. In our preferred hardware implementation shown in FIG. 10, an analyzer host interface and control 1118 may be configured to configured to access flow-entry records via cache system 1115 to output to a processor via the host bus interface. The processor may then do the reporting of the base metrics.

Fig. 15 describes how the monitor system can be set up with a host computer 1504. The monitor 300 sends metrics from time to time to the host computer 1504, and the host computer 1504 carries out part of the analysis.

This following section describes how the monitor of the invention can be used to monitor the Quality of Service (QOS) by providing QOS Metrics.

Quality of Service Traffic Statistics (Metrics)

This next section defines the common structure that may be applied for the Quality of Service (QOS) Metrics according to one aspect of the invention. It also defines the "original" (or "base") set of metrics that may be determined in an embodiment of the invention to support QOS. The base metrics are determined as part of state processing or by a processor connected to monitor 300, and the QOS metrics are determined from the base metrics by the host computer 1504. The main reason for the breakdown is that the complete QOS metrics may be computationally complex, involving square roots and other functions requiring more computational resources than may be available in real time. The base functions are chosen to be simple to calculate in real time and from which complete QOS metrics may be determined. Other breakdowns of functions clearly are possible within the scope of the invention.

Such metric determining may be carried out, for example, by the state processor running instructions in the state processor instruction and pattern database 326. Such base metrics may then be sent by the analyzer subsystem via a microprocessor or logic circuit connected to the monitor. Alternatively, such metric determining may be carried out by a microprocessor (or some other logic) connected to the flow-entry database 324. In our preferred hardware implementation shown in FIGS. 10 and 11, such a microprocessor is connected cache system 1115 via an analyzer host interface and control 1118 and host bus interface. These components may be configured to access flow-entry records via cache system 1115 to enable the microprocessor to determine and report the base metrics.

The QOS Metrics may broken into the following Metrics Groups. The names are descriptive. The list is not exhaustive, and other metrics may be used. The QOS metrics below include client-to-server (CS) and server-to-client (SC) metrics.

Traffic Metrics such as CSTraffic and SCTraffic.

Jitter Metrics such as CSTraffic and CS Traffic.

Exchange Response Metrics such as CSExchangeResponseTimeStartToStart, CSExchangeResponseTimeEndToStart, CSExchangeResponseTimeStartToEnd, SCExchangeResponseTimeStartToStart, SCExchangeResponseTimeEndToStart, and SCExchangeResponseTimeStartToEnd.

Transaction Response Metrics such as CSTransactionResponseTimeStartToStart, CSApplicationResponseTimeEndToStart, CSApplicationResponseTimeStartToEnd, SCTransactionResponseTimeStartToStart, SCApplicationResponseTimeEndToStart, and SCApplicationResponseTimeStartToEnd.

Connection Metrics such as ConnectionEstablishment and ConnectionGracefulTermination, and ConnectionTimeoutTermination.

Connection Sequence Metrics such as CSConnectionRetransmissions, SCConnectionRetransmissions, and CSConnectionOutOfOrders, SCConnectionOutOfOrders.

Connection Window Metrics, CSConnectionWindow, SCConnectionWindow, CSConnectionFrozenWindows, SCConnectionFrozenWindows, CSConnectionClosedWindows, and SCConnectionClosedWindows

QOS Base Metrics

The simplest means of representing a group of data is by frequency distributions in sub-ranges. In the preferred embodiment, there are some rules in creating the sub-ranges. First the range needs to be known. Second a sub-range size needs to be determined. Fixed sub-range sizes are preferred, alternate embodiments may use variable sub-range sizes.

Determining complete frequency distributions may be computationally expensive. Thus, the preferred embodiment uses metrics determined by summation functions on the individual data elements in a population.

The metrics reporting process provides data that can be used to calculate useful statistical measurements. In one embodiment, the metrics reporting process is part of the state processing that is carried out from time to time according to the state, and in another embodiment, the metrics reporting process carried out from time to time by a microprocessor having access to flow records. Preferably, the metrics reporting process provides base metrics and the final QOS metrics calculations are carried out by the host computer 1504. In addition to keeping the real time state processing simple, the partitioning of the tasks in this way provides metrics that are scalable. For example, the base metrics from two intervals may be combined to metrics for larger intervals.

Consider, for example is the arithmetic mean defined as the sum of the data divided by the number of data elements.

$$\overline{X} = \frac{\sum x}{N}$$

Two base metrics provided by the metrics reporting process are the sum of the x, and the number of elements N. The host computer 1504 performs the division to obtain the average. Furthermore, two sets base metrics for two intervals may be combined by adding the sum of the x's and by adding the number of elements to get a combined sum and number of elements. The average formula then works just the same.

The base metrics have been chosen to maximize the amount of data available while minimizing the amount of memory needed to store the metric and minimizing the processing requirement needed to generate the metric. The base metrics are provided in a metric data structure that contains five unsigned integer values.

• N	count of the number of data points for the metric.
• Σ X	sum of all the data point values for the metric.
• $\Sigma (X^2)$	sum of all the data point values squared for the metric.
• X_{max}	maximum data point value for the metric.
• X_{min}	minimum data point value for the metric.

A metric is used to describe events over a time interval. The base metrics are determined from statistical measures maintained in flow-entries. It is not necessary to cache all the events and then count them at the end of the interval. The base metrics have also been designed to be easily scaleable in terms of combining adjacent intervals.

The following rules are applied when combining base metrics for contiguous time intervals.

• N	ΣN
• ΣX	Σ(Σ (X))
• Σ (X^2)	Σ(Σ (X^2))
• X_{max}	MAX(X_{max})
• X_{min}	MIN(X_{min})

In addition to the above five values, a “trend” indicator is included in the preferred embodiment data structure. This is provided by an enumerated type. The reason for this is that the preferred method of generating trend information is by subtract an initial first value for the interval from the final value for the interval. Only the sign of the resulting number may have value, for example, to determine an indication of trend.

Typical operations that may be performed on the base metrics include:

- Number N.
- Frequency $\dfrac{N}{TimeInterval}$.
- Maximum $X_{\max}$.
- Minimum $X_{\min}$ ’
- Range $R = X_{\max} - X_{\min}$.
- Arithmetic Mean $\overline{X} = \dfrac{\sum X}{N}$.
- Root Mean Square $RMS = \sqrt{\dfrac{\sum\left(X^2\right)}{N}}$.

- Variance $\sigma^2 = \frac{\sum(X-\overline{X})^2}{N} = \frac{\left(\sum X^2\right) - 2\overline{X}\left(\sum X\right) + N\left(\overline{X}^2\right)}{N}$.

- Standard Deviation $\sigma = \sqrt{\frac{\sum\left((X-\overline{X})^2\right)}{N}} = \sqrt{\frac{\left(\sum\left(X^2\right)\right) - 2\overline{X}\left(\sum X\right) + N\left(\overline{X}^2\right)}{N}}$.

- Trend information, which may be the trend between polled intervals and the trend within an interval. Trending between polled intervals is a management application function. Typically the management station would trend on the average of the reported interval. The trend within an interval is presented as an enumerated type and can easily be generated by subtracting the first value in the interval from the last and assigning trend based on the sign value.

Alternate Embodiments

One or more of the following different data elements may be included in various implementation of the metric.

- Sum of the deltas (i.e., differential values). The trend enumeration can be based on this easy calculation.

- Sum of the absolute values of the delta values. This would provide a measurement of the overall movement within an interval.

- Sum of positive delta values and sum of the negative delta values. Expanding each of these with an associated count and maximum would give nice information.

- The statistical measurement of skew can be obtained by adding $\Sigma(X^3)$ to the existing metric.

- The statistical measurement of kurtosis can be obtained by adding $\Sigma(X^3)$ and $\Sigma(X^4)$ to the existing metric.

- Data to calculate a slope of a least-squares line through the data..

Various metrics are now described in more detail.

Traffic Metrics

CSTraffic

Definition

This metric contains information about the volume of traffic measured for a given application and either a specific Client-Server Pair or a specific Server and all of its clients.

This information duplicates, somewhat, that which may be found in the standard, RMON II, AL/NL Matrix Tables. It has been included here for convenience to applications and the associated benefit of improved performance by avoiding the need to access different functional RMON areas when performing QOS Analysis.

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Packets	Count of the # of Packets from the Client(s) to the Server
Σ	Applicable	Octets	Sum total of the # of Octets in these packets from the Client(s) to the Server.
Maximum	Not Applicable		
Minimum	Not Applicable		

SCTraffic

Definition

This metric contains information about the volume of traffic measured for a given application and either a specific Client-Server Pair or a specific Server and all of its clients.

This information duplicates, somewhat, that which may be found in the standard, RMON II, AL/NL Matrix Tables. It has been included here for convenience to applications and the associated benefit of improved performance by avoiding the need to access different functional RMON areas when performing QOS Analysis.

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Packets	Count of the # of Packets from the Server to the Client(s)
Σ	Applicable	Octets	Sum total of the # of Octets in these packets from the Server to the Client(s).
Maximum	Not Applicable		
Minimum	Not Applicable		

Jitter Metrics

CSJitter

Definition

This metric contains information about the Jitter (e.g. Inter-packet Gap) measured for data packets for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, *CSJitter* measures the Jitter for Data Messages from the Client to the Server.

A Data Message starts with the 1st Transport Protocol Data Packet/Unit (TPDU) from the Client to the Server and is demarcated (or terminated) by 1st subsequent Data Packet in the other direction. Client to Server Inter-packet Gaps are measured between Data packets within the Message. Note that in our implementaions, ACKnowledgements are not considered within the measurement of this metric.

Also, there is no consideration in the measurement for retransmissions or out-of-order data packets. The interval between the last packet in a Data Message from the Client to the Server and the 1st packet of the Next Message in the same direction is not interpreted as an Inter-Packet Gap.

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Inter-Packet Gaps	Count of the # of Inter-Packet Gaps measured for Data from the Client(s) to the Server
Σ	Applicable	uSeconds	Sum total of the Delta Times in these Inter-Packet Gaps
Maximum	Applicable	uSeconds	The maximum Delta Time of Inter-Packet Gaps measured
Minimum	Applicable	uSeconds	The minimum Delta Time of Inter-Packet Gaps measured.

SCJitter

Definition

This metric contains information about the Jitter (e.g. Inter-packet Gap) measured for data packets for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, *SCJitter* measures the Jitter for Data Messages from the Client to the Server.

A Data Message starts with the 1st Transport Protocol Data Packet/Unit (TPDU) from the Server to the Client and is demarcated (or terminated) by 1st subsequent Data Packet in the other direction. Server to Client Inter-packet Gaps are measured between Data packets within the Message. Note that in our implementaions, ACKnowledgements are not considered within the measurement of this metric.

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Inter-Packet Gaps	Count of the # of Inter-Packet Gaps measured for Data from the Server to the Client(s).
Σ	Applicable	uSeconds	Sum total of the Delta Times in these Inter-Packet Gaps.
Maximum	Applicable	uSeconds	The maximum Delta Time of Inter-Packet Gaps measured
Minimum	Applicable	uSeconds	The minimum Delta Time of Inter-Packet Gaps measured.

Exchange Response Metrics

CSExchangeResponseTimeStartToStart

Definition

This metric contains information about the Transport-level response time measured for data packets for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, *CSExchangeResponseTimeStartToStart* measures the response time between **start** of Data Messages from the Client to the Server and the **start** of their subsequent response Data Messages from the Server to the Client.

A Client->Server Data Message starts with the 1st Transport Protocol Data Packet/Unit (TPDU) from the Client to the Server and is demarcated (or terminated) by 1st subsequent Data Packet in the other direction. The total time between the start of the Client->Server Data Message and the start of the Server->Client Data Message is measured with this metric. Note that ACKnowledgements are not considered within the measurement of this metric.

Also, there is no consideration in the measurement for retransmissions or out-of-order data packets.

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Client-> Server Messages	Count of the # Client->Server Messages measured for Data Exchanges from the Client(s) to the Server
Σ	Applicable	uSeconds	Sum total of the Start-to-Start Delta Times in these Exchange Response Times
Maximum	Applicable	uSeconds	The maximum Start-to-Start Delta Time of these Exchange Response Times
Minimum	Applicable	uSeconds	The minimum Start-to-Start Delta Time of these Exchange Response Times

CSExchangeResponseTimeEndToStart

Definition

This metric contains information about the Transport-level response time measured for data packets for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, *CSExchangeResponseTimeEndToStart* measures the response time between **end** of Data Messages from the Client to the Server and the **start** of their subsequent response Data Messages from the Server to the Client.

A Client->Server Data Message starts with the 1st Transport Protocol Data Packet/Unit (TPDU) from the Client to the Server and is demarcated (or terminated) by 1st subsequent Data Packet in the other direction. The total time between the end of the Client->Server Data Message and the start of the Server->Client Data Message is measured with this metric. Note that ACKnowledgements are not considered within the measurement of this metric.

Also, there is no consideration in the measurement for retransmissions or out-of-order data packets.

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Client-> Server Messages	Count of the # Client->Server Messages measured for Data Exchanges from the Client(s) to the Server
Σ	Applicable	uSeconds	Sum total of the End-to-Start Delta Times in these Exchange Response Times
Maximum	Applicable	uSeconds	The maximum End-to-Start Delta Time of these Exchange Response Times
Minimum	Applicable	uSeconds	The minimum End-to-Start Delta Time of these Exchange Response Times

CSExchangeResponseTimeStartToEnd

Definition

This metric contains information about the Transport-level response time measured for data packets for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, *CSExchangeResponseTimeEndToStart* measures the response time between **Start** of Data Messages from the Client to the Server and the **End** of their subsequent response Data Messages from the Server to the Client.

A Client->Server Data Message starts with the 1st Transport Protocol Data Packet/Unit (TPDU) from the Client to the Server and is demarcated (or terminated) by 1st subsequent Data Packet in the other direction. The end of the Response Message in the other direction (e.g. from the Server to the Client) is demarcated by the last data of the Message prior to the 1st data packet of the **next** Client to Server Message. The total time between the start of the Client->Server Data Message and the end of the Server->Client Data Message is measured with this metric. Note that ACKnowledgements are not considered within the measurement of this metric.

Also, there is no consideration in the measurement for retransmissions or out-of-order data packets.

Metric Specification

Metric	**Applicability**	**Units**	**Description**
N	Applicable	Client-> Server Message Exchanges	Count of the # Client->Server and Server-> Client Exchange message pairs measured for Data Exchanges from the Client(s) to the Server
Σ	Applicable	uSeconds	Sum total of the Start-to-End Delta Times in these Exchange Response Times
Maximum	Applicable	uSeconds	The maximum Start-to-End Delta Time of these Exchange Response Times
Minimum	Applicable	uSeconds	The minimum Start-to-End Delta Time of these Exchange Response Times

SCExchangeResponseTimeStartToStart

Definition

This metric contains information about the Transport-level response time measured for data packets for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, *SCExchangeResponseTimeStartToStart* measures the response time between **start** of Data Messages from the Server to the Client and the **start** of their subsequent response Data Messages from the Client to the Server.

A Server->Client Data Message starts with the 1st Transport Protocol Data Packet/Unit (TPDU) from the Server to the Client and is demarcated (or terminated) by 1st subsequent Data Packet in the other direction. The total time between the start of the

Server->Client Data Message and the start of the Client->Sever Data Message is measured with this metric. Note that ACKnowledgements are not considered within the measurement of this metric.

Also, there is no consideration in the measurement for retransmissions or out-of-order data packets.

Metric Specification

Metric	**Applicability**	**Units**	**Description**
N	Applicable	Server-> Client Messages	Count of the # Server->Client Messages measured for Data Exchanges from the Client(s) to the Server
Σ	Applicable	uSeconds	Sum total of the Start-to-Start Delta Times in these Exchange Response Times
Maximum	Applicable	uSeconds	The maximum Start-to-Start Delta Time of these Exchange Response Times
Minimum	Applicable	uSeconds	The minimum Start-to-Start Delta Time of these Exchange Response Times

SCExchangeResponseTimeEndToStart

Definition

This metric contains information about the Transport-level response time measured for data packets for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, *SCExchangeResponseTimeEndToStart* measures the response time between **end** of Data Messages from the Server to the Client and the **start** of their subsequent response Data Messages from the Client to the Server.

A Server->Client Data Message starts with the 1st Transport Protocol Data Packet/Unit (TPDU) from the Server to the Client and is demarcated (or terminated) by 1st subsequent Data Packet in the other direction. The total time between the end of the Server->Client Data Message and the start of the Client->Server Data Message is measured with this metric. Note that ACKnowledgements are not considered within the measurement of this metric.

Also, there is no consideration in the measurement for retransmissions or out-of-order data packets.

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Server-> Client Messages	Count of the # Server->Client Messages measured for Data Exchanges from the Client(s) to the Server
Σ	Applicable	uSeconds	Sum total of the End-to-Start Delta Times in these Exchange Response Times
Maximum	Applicable	uSeconds	The maximum End-to-Start Delta Time of these Exchange Response Times
Minimum	Applicable	uSeconds	The minimum End-to-Start Delta Time of these Exchange Response Times

SCExchangeResponseTimeStartToEnd

Definition

This metric contains information about the Transport-level response time measured for data packets for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, *SCExchangeResponseTimeEndToStart* measures the response time between **Start** of Data Messages from the Server to the Client and the **End** of their subsequent response Data Messages from the Client to the Server.

A Server->Client Data Message starts with the 1st Transport Protocol Data Packet/Unit (TPDU) from the Server to the Client and is demarcated (or terminated) by 1st subsequent Data Packet in the other direction. The end of the Response Message in the other direction (e.g. from the Server to the Client) is demarcated by the last data of the Message prior to the 1st data packet of the **next** Server to Client Message. The total time between the start of the Server->Client Data Message and the end of the Client->Server Data Message is measured with this metric. Note that ACKnowledgements are not considered within the measurement of this metric.

Also, there is no consideration in the measurement for retransmissions or out-of-order data packets.

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Client-Server Message Exchanges	Count of the # Server->Client and Client-> Server Exchange message pairs measured for Data Exchanges from the Server to the Client(s)
Σ	Applicable	uSeconds	Sum total of the Start-to-End Delta Times in these Exchange Response Times
Maximum	Applicable	uSeconds	The maximum Start-to-End Delta Time of these Exchange Response Times
Minimum	Applicable	uSeconds	The minimum Start-to-End Delta Time of these Exchange Response Times

Transaction Response Metrics

CSTransactionResponseTimeStartToStart

Definition

This metric contains information about the **Application-level response time** measured for application transactions for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, *CSTransactionResponseTimeStartToStart* measures the response time between **start** of an application transaction from the Client to the Server and the **start** of their subsequent transaction response from the Server to the Client.

A Client->Server transaction starts with the 1st Transport Protocol Data Packet/Unit (TPDU) of a transaction request from the Client to the Server and is demarcated (or terminated) by 1st subsequent data packet of the response to the transaction request. The total time between the start of the Client->Server transaction request and the start of the actual transaction response from the Server->Client is measured with this metric.

This metric is considered a "best-effort" measurement. Systems implementing this metric should make a "best-effort" to demarcate the start and end of requests and responses with the specific application's definition of a logical transaction. The lowest level of support for this metric would make this metric the equivalent of *CSExchangeResponseTimeStartToStart.*

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Client->Svr Transaction Requests	Count of the # Client->Server Transaction Requests measured for Application requests from the Client(s) to the Server
Σ	Applicable	uSeconds	Sum total of the Start-to-Start Delta Times in these Application Response Times
Maximum	Applicable	uSeconds	The maximum Start-to-Start Delta Time of these Application Response Times
Minimum	Applicable	uSeconds	The minimum Start-to-Start Delta Time of these Application Response Times

CSApplicationResponseTimeEndToStart

Definition

This metric contains information about the **Application-level response time** measured for application transactions for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, *CSApplicationResponseTimeEndToStart* measures the response time between **end** of an application transaction from the Client to the Server and the **start** of their subsequent transaction response from the Server to the Client.

A Client->Server transaction starts with the 1st Transport Protocol Data Packet/Unit (TPDU) of a transaction request from the Client to the Server and is demarcated (or terminated) by 1st subsequent data packet of the response to the transaction request The total time between the end of the Client->Server transaction request and the start of the actual transaction response from the Server->Client is measured with this metric

This metric is considered a "best-effort" measurement. Systems implementing this metric should make a "best-effort" to demarcate the start and end of requests and responses with the specific application's definition of a logical transaction. The lowest level of support for this metric would make this metric the equivalent of *CSExchangeResponseTimeEndToStart.*

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Client->Svr Transaction Requests	Count of the # Client->Server Transaction Requests measured for Application requests from the Client(s) to the Server
Σ	Applicable	uSeconds	Sum total of the End-to-Start Delta Times in these Application Response Times
Maximum	Applicable	uSeconds	The maximum End-to-Start Delta Time of these Application Response Times
Minimum	Applicable	uSeconds	The minimum End-to-Start Delta Time of these Application Response Times

CSApplicationResponseTimeStartToEnd

Definition

This metric contains information about the **Application-level response time** measured for application transactions for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, *CSTransactionResponseTimeStartToEnd* measures the response time between **Start** of an application transaction from the Client to the Server and the **End** of their subsequent transaction response from the Server to the Client.

A Client->Server transaction starts with the 1st Transport Protocol Data Packet/Unit (TPDU) a transaction request from the Client to the Server and is demarcated (or terminated) by 1st subsequent data packet of the response to the transaction request. The end of the Transaction Response in the other direction (e.g. from the Server to the Client) is demarcated by the last data of the transaction response prior to the 1st data of the **next** Client to Server Transaction Request. The total time between the start of the Client->Server transaction request and the end of the Server->Client transaction response is measured with this metric.

This metric is considered a "best-effort" measurement. Systems implementing this metric should make a "best-effort" to demarcate the start and end of requests and responses with the specific application's definition of a logical transaction. The lowest level of support for this metric would make this metric the equivalent of *CSExchangeResponseTimeStartToEnd.*

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Client-> Server Transactions	Count of the # Client<->Server request/response pairs measured for transactions from the Client(s) to the Server
Σ	Applicable	uSeconds	Sum total of the Start-to-End Delta Times in these Application Response Times
Maximum	Applicable	uSeconds	The maximum Start-to-End Delta Time of these Application Response Times
Minimum	Applicable	uSeconds	The minimum Start-to-End Delta Time of these Application Response Times

SCTransactionResponseTimeStartToStart

Definition

This metric contains information about the **Application-level response time** measured for application transactions for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, *SCTransactionResponseTimeStartToStart* measures the response time between **start** of an application transaction from the Server to the Client and the **start** of their subsequent transaction response from the Client to the Server.

A Server->Client transaction starts with the 1st Transport Protocol Data Packet/Unit (TPDU) of a transaction request from the Server to the Client and is demarcated (or terminated) by 1st subsequent data packet of the response to the transaction request. The total time between the start of the Server->Client transaction request and the start of the actual transaction response from the Client->Server is measured with this metric.

This metric is considered a "best-effort" measurement. Systems implementing this metric should make a "best-effort" to demarcate the start and end of requests and responses with the specific application's definition of a logical transaction. The lowest level of support for this metric would make this metric the equivalent of *SCExchangeResponseTimeStartToStart.*

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Svr->Client Transaction Requests	Count of the # Server->Client Transaction Requests measured for Application requests from the Server to the Client(s)
Σ	Applicable	uSeconds	Sum total of the Start-to-Start Delta Times in these Application Response Times
Maximum	Applicable	uSeconds	The maximum Start-to-Start Delta Time of these Application Response Times
Minimum	Applicable	uSeconds	The minimum Start-to-Start Delta Time of these Application Response Times

SCApplicationResponseTimeEndToStart

Definition

This metric contains information about the **Application-level response time** measured for application transactions for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, *SCApplicationResponseTimeEndToStart* measures the response time between **end** of an application transaction from the Server to the Client and the **start** of their subsequent transaction response from the Client to the Server.

A Server->Client transaction starts with the 1st Transport Protocol Data Packet/Unit (TPDU) of a transaction request from the Server to the Client and is demarcated (or terminated) by 1st subsequent data packet of the response to the transaction request The total time between the end of the Server->Client transaction request and the start of the actual transaction response from the Client->Server is measured with this metric

This metric is considered a "best-effort" measurement. Systems implementing this metric should make a "best-effort" to demarcate the start and end of requests and responses with the specific application's definition of a logical transaction. The lowest level of support for this metric would make this metric the equivalent of *SCExchangeResponseTimeEndToStart*.

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Svr->Client Transaction Requests	Count of the # Server->Client Transaction Requests measured for Application requests from the Server to the Client(s)
Σ	Applicable	uSeconds	Sum total of the End-to-Start Delta Times in these Application Response Times
Maximum	Applicable	uSeconds	The maximum End-to-Start Delta Time of these Application Response Times
Minimum	Applicable	uSeconds	The minimum End-to-Start Delta Time of these Application Response Times

SCApplicationResponseTimeStartToEnd

Definition

This metric contains information about the **Application-level response time** measured for application transactions for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, *SCTransactionResponseTimeStartToEnd* measures the response time between **Start** of an application transaction from the Server to the Client and the **End** of their subsequent transaction response from the Client to the Server.

A Server->Client transaction starts with the 1st Transport Protocol Data Packet/Unit (TPDU) a transaction request from the Server to the Client and is demarcated (or terminated) by 1st subsequent data packet of the response to the transaction request. The end of the Transaction Response in the other direction (e.g. from the Client to the Server) is demarcated by the last data of the transaction response prior to the 1st data of the **next** Server to Client Transaction Request. The total time between the start of the Server->Client transaction request and the end of the Client->Server transaction response is measured with this metric.

This metric is considered a "best-effort" measurement. Systems implementing this metric should make a "best-effort" to demarcate the start and end of requests and responses with the specific application's definition of a logical transaction. The lowest level of support for this metric would make this metric the equivalent of *SCExchangeResponseTimeStartToEnd.*

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Server-> Client Transactions	Count of the # Server<->Client request/response pairs measured for transactions from the Server to the Client(s)
Σ	Applicable	uSeconds	Sum total of the Start-to-End Delta Times in these Application Response Times
Maximum	Applicable	uSeconds	The maximum Start-to-End Delta Time of these Application Response Times
Minimum	Applicable	uSeconds	The minimum Start-to-End Delta Time of these Application Response Times

Connection Metrics

ConnectionEstablishment

Definition

This metric contains information about the transport-level connection establishment for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, *ConnectionsEstablishment* measures number of connections established the Client(s) to the Server. The information contain, in essence, includes:

- # Transport Connections Successfully established
- Set-up Times of the established connections
- Max. # of Simultaneous established connections.
- # Failed Connection establishment attempts (due to either timeout or rejection)

Note that the "# of CURRENT Established Transport Connections" may be derived from this metric along with the *ConnectionGracefulTermination* and *ConnectionTimeoutTermination* metrics, as follows:

# current connections :== "# successfully established"
- "# terminated gracefully"
- "# terminated by time-out"

The set-up time of a connection is defined to be the delta time between the first transport-level, Connection Establishment Request (*i.e.,* SYN, CR-TPDU, etc.) and the first Data Packet exchanged on the connection.

Metric Specification

Metric	**Applicability**	**Units**	**Description**
N	Applicable	Connections	Count of the # Connections Established from the Client(s) to the Server
Σ	Applicable	uSeconds	Sum total of the Connection Set-up Times in these Established connections
Maximum	Applicable	Connections	Count of the MAXIMUM simultaneous # Connections Established from the Client(s) to the Server
Minimum	Not Applicable	Connections	Count of the Failed simultaneous # Connections Established from the Client(s) to the Server

ConnectionGracefulTermination

Definition

This metric contains information about the transport-level connections terminated gracefully for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, *ConnectionsGracefulTermination* measures gracefully terminated connections both in volume and summary connection duration. The information contain, in essence, includes:

- # Gracefully terminated Transport Connections
- Durations (lifetimes) of gracefully terminated connections.

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Connections	Count of the # Connections Gracefully Terminated between Client(s) to the Server
Σ	Applicable	mSeconds	Sum total of the Connection Durations (Lifetimes) of these terminated connections
Maximum	Not Applicable		
Minimum	Not Applicable		

ConnectionTimeoutTermination

Definition

This metric contains information about the transport-level connections terminated non-gracefully (e.g. Timed-Out) for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, *ConnectionsTimeoutTermination* measures previously established and timed-out connections both in volume and summary connection duration. The information contain, in essence, includes:

- # Timed-out Transport Connections
- Durations (lifetimes) of timed-out terminated connections.

The duration factor of this metric is considered a "best-effort" measurement. Independent network monitoring devices cannot really know when network entities actually detect connection timeout conditions and hence may need to extrapolate or estimate when connection timeouts actually occur.

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Connections	Count of the # Connections Timed-out between Client(s) to the Server
Σ	Applicable	mSeconds	Sum total of the Connection Durations (Lifetimes) of these terminated connections
Maximum	Not Applicable		
Minimum	Not Applicable		

Connection Sequence Metrics

CSConnectionRetransmissions

Definition

This metric contains information about the transport-level connection health for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, CS*ConnectionRetransmissions* measures number of actual events within established connection lifetimes in which Transport, data-bearing PDUs (packets) from the Client->Server were retransmitted.

Note that retransmission events as seen by the Network Monitoring device indicate the "duplicate" presence of a TPDU as observed on the network.

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Events	Count of the # Data TPDU retransmissions from the Client(s) to the Server
Σ	Not Applicable		
Maximum	Not Applicable		
Minimum	Not Applicable		

SCConnectionRetransmissions

Definition

This metric contains information about the transport-level connection health for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, SC*ConnectionRetransmissions* measures number of actual events within established connection lifetimes in which Transport, data-bearing PDUs (packets) from the Server->Client were retransmitted.

Note that retransmission events as seen by the Network Monitoring device indicate the "duplicate" presence of a TPDU as observed on the network.

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Events	Count of the # Data TPDU retransmissions from the Server to the Client(s)
Σ	Not Applicable		
Maximum	Not Applicable		
Minimum	Not Applicable		

CSConnectionOutOfOrders

Definition

This metric contains information about the transport-level connection health for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, CS*ConnectionOutOfOrders* measures number of actual events within established connection lifetimes in which Transport, data-bearing PDUs (packets) from the Client->Server were detected as being out of sequential order.

Note that retransmissions (or duplicates) are considered to be different than out-of-order events and are tracked separately in the CS*ConnectionRetransmissions* metric.

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Events	Count of the # Out-of-Order TPDU events from the Client(s) to the Server
Σ	Not Applicable		
Maximum	Not Applicable		
Minimum	Not Applicable		

SCConnectionOutOfOrders

Definition

This metric contains information about the transport-level connection health for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, SC*ConnectionOutOfOrders* measures number of actual events within established connection lifetimes in which Transport, data-bearing PDUs (packets) from the Server->Client were detected as being out of sequential order.

Note that retransmissions (or duplicates) are considered to be different than out-of-order events and are tracked separately in the SC*ConnectionRetransmissions* metric.

Metric Specification

Metric	**Applicability**	**Units**	**Description**
N	Applicable	Events	Count of the # Out-of-Order TPDU events from the Server to the Client(s)
Σ	Not Applicable		
Maximum	Not Applicable		
Minimum	Not Applicable		

Connection Window Metrics

CSConnectionWindow

Definition

This metric contains information about the transport-level connection windows for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, CS*ConnectionWindow* measures number of Transport-level Acknowledges within established connection lifetimes and their relative sizes from the Client->Server.

Note that the number of DATA TPDUs (packets) may be estimated by differencing the Acknowledge count of this metric and the overall traffic from the Client to the Server (see *CSTraffic* above). A slight error in this calculation may occur due to Connection Establishment and Termination TPDUS, but it should not be significant.

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Events	Count of the # ACK TPDU retransmissions from the Client(s) to the Server
Σ	Not Applicable	Increments	Sum total of the Window Sizes of the Acknowledges
Maximum	Not Applicable	Increments	The maximum Window Size of these Acknowledges
Minimum	Not Applicable	Increments	The minimum Window Size of these Acknowledges

SCConnectionWindow

Definition

This metric contains information about the transport-level connection windows for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, SS*ConnectionWindow* measures number of Transport-level Acknowledges within established connection lifetimes and their relative sizes from the Server->Client.

Note that the number of DATA TPDUs (packets) may be estimated by differencing the Acknowledge count of this metric and the overall traffic from the Client to the Server (see *SCTraffic* above).. A slight error in this calculation may occur due to Connection Establishment and Termination TPDUS, but it should not be significant.

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Events	Count of the # ACK TPDU retransmissions from the Server to the Client(s)
Σ	Applicable	Increments	Sum total of the Window Sizes of the Acknowledges
Maximum	Applicable	Increments	The maximum Window Size of these Acknowledges
Minimum	Applicable	Increments	The minimum Window Size of these Acknowledges

CSConnectionFrozenWindows

Definition

This metric contains information about the transport-level connection windows for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, CS*ConnectionWindow* measures number of Transport-level Acknowledges from Client->Server within established connection lifetimes which validly acknowledge data, but either

- failed to increase the upper window edge,
- reduced the upper window edge

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Events	Count of the # ACK TPDU with frozen/reduced windows from the Client(s) to the Server
Σ	Not Applicable		
Maximum	Not Applicable		
Minimum	Not Applicable		

SCConnectionFrozenWindows

Definition

This metric contains information about the transport-level connection windows for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, SC*ConnectionWindow* measures number of Transport-level Acknowledges from Server->Client within established connection lifetimes which validly acknowledge data, but either

- failed to increase the upper window edge,
- reduced the upper window edge

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Events	Count of the # ACK TPDU with frozen/reduced windows from the Client(s) to the Server
Σ	Not Applicable		
Maximum	Not Applicable		
Minimum	Not Applicable		

CSConnectionClosedWindows

Definition

This metric contains information about the transport-level connection windows for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, CS*ConnectionWindow* measures number of Transport-level Acknowledges from Client->Server within established connection lifetimes which fully closed the acknowledge/sequence window.

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Events	Count of the # ACK TPDU with Closed windows from the Client(s) to the Server
Σ	Not Applicable		
Maximum	Not Applicable		
Minimum	Not Applicable		

SCConnectionClosedWindows

Definition

This metric contains information about the transport-level connection windows for a given application and either a specific Client-Server Pair or a specific Server and all of its clients. Specifically, SC*ConnectionWindow* measures number of Transport-level Acknowledges from Server->Client within established connection lifetimes which fully closed the acknowledge/sequence window.

Metric Specification

Metric	Applicability	Units	Description
N	Applicable	Events	Count of the # ACK TPDU with Closed windows from the Client(s) to the Server
Σ	Not Applicable		
Maximum	Not Applicable		
Minimum	Not Applicable		

Embodiments of the present invention automatically generate flow signatures with the necessary recognition patterns and state transition climb procedure. Such comes from analyzing packets according to parsing rules, and also generating state transitions to search for. Applications and protocols, at any level, are recognized through state analysis of sequences of packets.

Note that one in the art will understand that computer networks are used to connect many different types of devices, including network appliances such as telephones, "Internet" radios, pagers, and so forth. The term computer as used herein encompasses all such devices and a computer network as used herein includes networks of such computers.

Although the present invention has been described in terms of the presently preferred embodiments, it is to be understood that the disclosure is not to be interpreted as limiting. Various alterations and modifications will no doubt become apparent to those or ordinary skill in the art after having read the above disclosure. Accordingly, it is intended that the claims be interpreted as covering all alterations and modifications as fall within the true spirit and scope of the present invention.

CLAIMS

What is claimed is:

1. 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;

   (b) looking up a flow-entry database comprising none or more flow-entries for previously encountered conversational flows, the looking up to determine if the received packet is of an existing flow;

   (d) if the packet is of an existing flow, updating the flow-entry of the existing flow including storing one or more statistical measures kept in the flow-entry; and

   (e) if the packet is of a new flow, 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 though the connection point is received by the packet acquisition device.

2. A method according to claim 1¸ further including:

   extracting identifying portions from the packet,

   wherein the looking up uses a function of the identifying portions.

3. A method according to claim 1, wherein the steps are carried out in real time on each packet passing through the connection point.

4. 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. 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.

6. A method according to claim 7, wherein the metrics include one or more quality of service (QOS) metrics.

7. 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. A method according to claim 7, further comprising calculating one or more quality of service (QOS) metrics from the base metrics.

9. 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. A method according to claim 1, wherein step (d) 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 (e) includes if the packet is of a new flow, performing any state operations required for the initial state of the new flow.

11. 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. 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. A method according to claim 12, wherein the reporting is part of the state operations for the state of the flow.

14. 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.

15. 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. 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. 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 comprising none or more flow-entries for previously encountered conversational flows to which a received packet may belong; and

(c) an analyzer subsystem coupled to the packet acquisition device configured to lookup 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.

18. 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. 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.

20. 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. A packet monitor according to claim 20, wherein the statistical processor determine and reports the one or more metrics from time to time.

ABSTRACT

A method of and monitor apparatus for analyzing a flow of packets passing through a connection point on a computer network. The method includes receiving a packet from a packet acquisition device, and looking up a flow-entry database containing flow-entries for previously encountered conversational flows. The looking up to determine if the received packet is of an existing flow. Each and every packet is processed. If the packet is of an existing flow, the method updates the flow-entry of the existing flow, including storing one or more statistical measures kept in the flow-entry. If the packet is of a new flow, the method stores 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. The statistical measures are used to determine metrics related to the flow. The metrics may be base metrics from which quality of service metrics are determined, or may be the quality of service metrics.

6839751

1/18

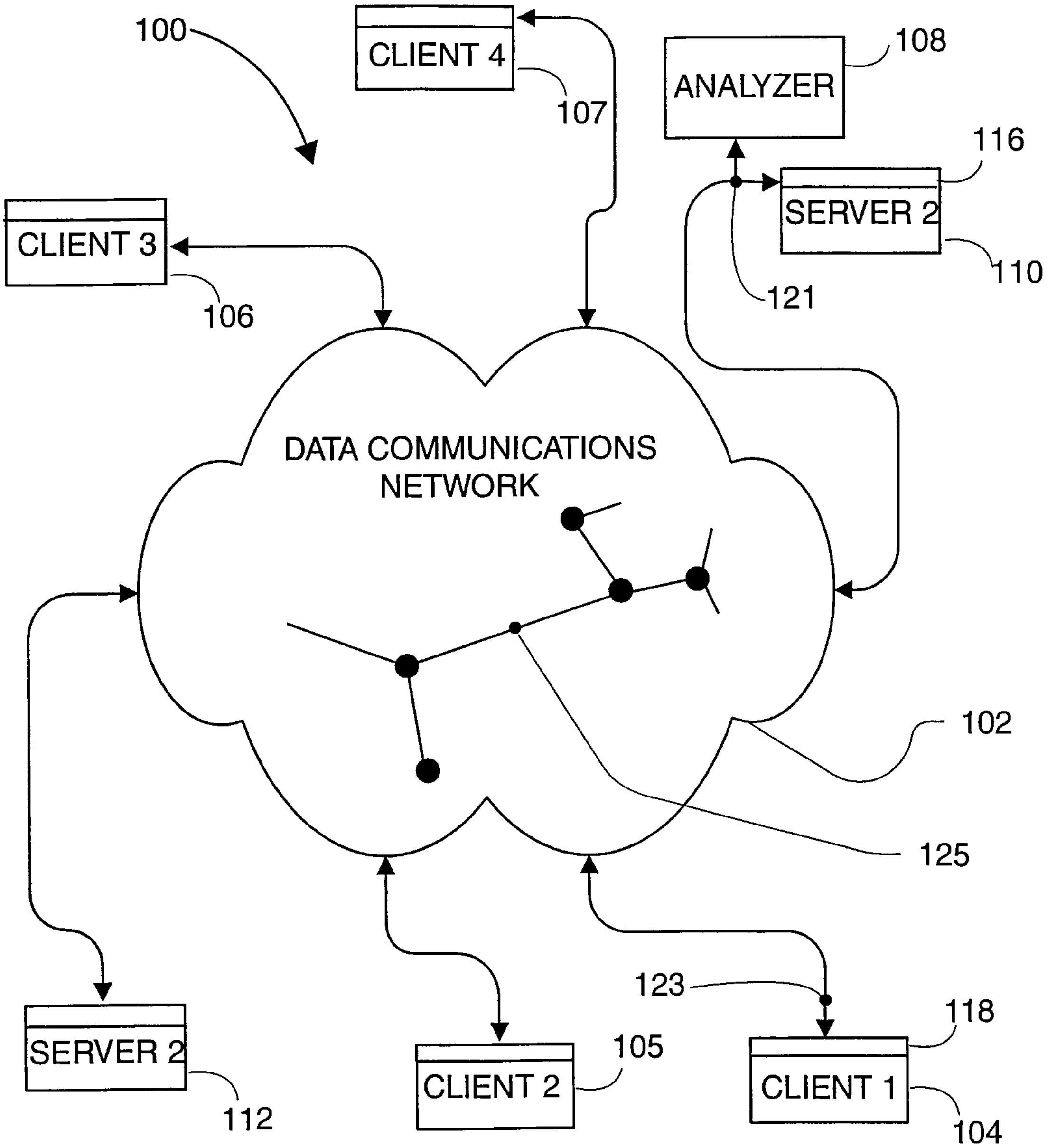


FIG. 1

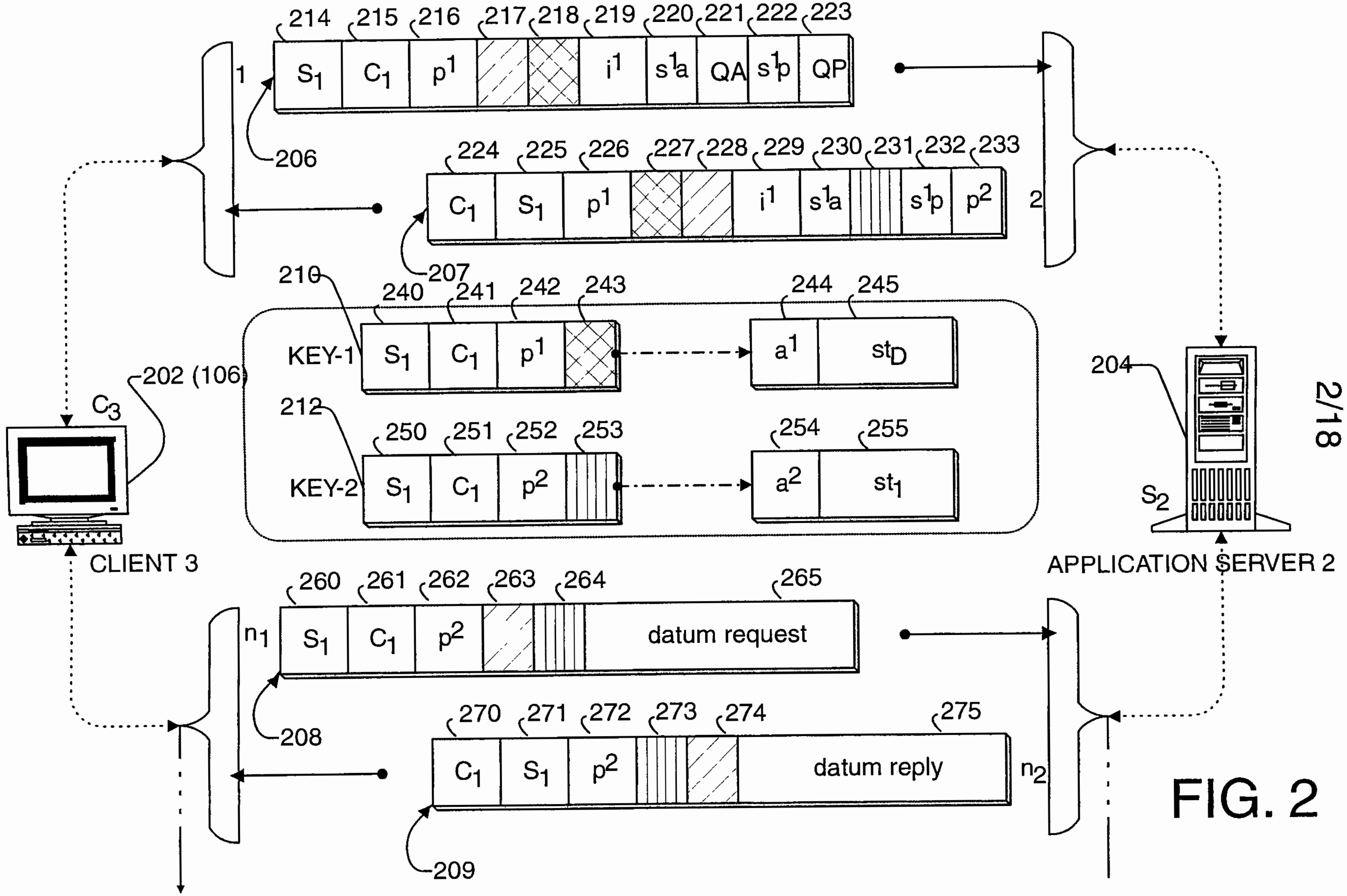

214
215
216
217
218
219
220
221
222
223
1
S_1
C_1
p^1
i^1
s^1a
QA
s^1p
QP
206
224
225
226
227
228
229
230
231
232
233
C_1
S_1
p^1
i^1
s^1a
s^1p
p^2
2
207
210
240
241
242
243
KEY-1
S_1
C_1
p^1
244
245
a^1
st_D
212
250
251
252
253
KEY-2
S_1
C_1
p^2
254
255
a^2
st_1
202 (106)
C_3
CLIENT 3
204
S_2
APPLICATION SERVER 2
260
261
262
263
264
265
n_1
S_1
C_1
p^2
datum request
208
270
271
272
273
274
275
C_1
S_1
p^2
datum reply
n_2
209

FIG. 2

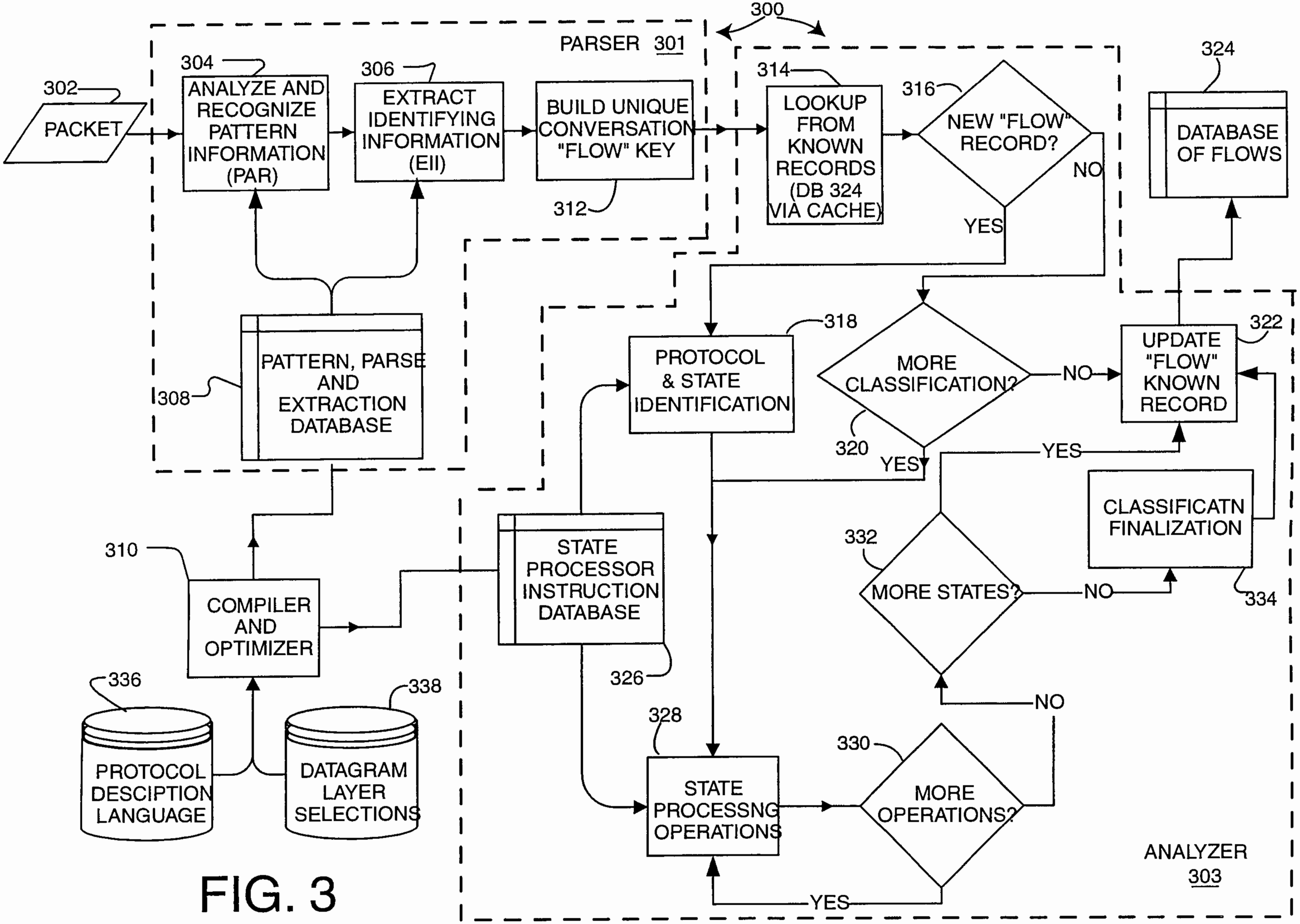

300
PARSER 301
302
PACKET
304
ANALYZE AND RECOGNIZE PATTERN INFORMATION (PAR)
306
EXTRACT IDENTIFYING INFORMATION (EII)
BUILD UNIQUE CONVERSATION "FLOW" KEY
312
308
PATTERN, PARSE AND EXTRACTION DATABASE
314
LOOKUP FROM KNOWN RECORDS (DB 324 VIA CACHE)
316
NEW "FLOW" RECORD?
NO
YES
324
DATABASE OF FLOWS
318
PROTOCOL & STATE IDENTIFICATION
MORE CLASSIFICATION?
320
NO
YES
UPDATE "FLOW" KNOWN RECORD
322
YES
CLASSIFICATN FINALIZATION
334
332
MORE STATES?
NO
310
COMPILER AND OPTIMIZER
STATE PROCESSOR INSTRUCTION DATABASE
326
336
PROTOCOL DESCIPTION LANGUAGE
338
DATAGRAM LAYER SELECTIONS
328
STATE PROCESSNG OPERATIONS
330
MORE OPERATIONS?
NO
YES
ANALYZER 303

FIG. 3

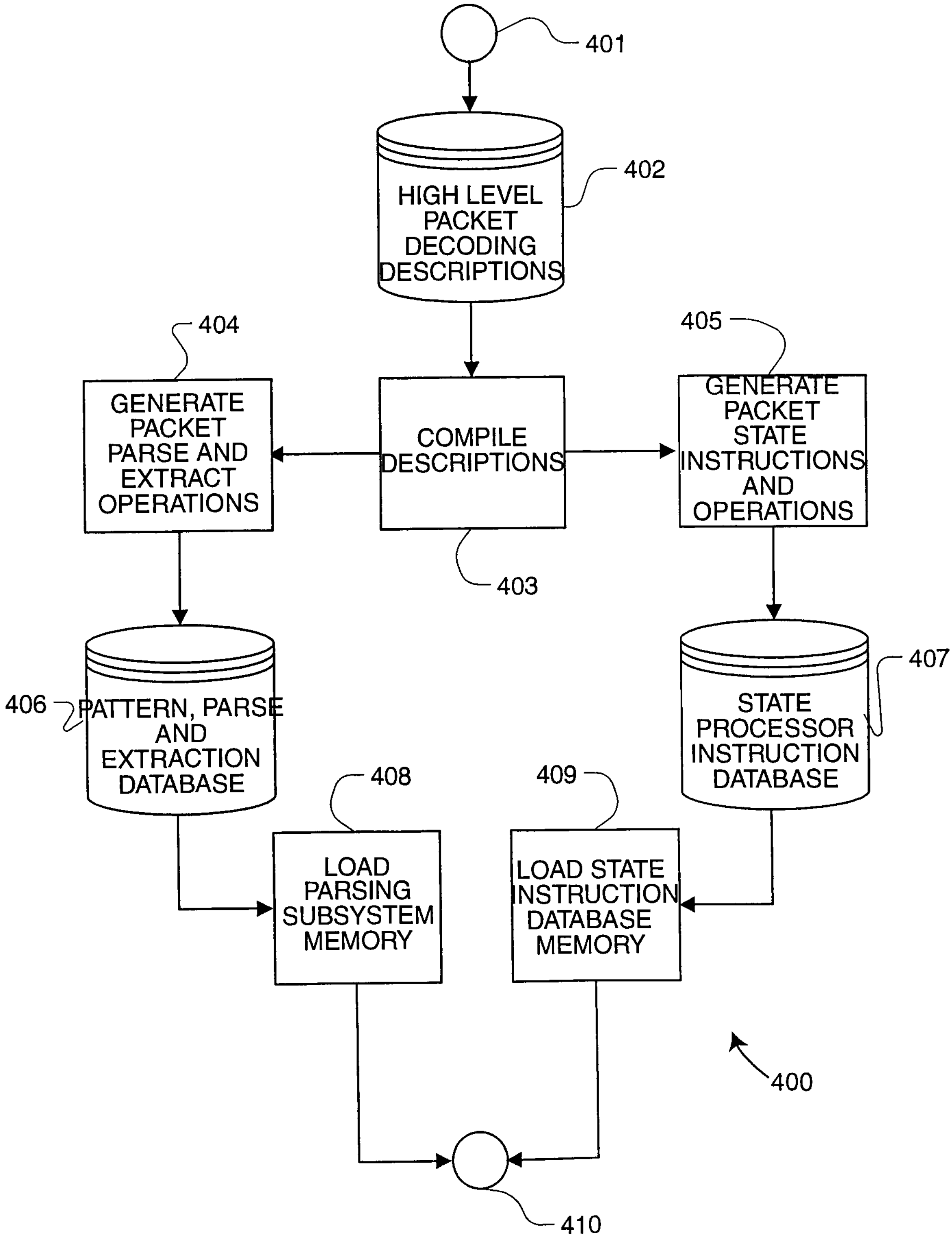

401
HIGH LEVEL PACKET DECODING DESCRIPTIONS
402
COMPILE DESCRIPTIONS
403
GENERATE PACKET PARSE AND EXTRACT OPERATIONS
404
GENERATE PACKET STATE INSTRUCTIONS AND OPERATIONS
405
PATTERN, PARSE AND EXTRACTION DATABASE
406
STATE PROCESSOR INSTRUCTION DATABASE
407
LOAD PARSING SUBSYSTEM MEMORY
408
LOAD STATE INSTRUCTION DATABASE MEMORY
409
400
410

FIG. 4

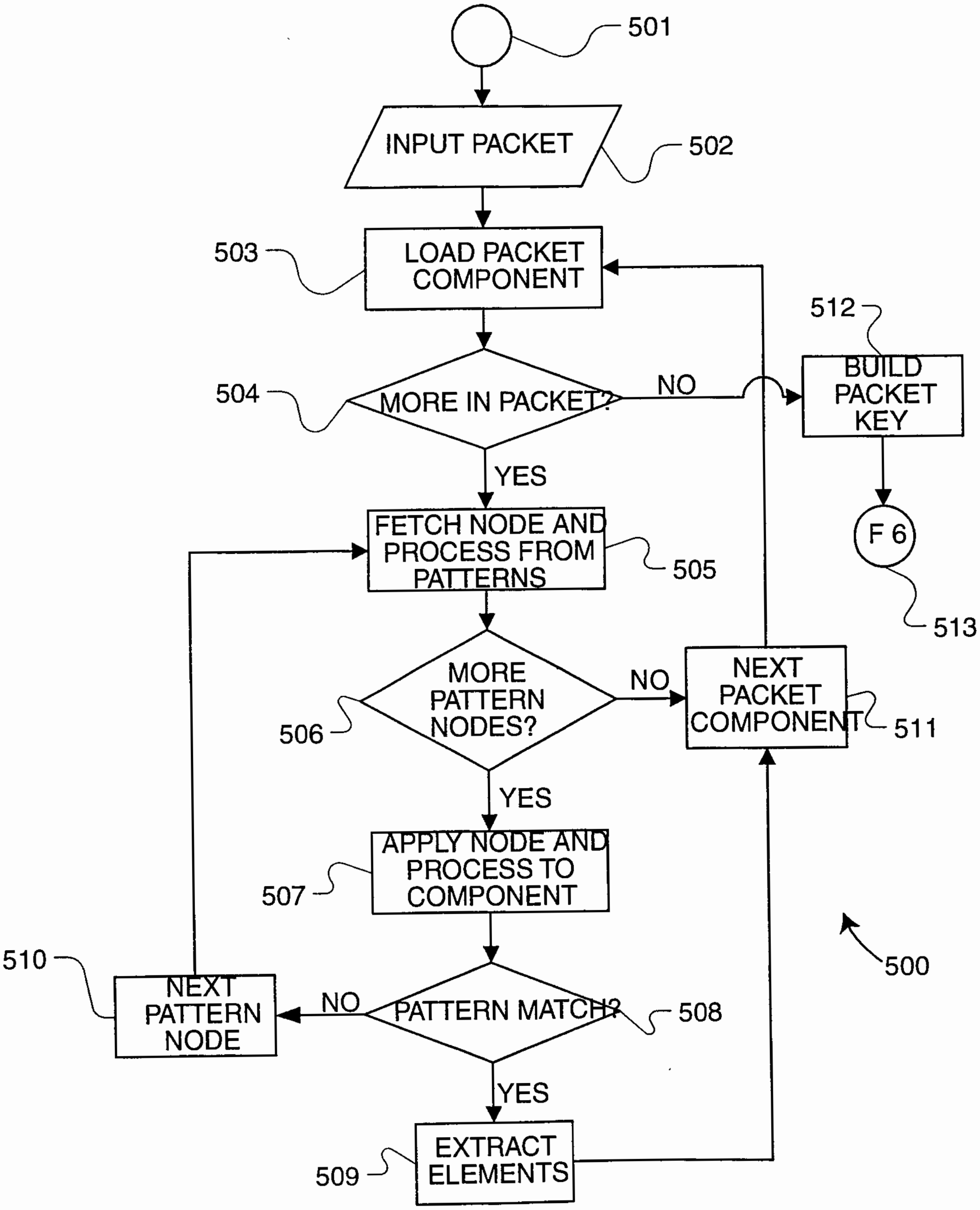

501
INPUT PACKET
502
503
LOAD PACKET COMPONENT
504
MORE IN PACKET?
NO
512
BUILD PACKET KEY
F 6
513
YES
FETCH NODE AND PROCESS FROM PATTERNS
505
506
MORE PATTERN NODES?
NO
NEXT PACKET COMPONENT
511
YES
APPLY NODE AND PROCESS TO COMPONENT
507
500
510
NEXT PATTERN NODE
NO
PATTERN MATCH?
508
YES
509
EXTRACT ELEMENTS

FIG. 5

6/18

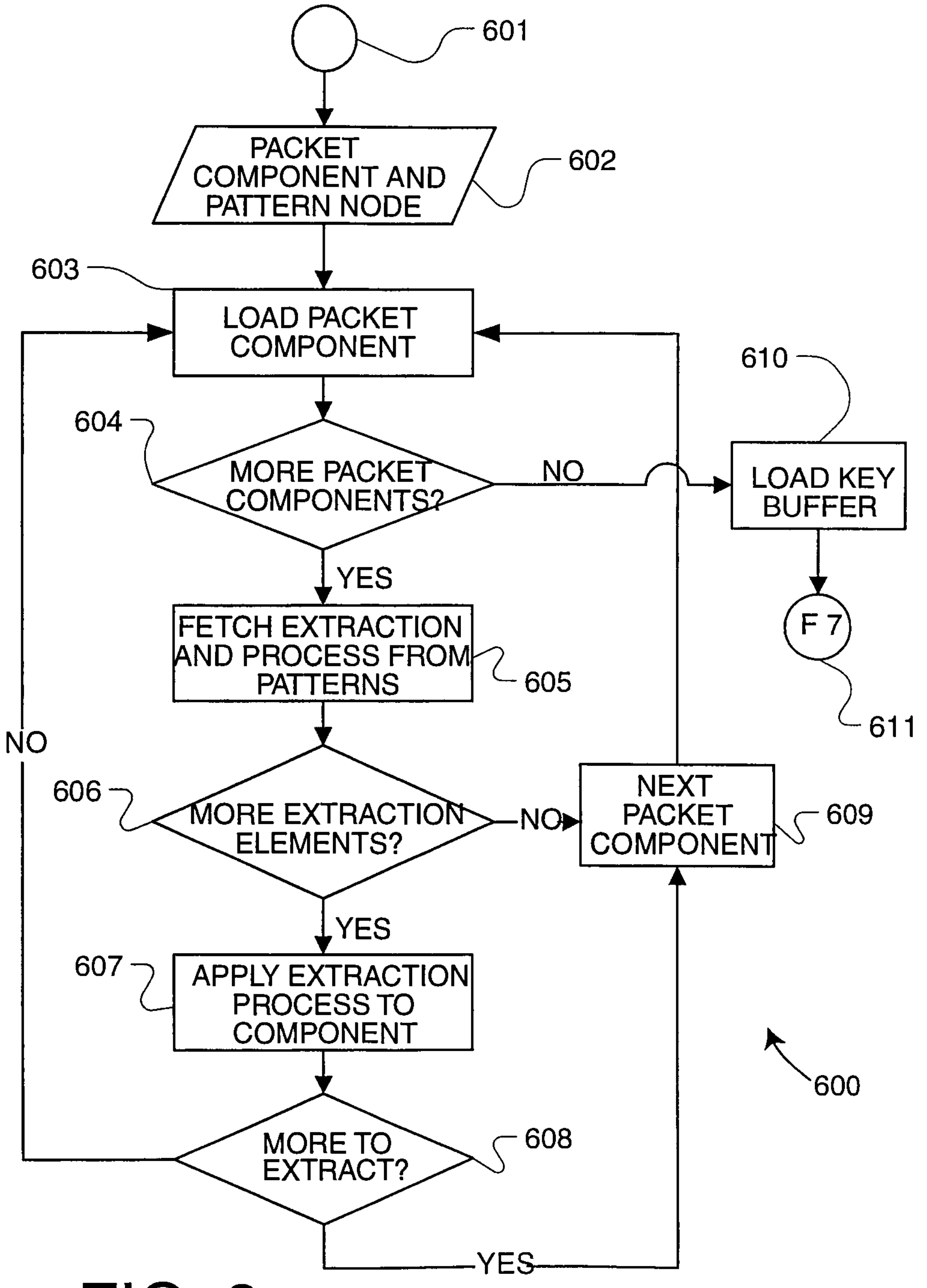

601
PACKET COMPONENT AND PATTERN NODE
602
603
LOAD PACKET COMPONENT
610
604
MORE PACKET COMPONENTS?
NO
LOAD KEY BUFFER
YES
FETCH EXTRACTION AND PROCESS FROM PATTERNS
605
F 7
611
NO
606
MORE EXTRACTION ELEMENTS?
NO
NEXT PACKET COMPONENT
609
YES
607
APPLY EXTRACTION PROCESS TO COMPONENT
600
MORE TO EXTRACT?
608
YES

FIG. 6

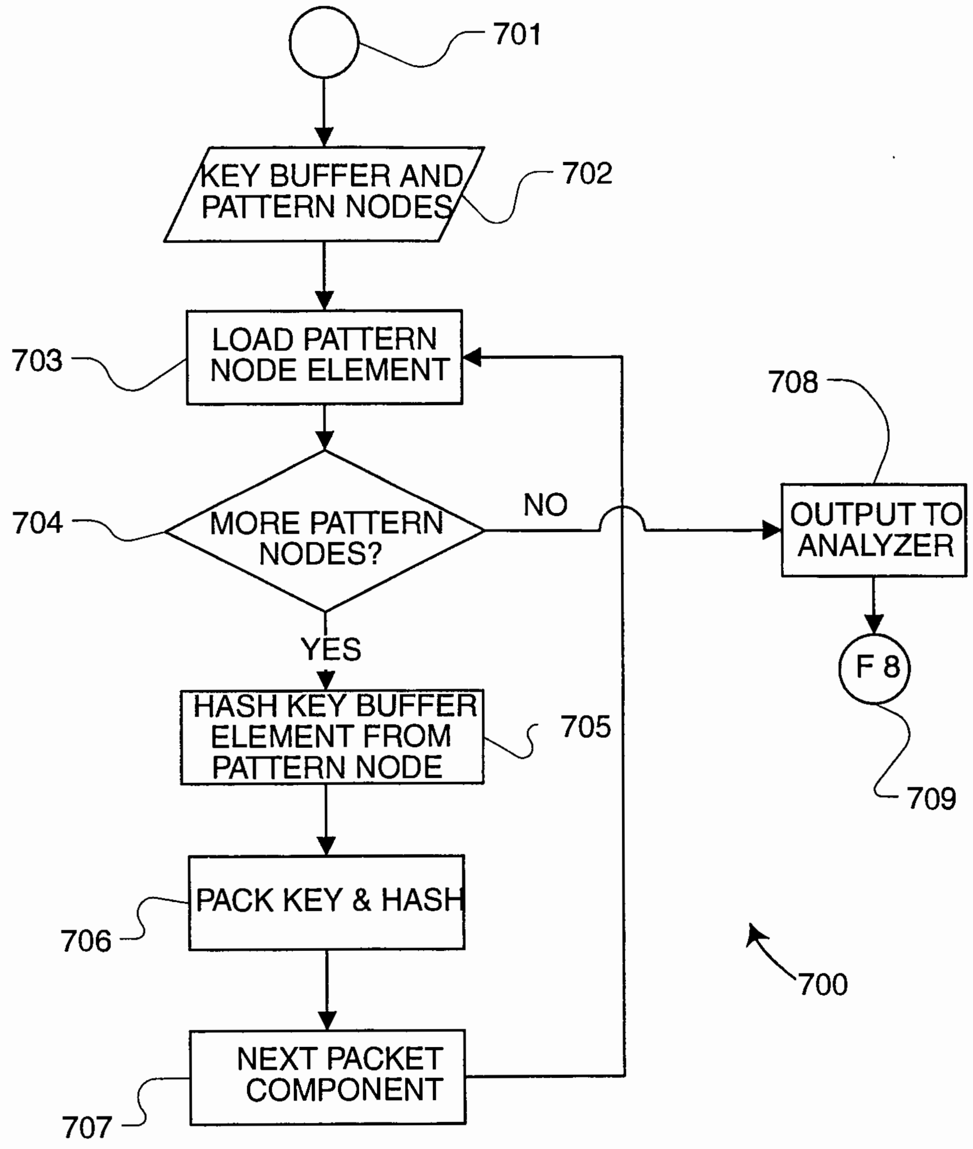

701
KEY BUFFER AND PATTERN NODES
702
703
LOAD PATTERN NODE ELEMENT
704
MORE PATTERN NODES?
NO
YES
708
OUTPUT TO ANALYZER
F 8
709
HASH KEY BUFFER ELEMENT FROM PATTERN NODE
705
706
PACK KEY & HASH
NEXT PACKET COMPONENT
707
700

FIG. 7

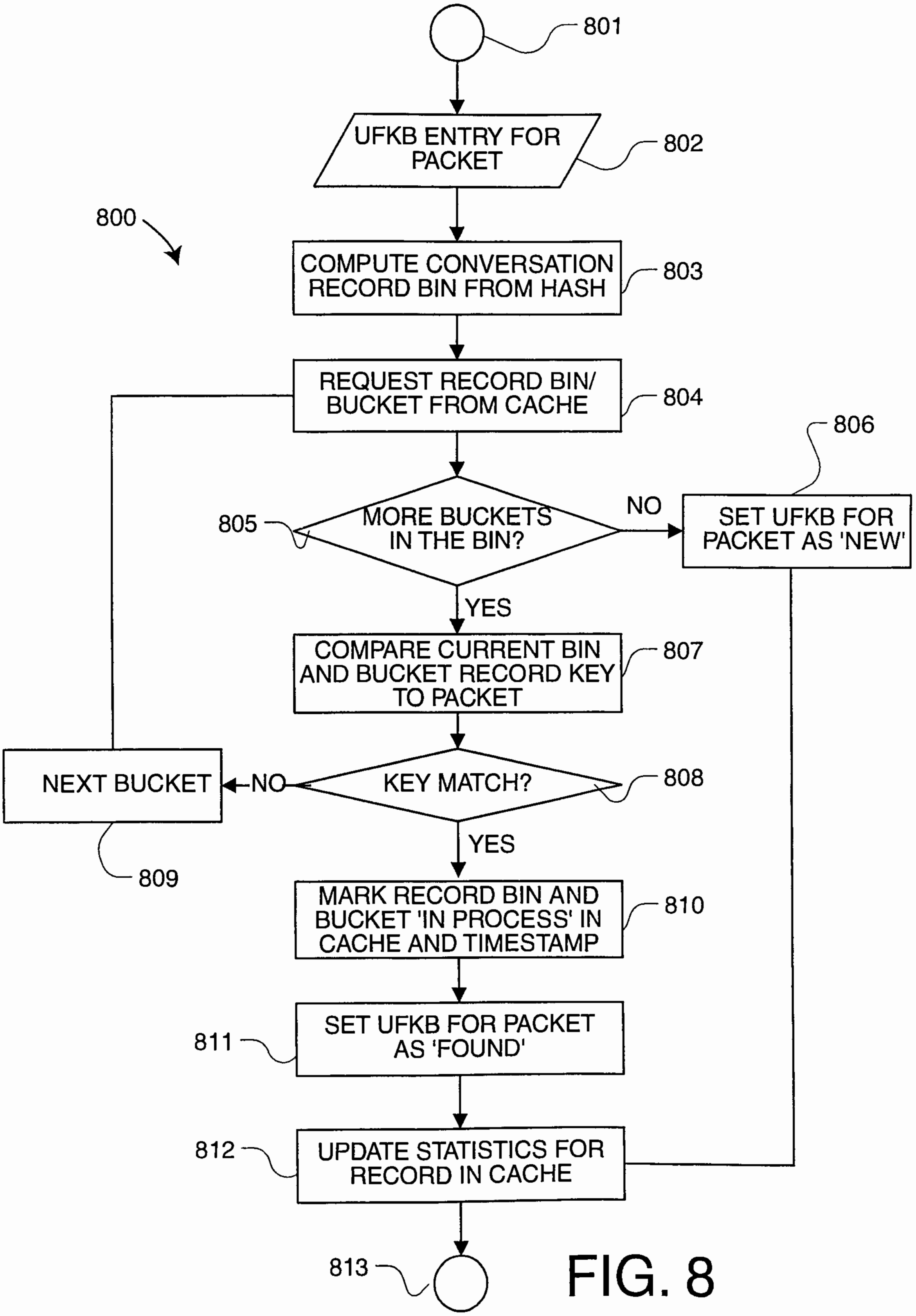

801
UFKB ENTRY FOR PACKET
802
800
COMPUTE CONVERSATION RECORD BIN FROM HASH
803
REQUEST RECORD BIN/ BUCKET FROM CACHE
804
806
805
MORE BUCKETS IN THE BIN?
NO
SET UFKB FOR PACKET AS 'NEW'
YES
COMPARE CURRENT BIN AND BUCKET RECORD KEY TO PACKET
807
NEXT BUCKET
NO
KEY MATCH?
808
809
YES
MARK RECORD BIN AND BUCKET 'IN PROCESS' IN CACHE AND TIMESTAMP
810
811
SET UFKB FOR PACKET AS 'FOUND'
812
UPDATE STATISTICS FOR RECORD IN CACHE
813

FIG. 8

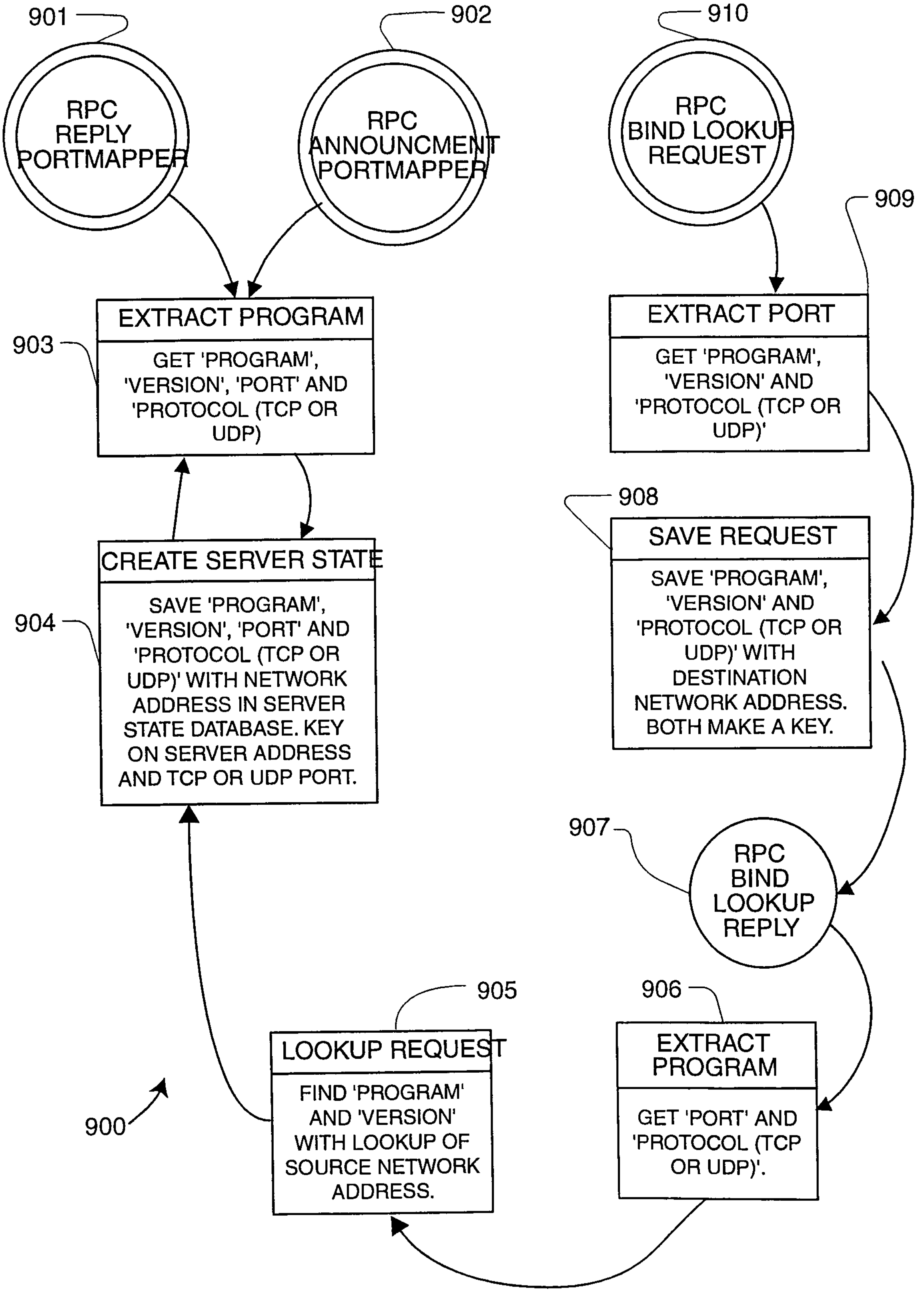

901
RPC REPLY PORTMAPPER
902
RPC ANNOUNCMENT PORTMAPPER
910
RPC BIND LOOKUP REQUEST
909
903
EXTRACT PROGRAM
GET 'PROGRAM', 'VERSION', 'PORT' AND 'PROTOCOL (TCP OR UDP)
EXTRACT PORT
GET 'PROGRAM', 'VERSION' AND 'PROTOCOL (TCP OR UDP)'
908
SAVE REQUEST
SAVE 'PROGRAM', 'VERSION' AND 'PROTOCOL (TCP OR UDP)' WITH DESTINATION NETWORK ADDRESS. BOTH MAKE A KEY.
904
CREATE SERVER STATE
SAVE 'PROGRAM', 'VERSION', 'PORT' AND 'PROTOCOL (TCP OR UDP)' WITH NETWORK ADDRESS IN SERVER STATE DATABASE. KEY ON SERVER ADDRESS AND TCP OR UDP PORT.
907
RPC BIND LOOKUP REPLY
905
906
LOOKUP REQUEST
FIND 'PROGRAM' AND 'VERSION' WITH LOOKUP OF SOURCE NETWORK ADDRESS.
EXTRACT PROGRAM
GET 'PORT' AND 'PROTOCOL (TCP OR UDP)'.
900

FIG. 9

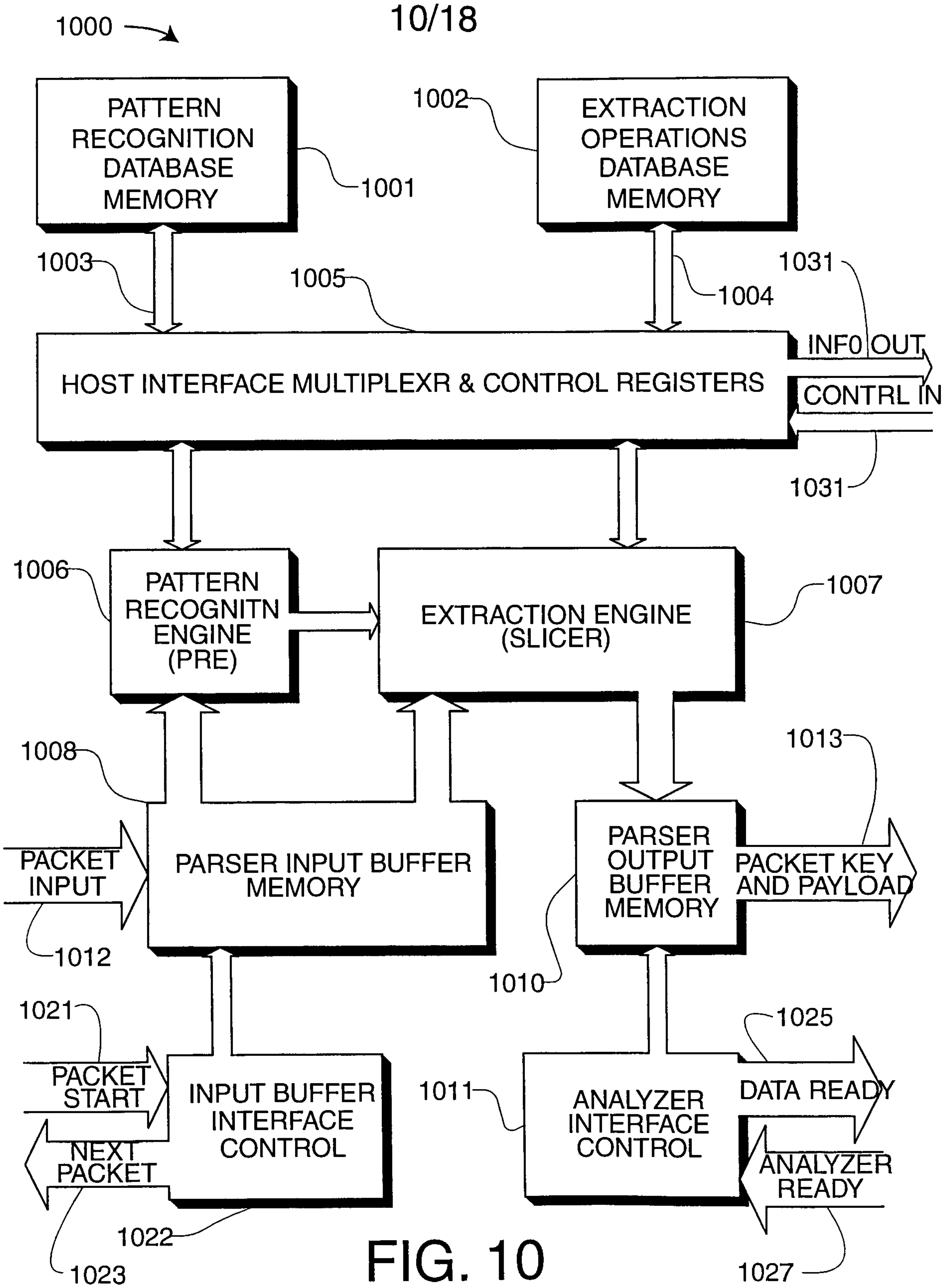

1000
PATTERN RECOGNITION DATABASE MEMORY
1001
1002
EXTRACTION OPERATIONS DATABASE MEMORY
1003
1005
1004
1031
INF0 OUT
HOST INTERFACE MULTIPLEXR & CONTROL REGISTERS
CONTRL IN
1031
1006
PATTERN RECOGNITN ENGINE (PRE)
EXTRACTION ENGINE (SLICER)
1007
1008
1013
PACKET INPUT
PARSER INPUT BUFFER MEMORY
PARSER OUTPUT BUFFER MEMORY
PACKET KEY AND PAYLOAD
1012
1010
1021
1025
PACKET START
INPUT BUFFER INTERFACE CONTROL
1011
ANALYZER INTERFACE CONTROL
DATA READY
NEXT PACKET
ANALYZER READY
1022
1023
1027

FIG. 10

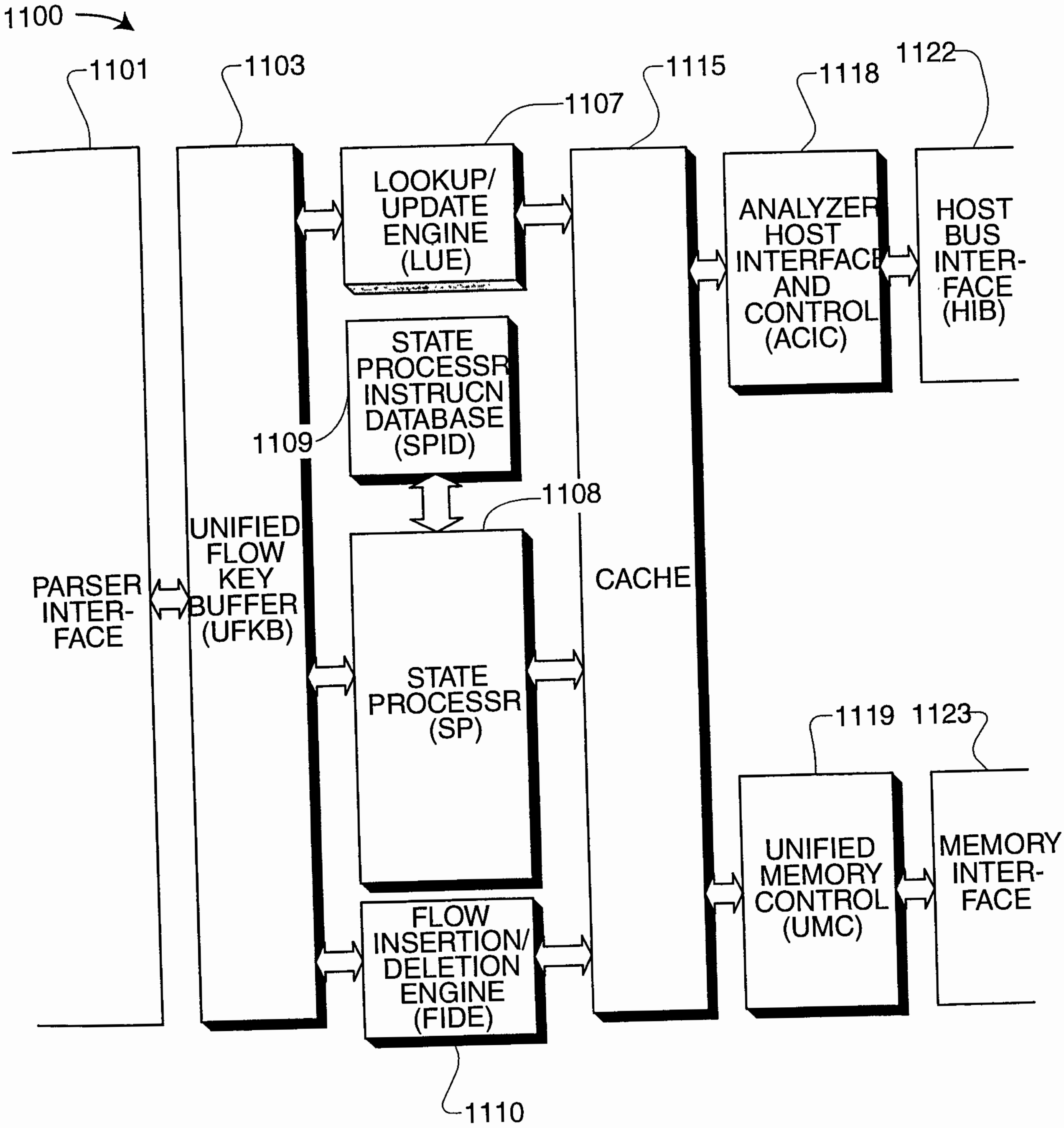

1100
1101
1103
1107
1115
1118
1122
LOOKUP/
UPDATE
ENGINE
(LUE)
ANALYZER
HOST
INTERFACE
AND
CONTROL
(ACIC)
HOST
BUS
INTER-
FACE
(HIB)
STATE
PROCESSR
INSTRUCN
DATABASE
(SPID)
1109
1108
PARSER
INTER-
FACE
UNIFIED
FLOW
KEY
BUFFER
(UFKB)
CACHE
STATE
PROCESSR
(SP)
1119
1123
UNIFIED
MEMORY
CONTROL
(UMC)
MEMORY
INTER-
FACE
FLOW
INSERTION/
DELETION
ENGINE
(FIDE)
1110

FIG. 11

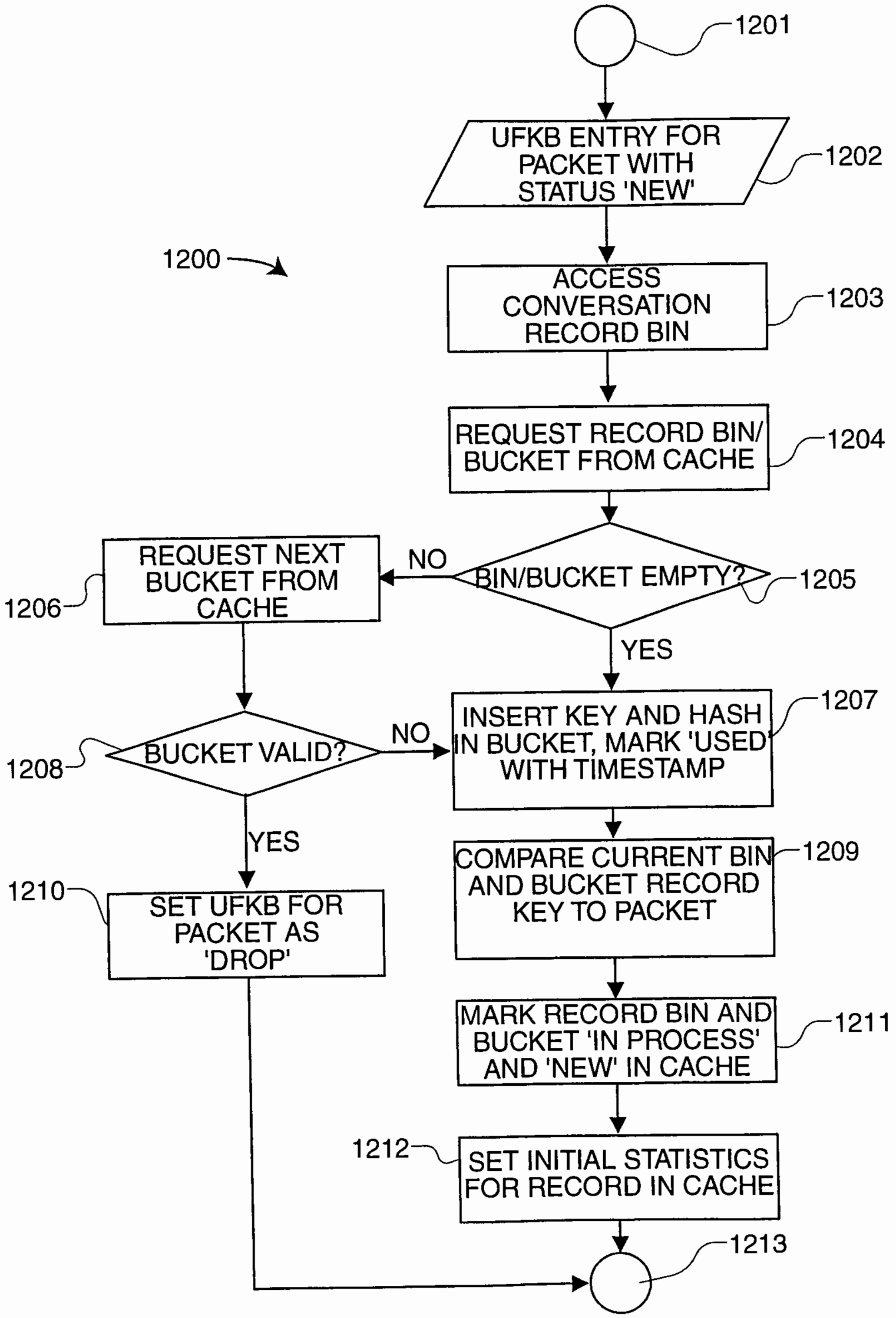

1201
UFKB ENTRY FOR PACKET WITH STATUS 'NEW'
1202
1200
ACCESS CONVERSATION RECORD BIN
1203
REQUEST RECORD BIN/ BUCKET FROM CACHE
1204
REQUEST NEXT BUCKET FROM CACHE
1206
NO
BIN/BUCKET EMPTY?
1205
YES
1207
INSERT KEY AND HASH IN BUCKET, MARK 'USED' WITH TIMESTAMP
BUCKET VALID?
1208
NO
YES
1210
SET UFKB FOR PACKET AS 'DROP'
COMPARE CURRENT BIN AND BUCKET RECORD KEY TO PACKET
1209
MARK RECORD BIN AND BUCKET 'IN PROCESS' AND 'NEW' IN CACHE
1211
1212
SET INITIAL STATISTICS FOR RECORD IN CACHE
1213

FIG. 12

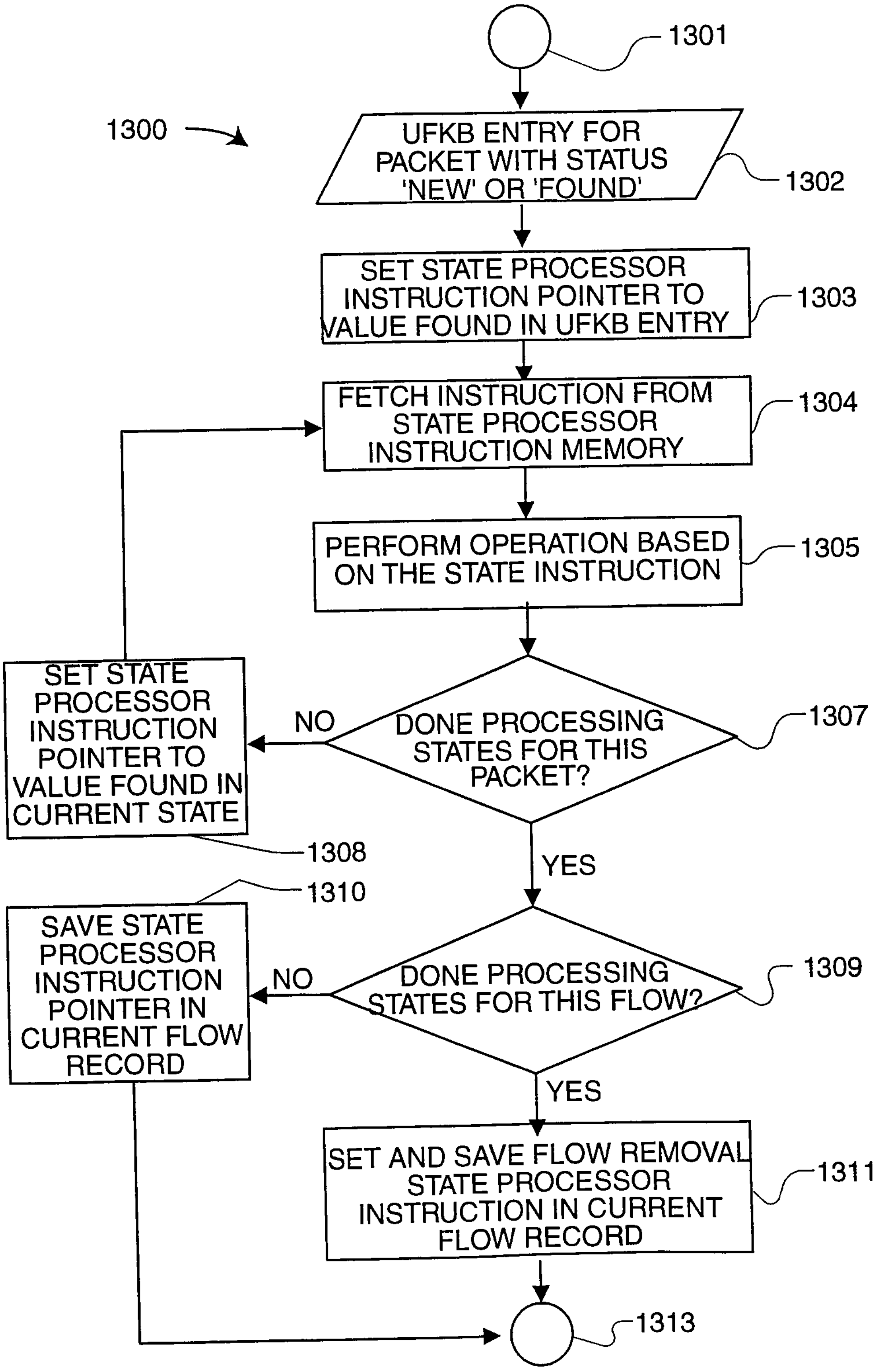

1300
1301
UFKB ENTRY FOR PACKET WITH STATUS 'NEW' OR 'FOUND'
1302
SET STATE PROCESSOR INSTRUCTION POINTER TO VALUE FOUND IN UFKB ENTRY
1303
FETCH INSTRUCTION FROM STATE PROCESSOR INSTRUCTION MEMORY
1304
PERFORM OPERATION BASED ON THE STATE INSTRUCTION
1305
DONE PROCESSING STATES FOR THIS PACKET?
1307
NO
SET STATE PROCESSOR INSTRUCTION POINTER TO VALUE FOUND IN CURRENT STATE
1308
YES
1310
DONE PROCESSING STATES FOR THIS FLOW?
1309
NO
SAVE STATE PROCESSOR INSTRUCTION POINTER IN CURRENT FLOW RECORD
YES
SET AND SAVE FLOW REMOVAL STATE PROCESSOR INSTRUCTION IN CURRENT FLOW RECORD
1311
1313

FIG. 13

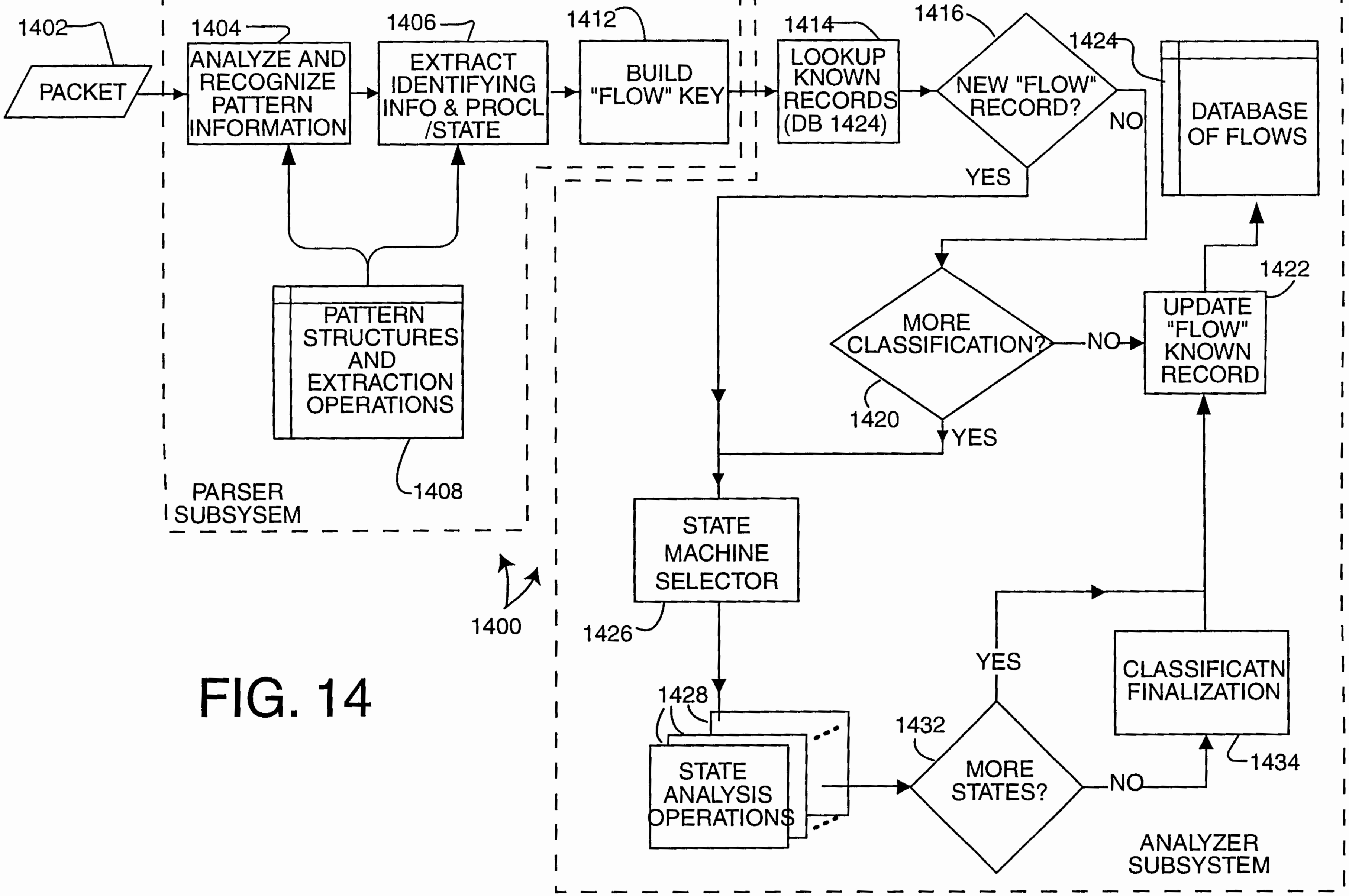

1402
PACKET
1404
ANALYZE AND RECOGNIZE PATTERN INFORMATION
1406
EXTRACT IDENTIFYING INFO & PROCL /STATE
1412
BUILD "FLOW" KEY
PATTERN STRUCTURES AND EXTRACTION OPERATIONS
1408
PARSER SUBSYSEM
1414
LOOKUP KNOWN RECORDS (DB 1424)
1416
NEW "FLOW" RECORD?
NO
YES
1424
DATABASE OF FLOWS
MORE CLASSIFICATION?
NO
1420
YES
1422
UPDATE "FLOW" KNOWN RECORD
STATE MACHINE SELECTOR
1426
1428
STATE ANALYSIS OPERATIONS
1432
MORE STATES?
YES
NO
CLASSIFICATN FINALIZATION
1434
ANALYZER SUBSYSTEM
1400

FIG. 14

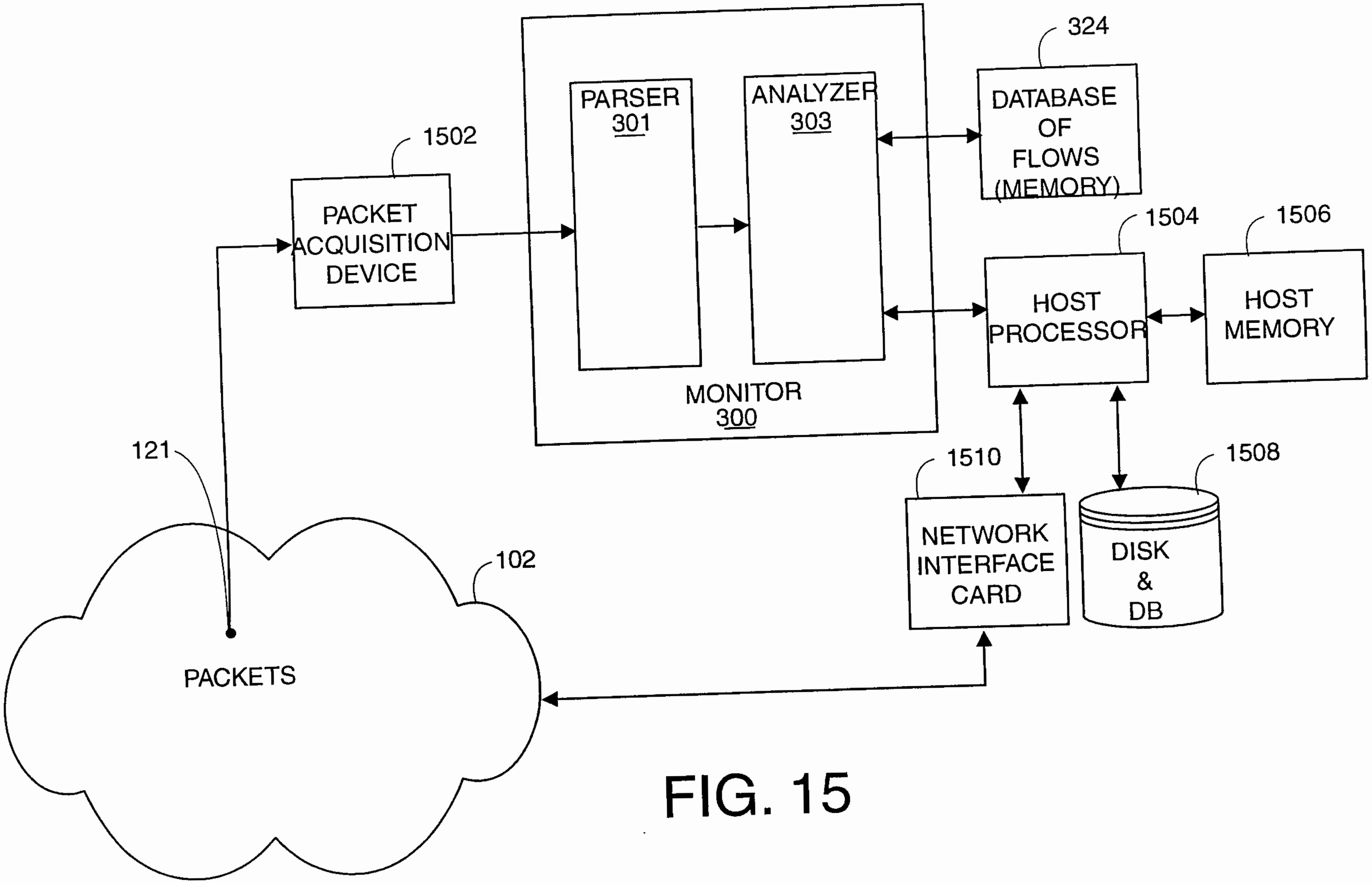

PACKET ACQUISITION DEVICE
1502
PARSER 301
ANALYZER 303
MONITOR 300
DATABASE OF FLOWS (MEMORY)
324
HOST PROCESSOR
1504
HOST MEMORY
1506
NETWORK INTERFACE CARD
1510
DISK & DB
1508
121
102
PACKETS

FIG. 15

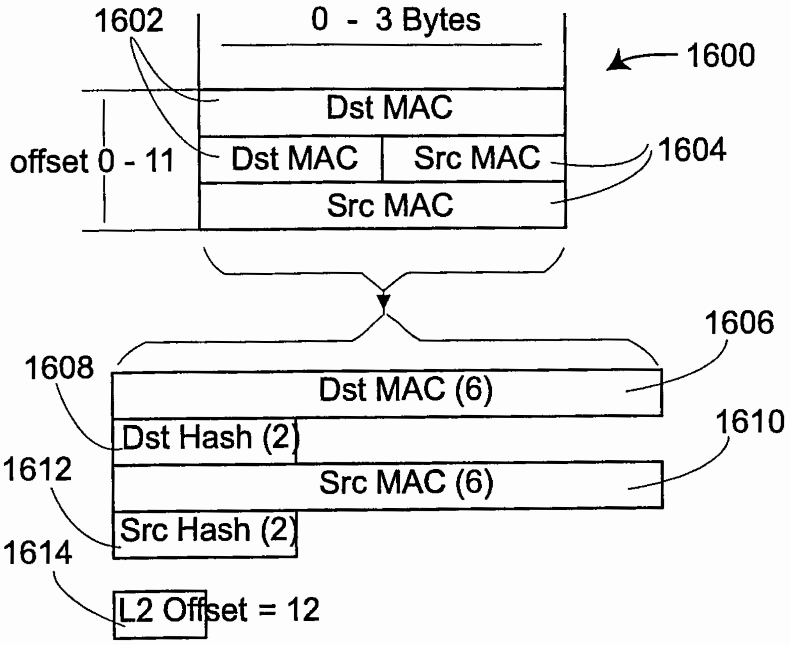

1602
0 - 3 Bytes
1600
Dst MAC
offset 0 - 11
Dst MAC
Src MAC
1604
Src MAC
1606
1608
Dst MAC (6)
Dst Hash (2)
1610
1612
Src MAC (6)
Src Hash (2)
1614
L2 Offset = 12

FIG. 16

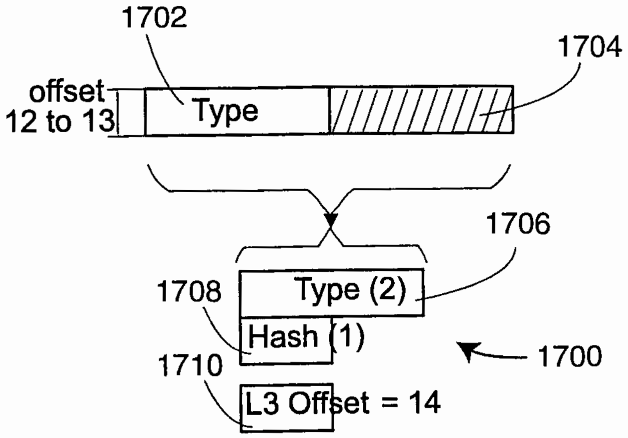


FIG. 17A

1712

IDP	= 0x0600*
IP	= 0x0800*
CHAOSNET	= 0x0804
ARP	= 0x0806
VIP	= 0x0BAD*
VLOOP	= 0x0BAE
VECHO	= 0x0BAF
NETBIOS-3COM	= 0x3C00 - 0x3C0D#
DEC-MOP	= 0x6001
DEC-RC	= 0x6002
DEC-DRP	= 0x6003*
DEC-LAT	= 0x6004
DEC-DIAG	= 0x6005
DEC-LAVC	= 0x6007
RARP	= 0x8035
ATALK	= 0x809B*
VLOOP	= 0x80C4
VECHO	= 0x80C5
SNA-TH	= 0x80D5*
ATALKARP	= 0x80F3
IPX	= 0x8137*
SNMP	= 0x814C#
IPv6	= 0x86DD*
LOOPBACK	= 0x9000
Apple	= 0x080007

* L3 Decoding
# L5 Decoding

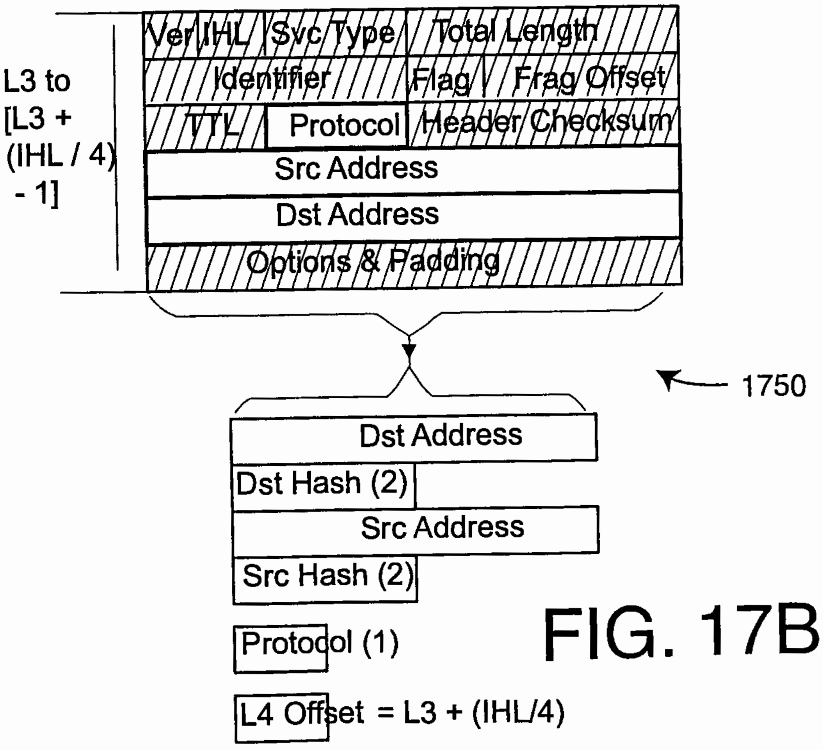


FIG. 17B

1752

ICMP	= 1
IGMP	= 2
GGP	= 3
TCP	= 6*
EGP	= 8
IGRP	= 9
PUP	= 12
CHAOS	= 16
UDP	= 17*
IDP	= 22#
ISO-TP4	= 29
DDP	= 37#
ISO-IP	= 80
VIP	= 83#
EIGRP	= 88
OSPF	= 89

* L4 Decoding
# L3 Re-Decoding

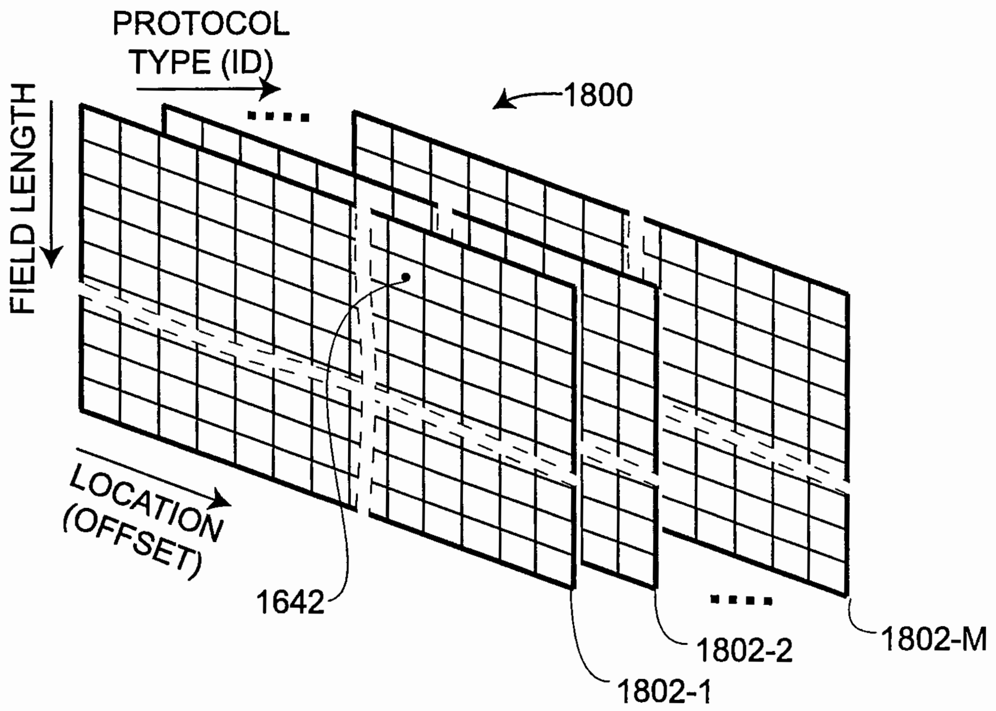


FIG. 18A

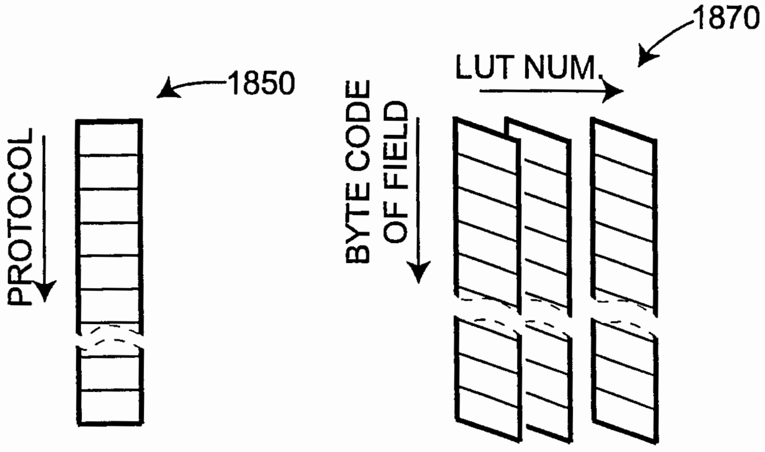


FIG. 18B

Our Ref./Docket No: APPT-001-3 Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz, *et al.* Title: RE-USING INFORMATION FROM DATA TRANSACTIONS FOR MAINTAINING STATISTICS IN NETWORK MONITORING	Group Art Unit: unassigned Examiner: unassigned

LETTER TO OFFICIAL DRAFTSPERSON SUBMISSION OF FORMAL DRAWINGS

The Assistant Commissioner for Patents
Washington, DC 20231
ATTN: Official Draftsperson

Dear Sir or Madam:

Attached please find 18 sheets of formal drawings to be made of record for the above identified patent application submitted herewith.

Respectfully Submitted,

June 30, 2000
Date

Dov Rosenfeld, Reg. No. 38687

Address for correspondence and attorney for applicant(s):
Dov Rosenfeld, Reg. No. 38,687
5507 College Avenue, Suite 2
Oakland, CA 94618
Telephone: (510) 547-3378; Fax: (510) 653-7992

Certificate of Mailing under 37 CFR 1.10

I hereby certify that this application and all attachments are being deposited with the United States Postal Service as Express Mail (Express Mail Label: EI417961927US in an envelope addressed to Box Patent Application, Assistant Commissioner for Patents, Washington, D.C. 20231 on.

Date: June 30, 2000

Signed:

Name: Dov Rosenfeld, Reg. No. 38687

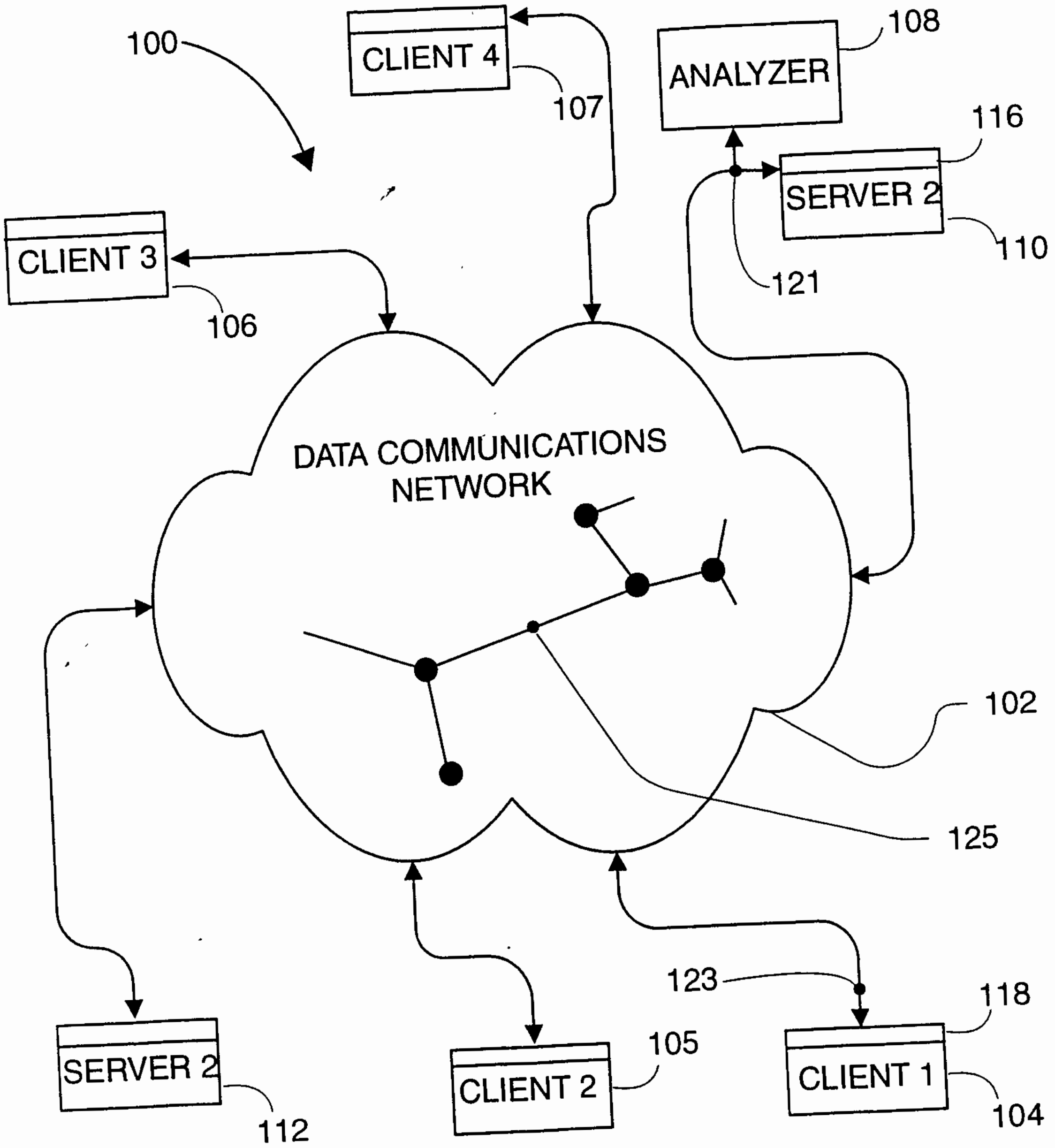

100
CLIENT 4
107
ANALYZER
108
116
SERVER 2
110
121
CLIENT 3
106
DATA COMMUNICATIONS NETWORK
102
125
123
118
SERVER 2
112
CLIENT 2
105
CLIENT 1
104

FIG. 1

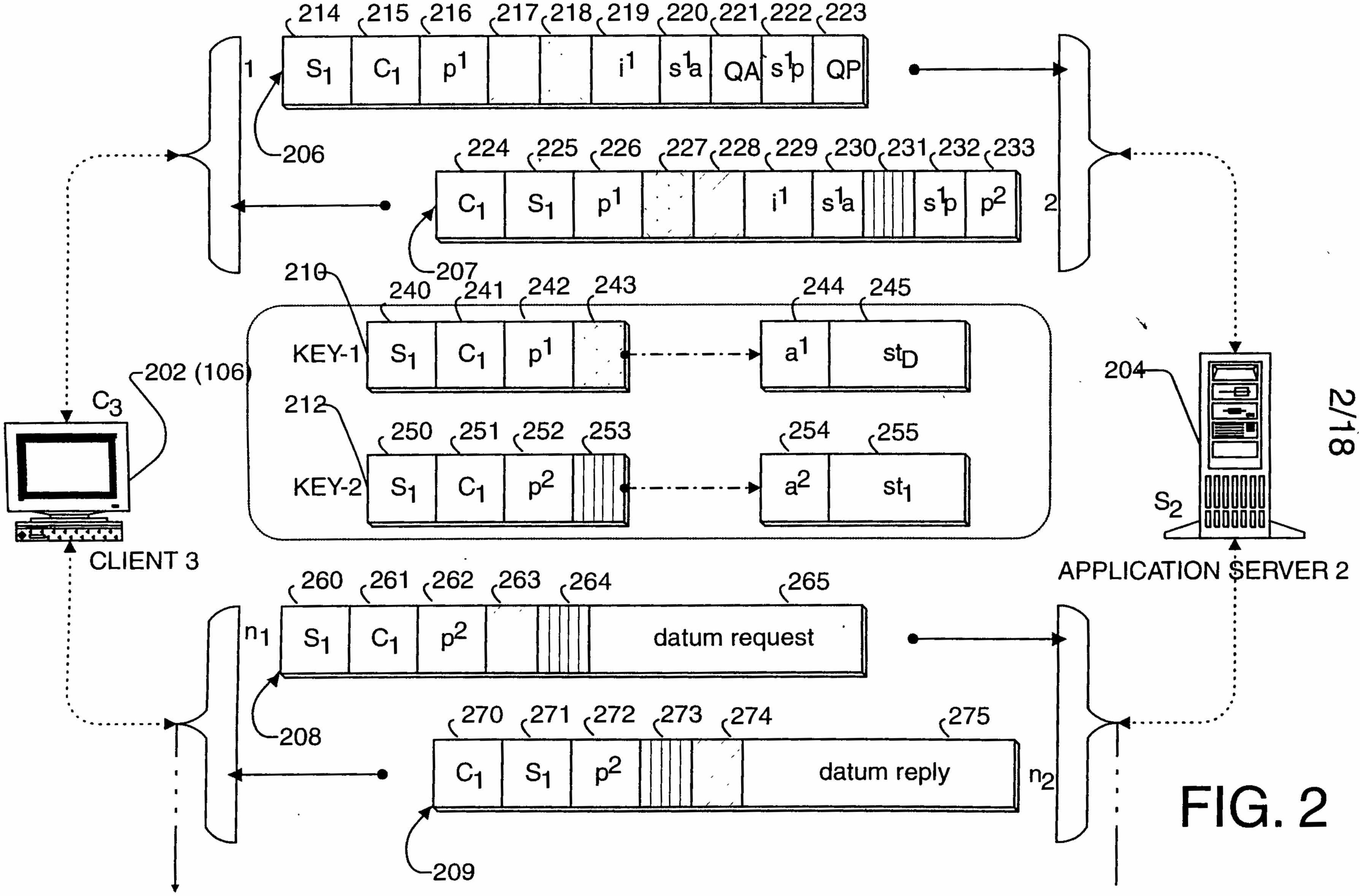

214
215
216
217
218
219
220
221
222
223
1
S_1
C_1
p^1
i^1
s^1a
QA
s^1p
QP
206
224
225
226
227
228
229
230
231
232
233
C_1
S_1
p^1
i^1
s^1a
s^1p
p^2
2
207
210
240
241
242
243
244
245
KEY-1
S_1
C_1
p^1
a^1
st_D
212
250
251
252
253
254
255
KEY-2
S_1
C_1
p^2
a^2
st_1
202 (106)
C_3
CLIENT 3
204
S_2
APPLICATION SERVER 2
260
261
262
263
264
265
n_1
S_1
C_1
p^2
datum request
208
270
271
272
273
274
275
C_1
S_1
p^2
datum reply
n_2
209

FIG. 2

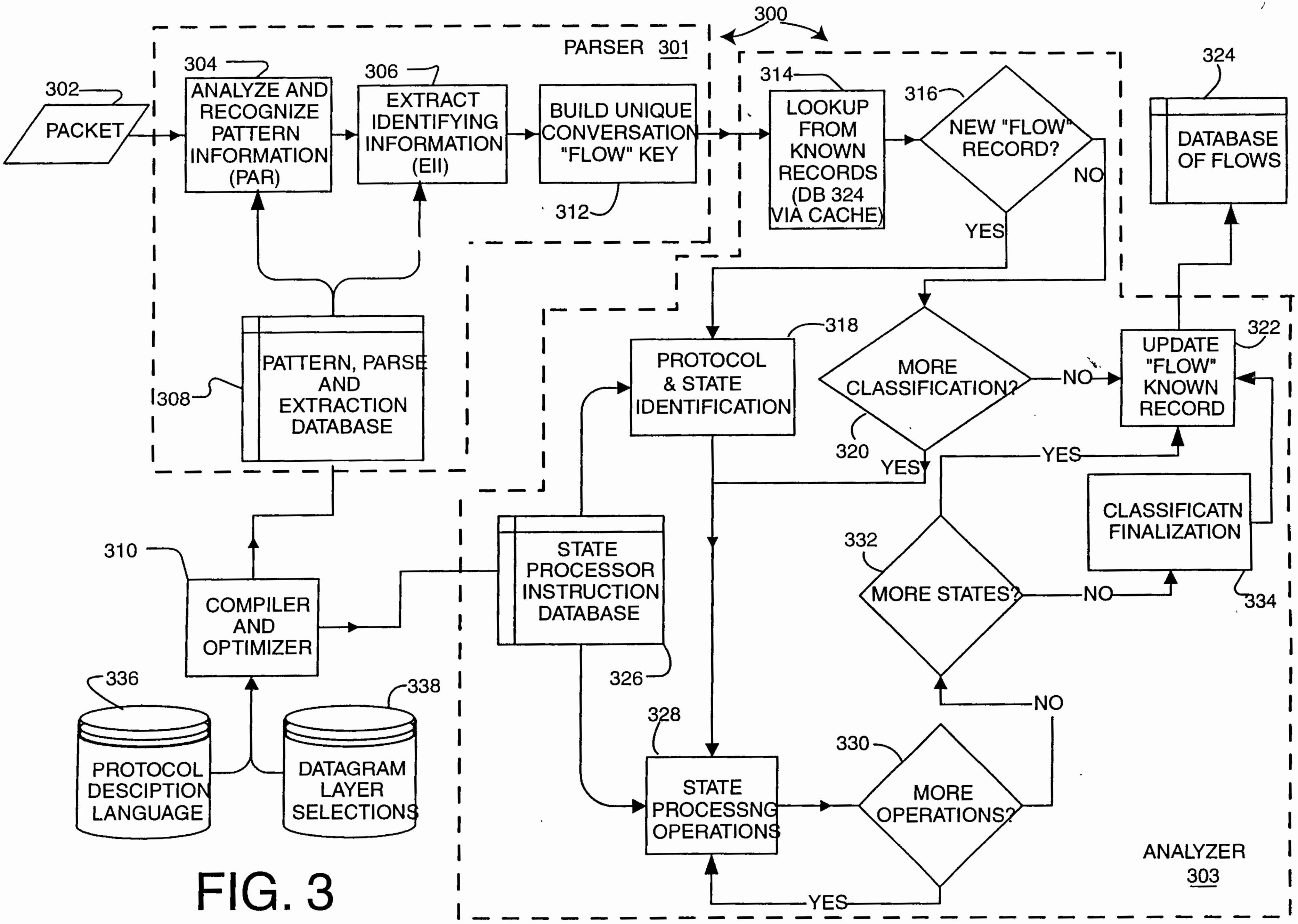

300
PARSER 301
302
PACKET
304
ANALYZE AND RECOGNIZE PATTERN INFORMATION (PAR)
306
EXTRACT IDENTIFYING INFORMATION (EII)
BUILD UNIQUE CONVERSATION "FLOW" KEY
312
308
PATTERN, PARSE AND EXTRACTION DATABASE
314
LOOKUP FROM KNOWN RECORDS (DB 324 VIA CACHE)
316
NEW "FLOW" RECORD?
NO
YES
324
DATABASE OF FLOWS
318
PROTOCOL & STATE IDENTIFICATION
MORE CLASSIFICATION?
NO
320
YES
322
UPDATE "FLOW" KNOWN RECORD
YES
CLASSIFICATN FINALIZATION
332
MORE STATES?
NO
334
310
COMPILER AND OPTIMIZER
STATE PROCESSOR INSTRUCTION DATABASE
326
336
PROTOCOL DESCIPTION LANGUAGE
338
DATAGRAM LAYER SELECTIONS
328
STATE PROCESSNG OPERATIONS
330
MORE OPERATIONS?
NO
YES
ANALYZER 303

FIG. 3

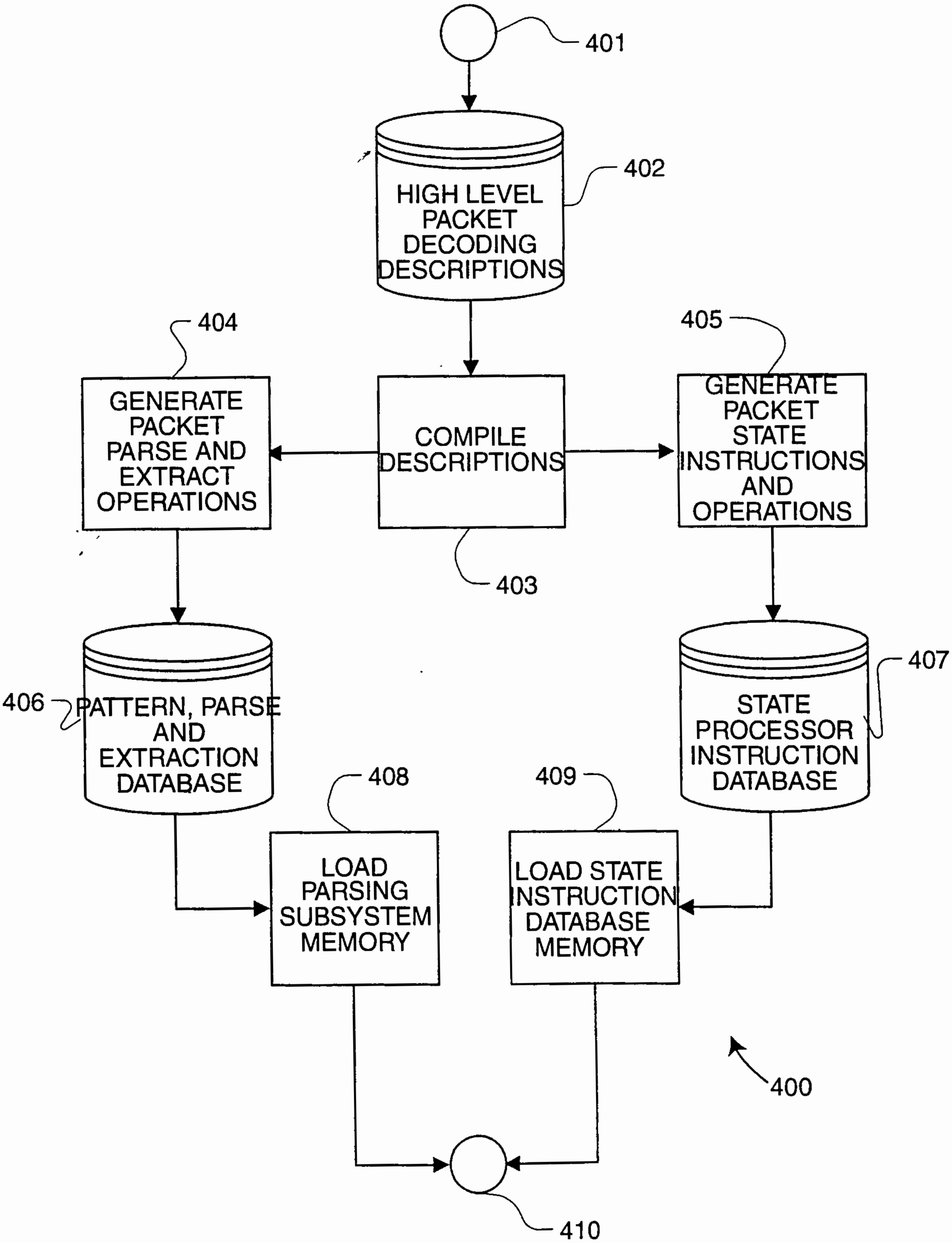

401
HIGH LEVEL PACKET DECODING DESCRIPTIONS
402
404
GENERATE PACKET PARSE AND EXTRACT OPERATIONS
COMPILE DESCRIPTIONS
403
405
GENERATE PACKET STATE INSTRUCTIONS AND OPERATIONS
406
PATTERN, PARSE AND EXTRACTION DATABASE
407
STATE PROCESSOR INSTRUCTION DATABASE
408
LOAD PARSING SUBSYSTEM MEMORY
409
LOAD STATE INSTRUCTION DATABASE MEMORY
400
410

FIG. 4

001-3

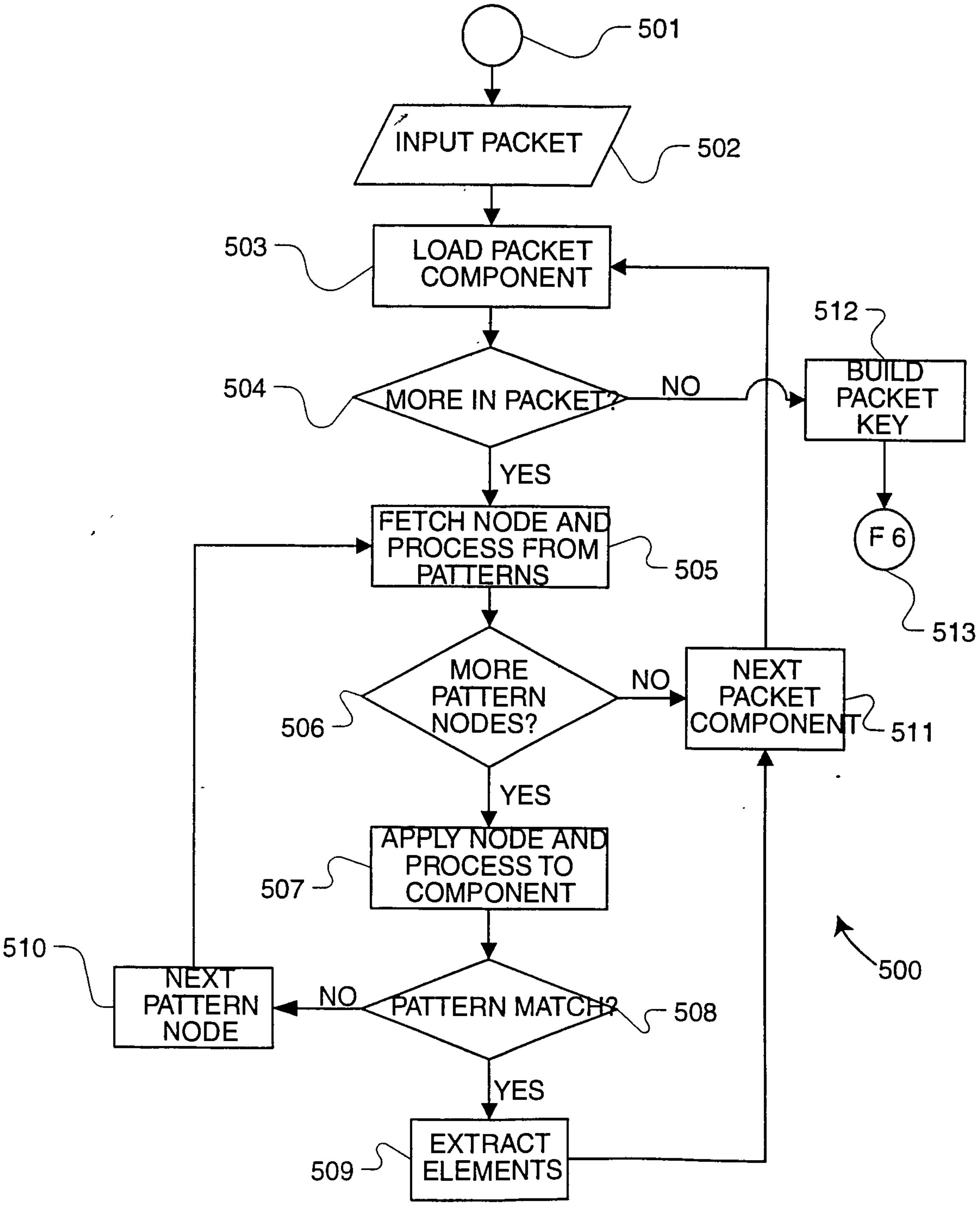

501
INPUT PACKET
502
503
LOAD PACKET COMPONENT
504
MORE IN PACKET?
NO
512
BUILD PACKET KEY
F 6
513
YES
FETCH NODE AND PROCESS FROM PATTERNS
505
MORE PATTERN NODES?
506
NO
NEXT PACKET COMPONENT
511
YES
APPLY NODE AND PROCESS TO COMPONENT
507
510
NEXT PATTERN NODE
NO
PATTERN MATCH?
508
500
YES
EXTRACT ELEMENTS
509

FIG. 5

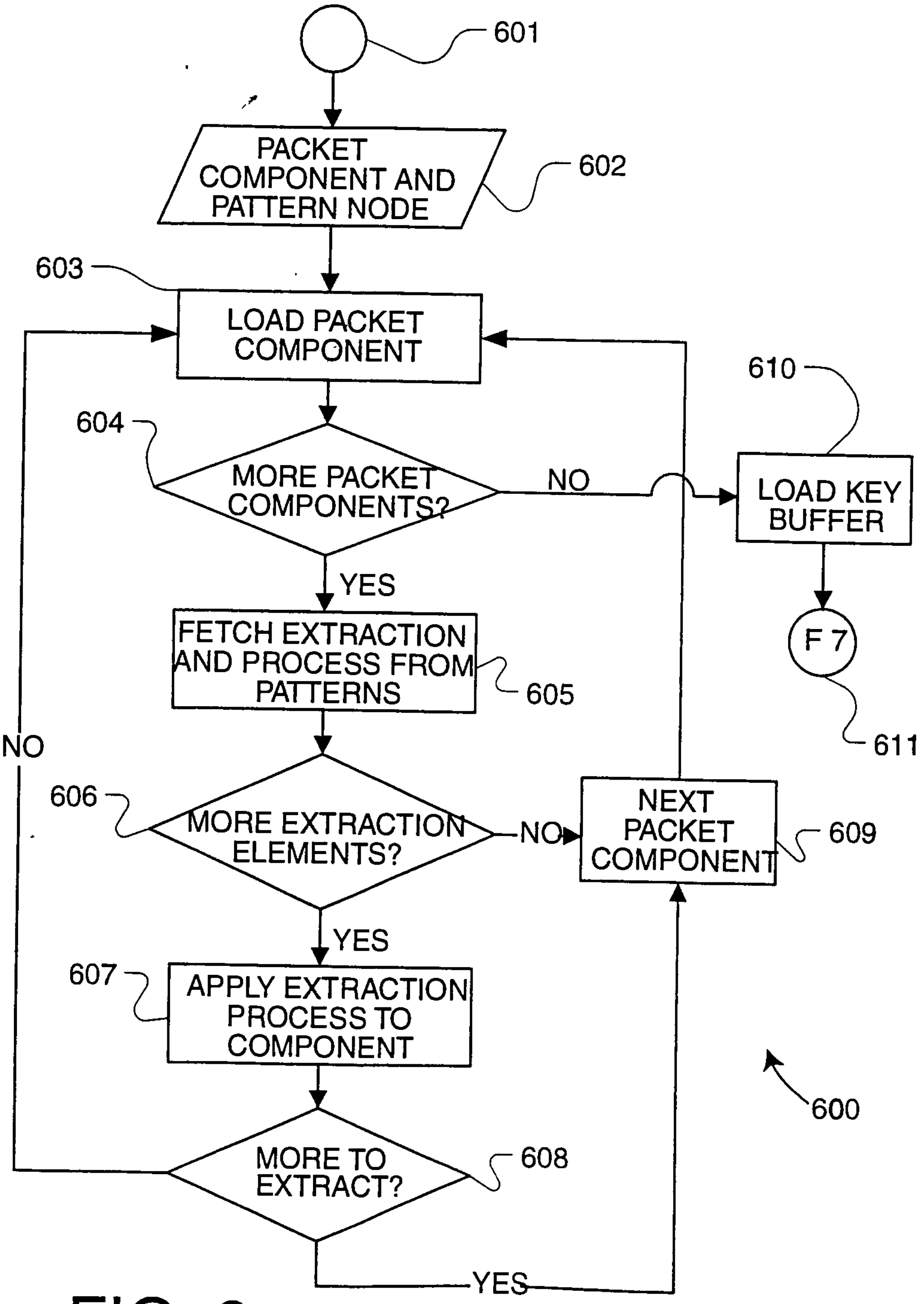

601
PACKET COMPONENT AND PATTERN NODE
602
603
LOAD PACKET COMPONENT
604
MORE PACKET COMPONENTS?
NO
610
LOAD KEY BUFFER
F 7
611
YES
FETCH EXTRACTION AND PROCESS FROM PATTERNS
605
NO
606
MORE EXTRACTION ELEMENTS?
NO
NEXT PACKET COMPONENT
609
YES
607
APPLY EXTRACTION PROCESS TO COMPONENT
600
MORE TO EXTRACT?
608
YES

FIG. 6

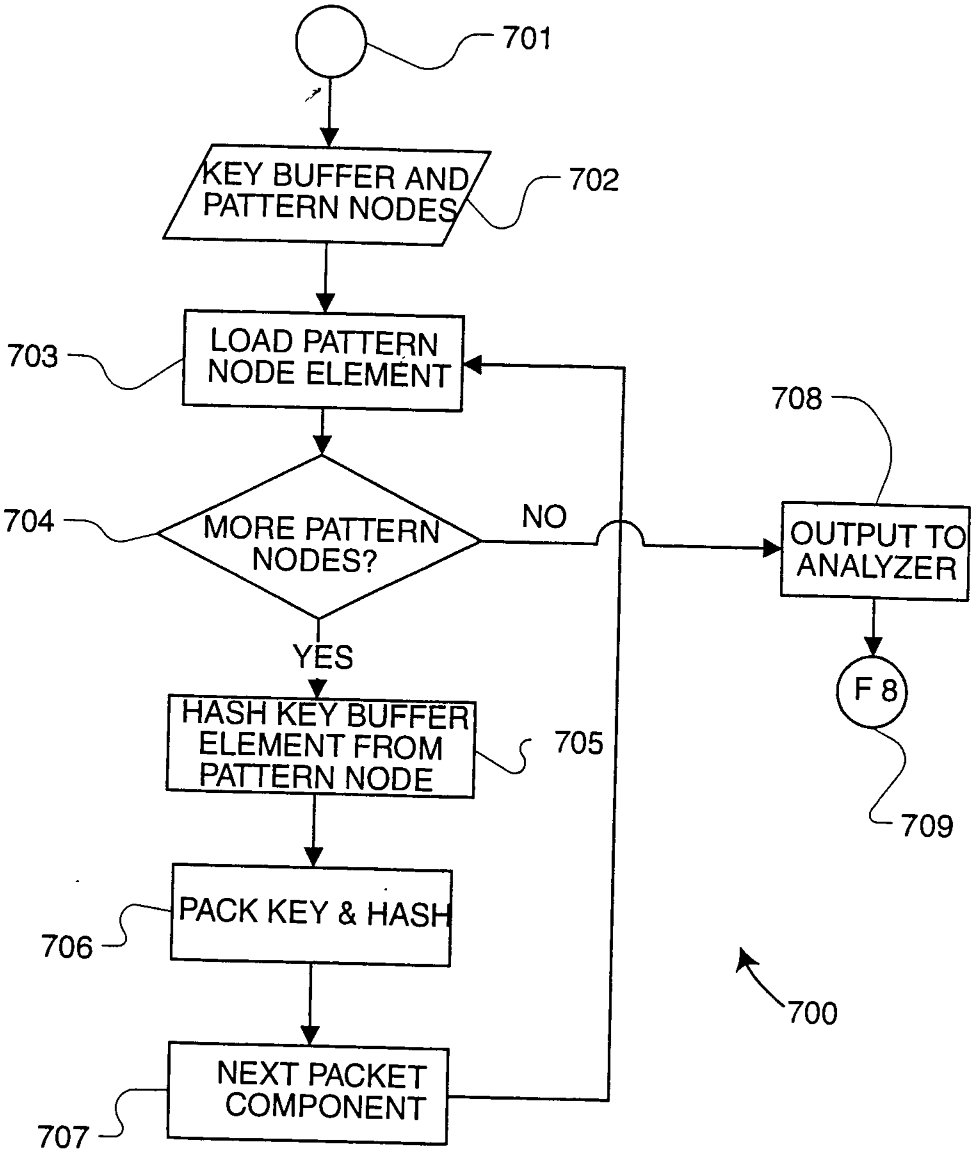

701
KEY BUFFER AND PATTERN NODES
702
LOAD PATTERN NODE ELEMENT
703
MORE PATTERN NODES?
704
NO
YES
708
OUTPUT TO ANALYZER
F 8
709
HASH KEY BUFFER ELEMENT FROM PATTERN NODE
705
PACK KEY & HASH
706
NEXT PACKET COMPONENT
707
700

FIG. 7

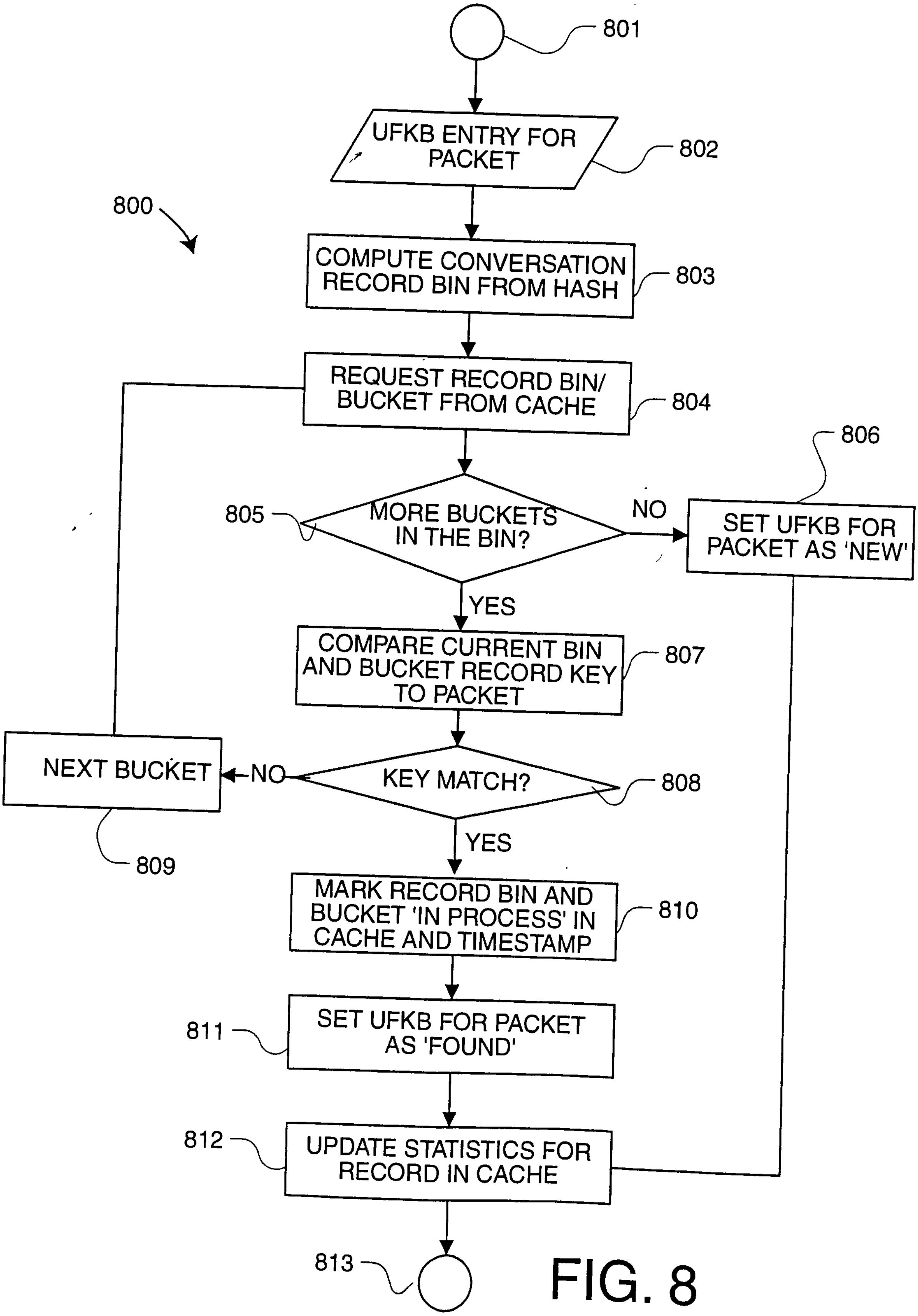

801
UFKB ENTRY FOR PACKET
802
800
COMPUTE CONVERSATION RECORD BIN FROM HASH
803
REQUEST RECORD BIN/ BUCKET FROM CACHE
804
806
805
MORE BUCKETS IN THE BIN?
NO
SET UFKB FOR PACKET AS 'NEW'
YES
COMPARE CURRENT BIN AND BUCKET RECORD KEY TO PACKET
807
NEXT BUCKET
NO
KEY MATCH?
808
YES
809
MARK RECORD BIN AND BUCKET 'IN PROCESS' IN CACHE AND TIMESTAMP
810
811
SET UFKB FOR PACKET AS 'FOUND'
812
UPDATE STATISTICS FOR RECORD IN CACHE
813

FIG. 8

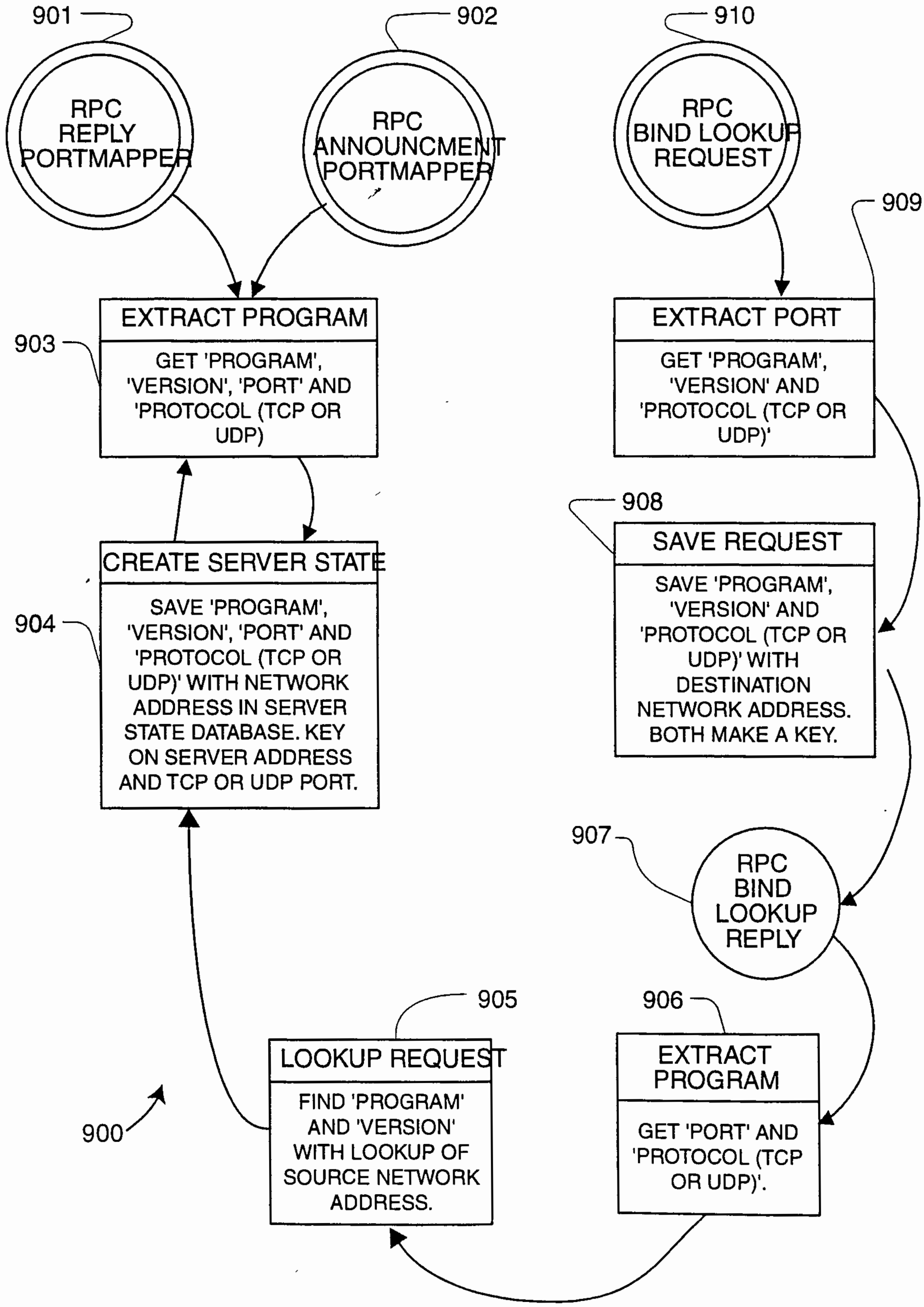

901
RPC REPLY PORTMAPPER
902
RPC ANNOUNCMENT PORTMAPPER
910
RPC BIND LOOKUP REQUEST
909
903
EXTRACT PROGRAM
GET 'PROGRAM', 'VERSION', 'PORT' AND 'PROTOCOL (TCP OR UDP)
EXTRACT PORT
GET 'PROGRAM', 'VERSION' AND 'PROTOCOL (TCP OR UDP)'
908
SAVE REQUEST
SAVE 'PROGRAM', 'VERSION' AND 'PROTOCOL (TCP OR UDP)' WITH DESTINATION NETWORK ADDRESS. BOTH MAKE A KEY.
904
CREATE SERVER STATE
SAVE 'PROGRAM', 'VERSION', 'PORT' AND 'PROTOCOL (TCP OR UDP)' WITH NETWORK ADDRESS IN SERVER STATE DATABASE. KEY ON SERVER ADDRESS AND TCP OR UDP PORT.
907
RPC BIND LOOKUP REPLY
905
906
LOOKUP REQUEST
FIND 'PROGRAM' AND 'VERSION' WITH LOOKUP OF SOURCE NETWORK ADDRESS.
EXTRACT PROGRAM
GET 'PORT' AND 'PROTOCOL (TCP OR UDP)'.
900

FIG. 9

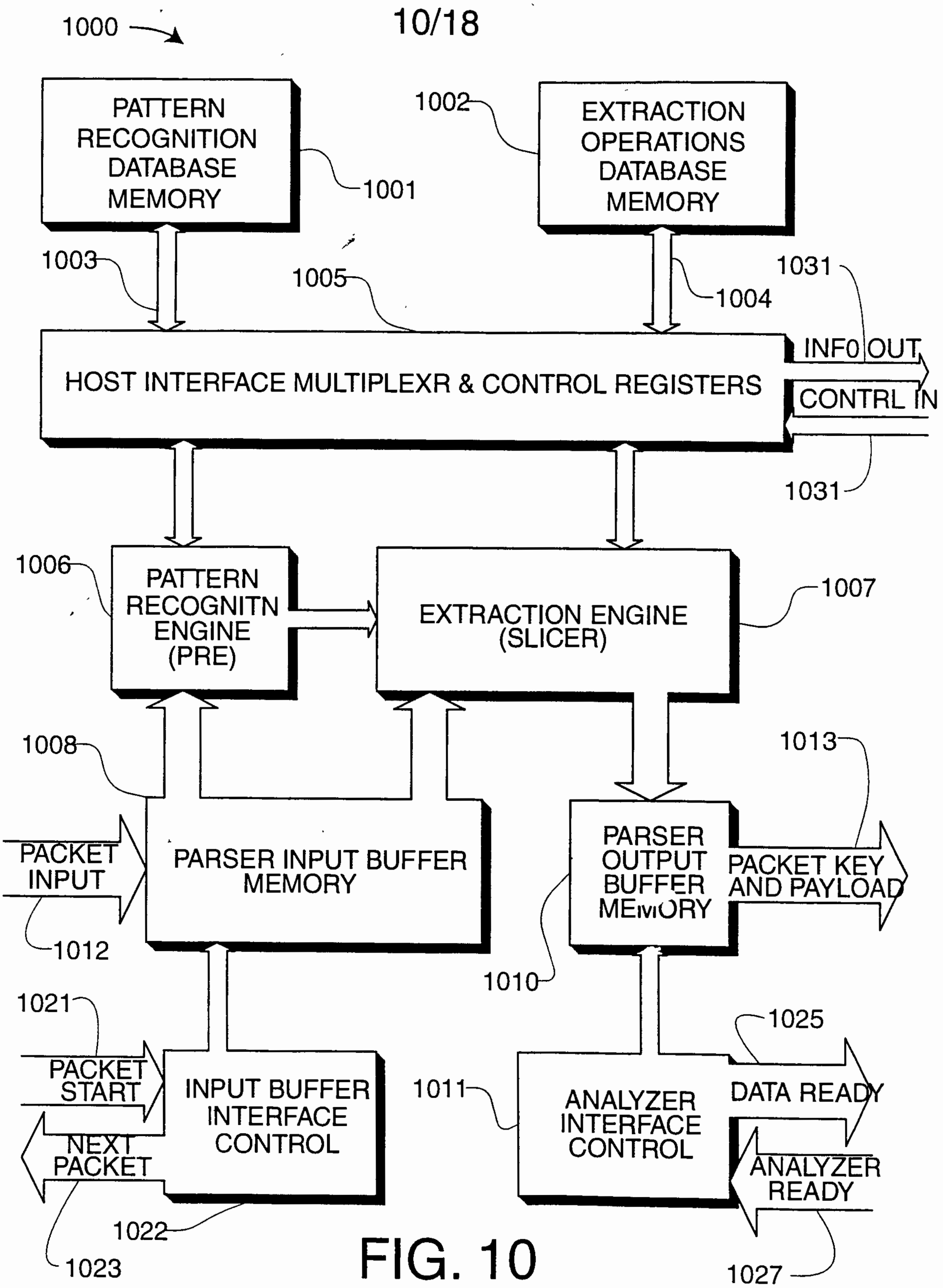

1000
PATTERN RECOGNITION DATABASE MEMORY
1001
1002
EXTRACTION OPERATIONS DATABASE MEMORY
1003
1005
1004
1031
INF0 OUT
CONTRL IN
HOST INTERFACE MULTIPLEXR & CONTROL REGISTERS
1031
1006
PATTERN RECOGNITN ENGINE (PRE)
EXTRACTION ENGINE (SLICER)
1007
1008
1013
PACKET INPUT
PARSER INPUT BUFFER MEMORY
PARSER OUTPUT BUFFER MEMORY
PACKET KEY AND PAYLOAD
1012
1010
1021
1025
PACKET START
INPUT BUFFER INTERFACE CONTROL
1011
ANALYZER INTERFACE CONTROL
DATA READY
NEXT PACKET
ANALYZER READY
1022
1023
1027

FIG. 10

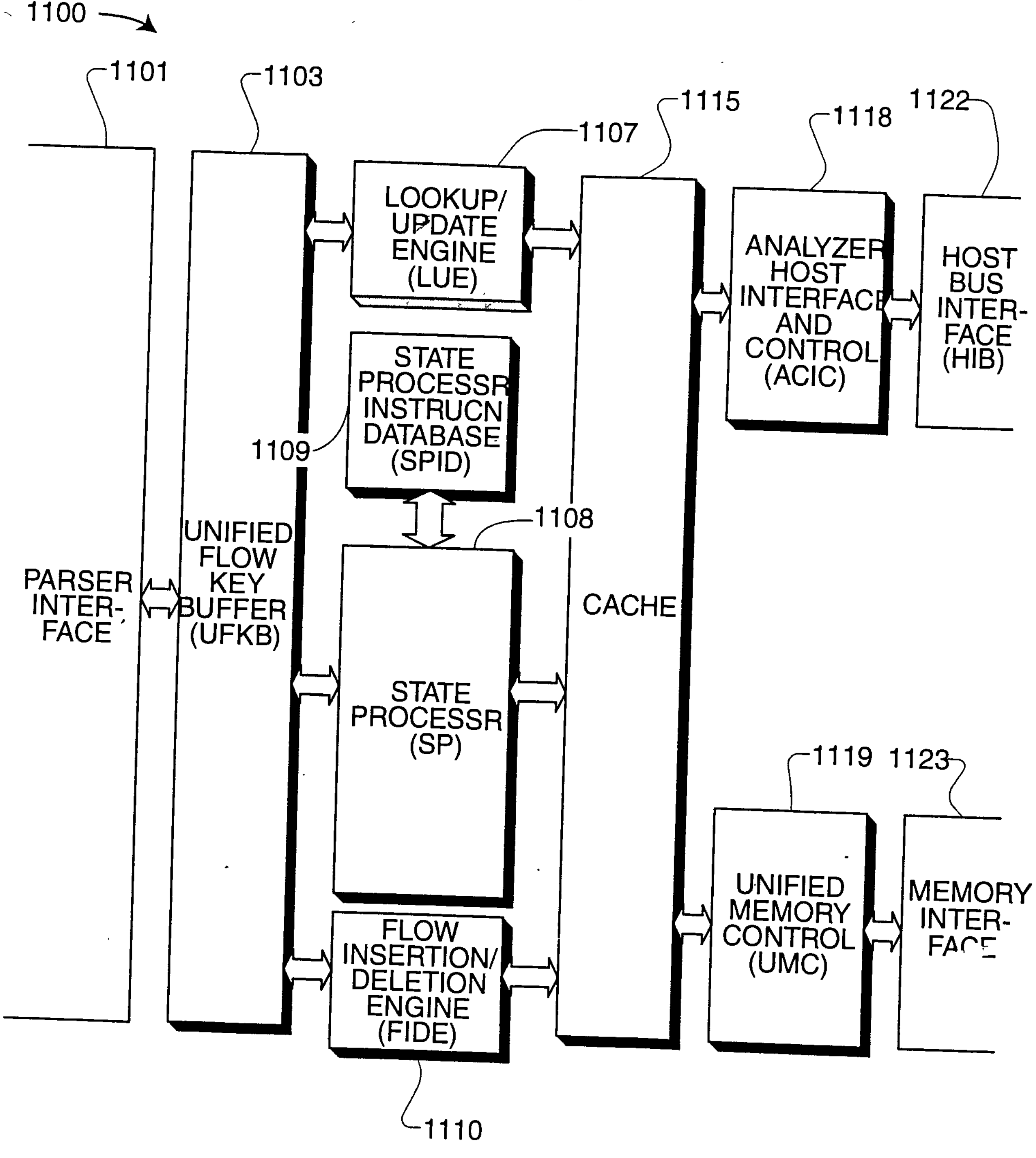
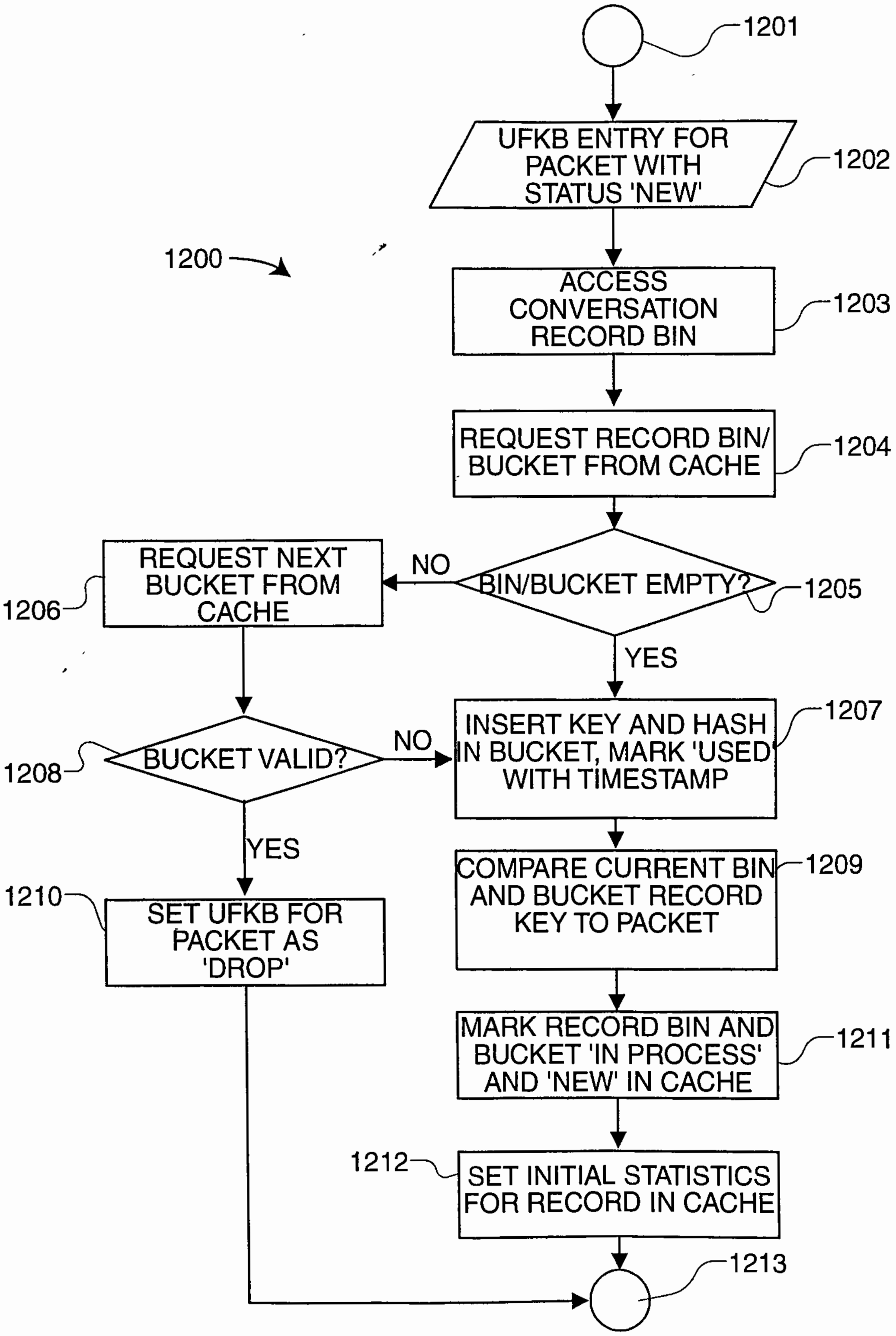

1100
1101
1103
1107
1115
1118
1122
LOOKUP/
UPDATE
ENGINE
(LUE)
ANALYZER
HOST
INTERFACE
AND
CONTROL
(ACIC)
HOST
BUS
INTER-
FACE
(HIB)
STATE
PROCESSR
INSTRUCN
DATABASE
(SPID)
1109
1108
PARSER
INTER-
FACE
UNIFIED
FLOW
KEY
BUFFER
(UFKB)
CACHE
STATE
PROCESSR
(SP)
1119
1123
UNIFIED
MEMORY
CONTROL
(UMC)
MEMORY
INTER-
FACE
FLOW
INSERTION/
DELETION
ENGINE
(FIDE)
1110

FIG. 11



1201
UFKB ENTRY FOR PACKET WITH STATUS 'NEW'
1202
1200
ACCESS CONVERSATION RECORD BIN
1203
REQUEST RECORD BIN/ BUCKET FROM CACHE
1204
REQUEST NEXT BUCKET FROM CACHE
1206
NO
BIN/BUCKET EMPTY?
1205
YES
BUCKET VALID?
1208
NO
INSERT KEY AND HASH IN BUCKET, MARK 'USED' WITH TIMESTAMP
1207
YES
1210
SET UFKB FOR PACKET AS 'DROP'
COMPARE CURRENT BIN AND BUCKET RECORD KEY TO PACKET
1209
MARK RECORD BIN AND BUCKET 'IN PROCESS' AND 'NEW' IN CACHE
1211
1212
SET INITIAL STATISTICS FOR RECORD IN CACHE
1213

FIG. 12

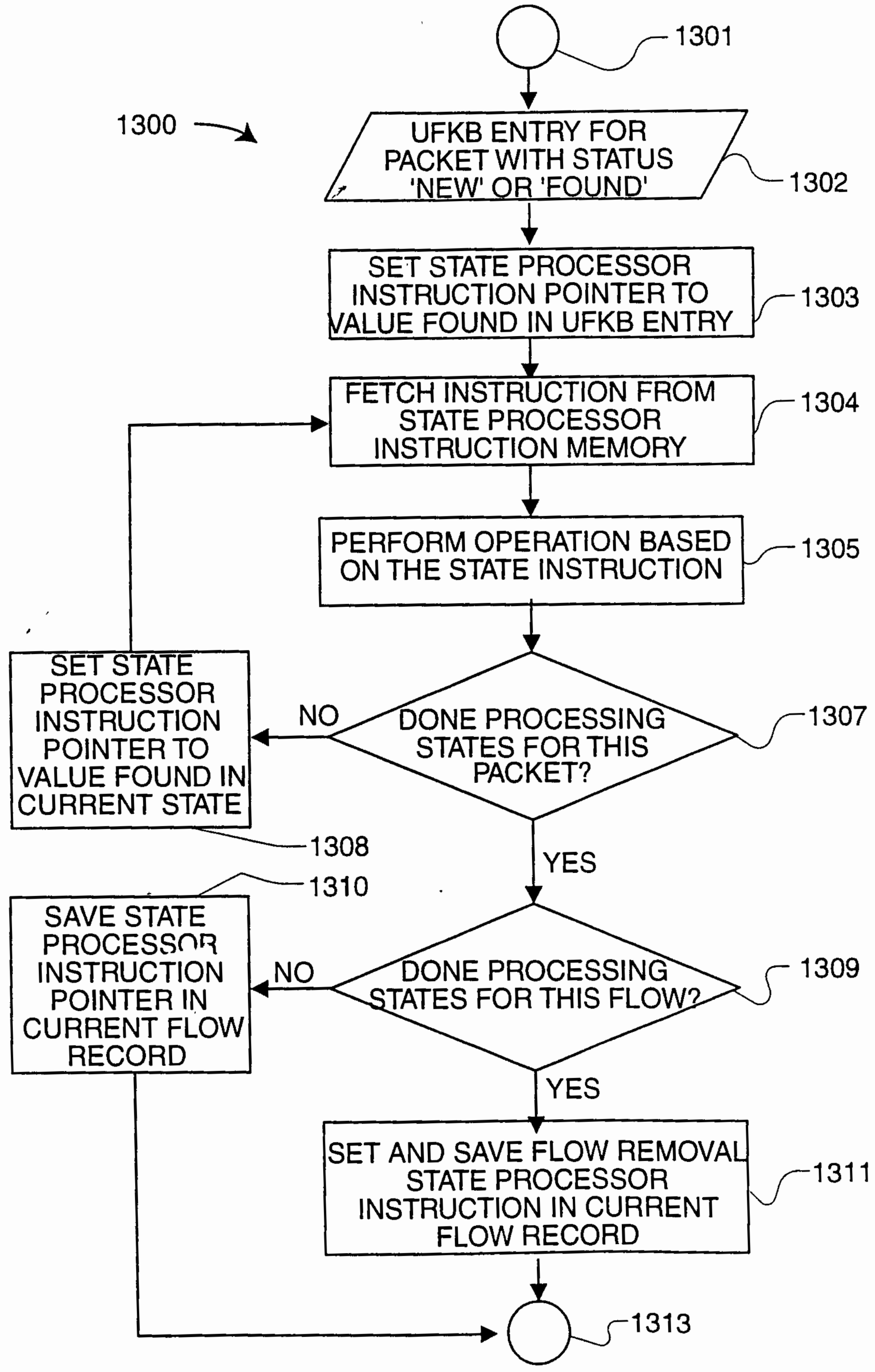

1301
1300
UFKB ENTRY FOR PACKET WITH STATUS 'NEW' OR 'FOUND'
1302
SET STATE PROCESSOR INSTRUCTION POINTER TO VALUE FOUND IN UFKB ENTRY
1303
FETCH INSTRUCTION FROM STATE PROCESSOR INSTRUCTION MEMORY
1304
PERFORM OPERATION BASED ON THE STATE INSTRUCTION
1305
SET STATE PROCESSOR INSTRUCTION POINTER TO VALUE FOUND IN CURRENT STATE
NO
DONE PROCESSING STATES FOR THIS PACKET?
1307
1308
YES
1310
SAVE STATE PROCESSOR INSTRUCTION POINTER IN CURRENT FLOW RECORD
NO
DONE PROCESSING STATES FOR THIS FLOW?
1309
YES
SET AND SAVE FLOW REMOVAL STATE PROCESSOR INSTRUCTION IN CURRENT FLOW RECORD
1311
1313

FIG. 13

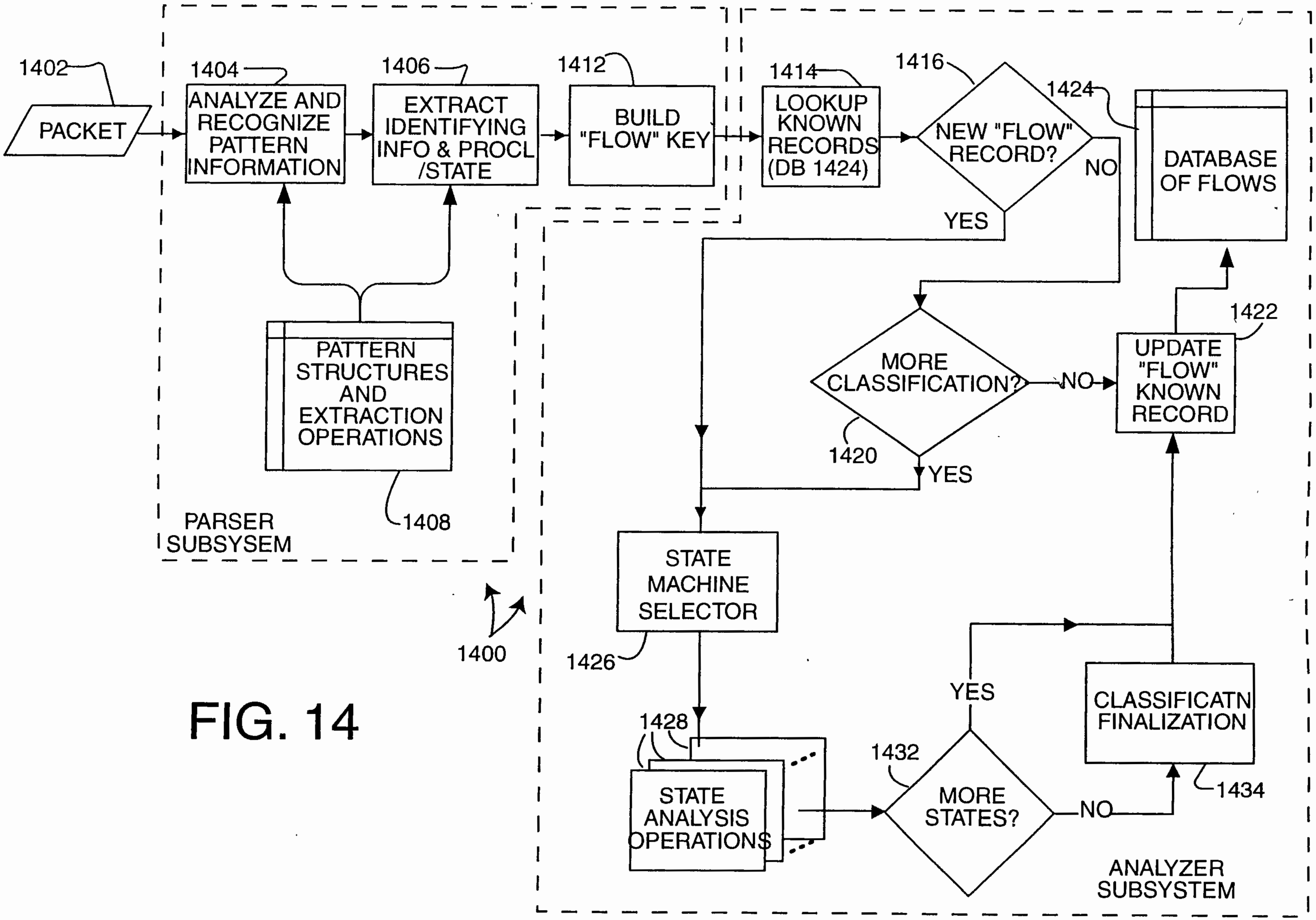

1402
PACKET
1404
ANALYZE AND RECOGNIZE PATTERN INFORMATION
1406
EXTRACT IDENTIFYING INFO & PROCL /STATE
1412
BUILD "FLOW" KEY
1414
LOOKUP KNOWN RECORDS (DB 1424)
1416
NEW "FLOW" RECORD?
NO
YES
1424
DATABASE OF FLOWS
PATTERN STRUCTURES AND EXTRACTION OPERATIONS
1408
PARSER SUBSYSEM
MORE CLASSIFICATION?
NO
YES
1420
1422
UPDATE "FLOW" KNOWN RECORD
1400
STATE MACHINE SELECTOR
1426
1428
STATE ANALYSIS OPERATIONS
1432
MORE STATES?
YES
NO
CLASSIFICATN FINALIZATION
1434
ANALYZER SUBSYSTEM

FIG. 14

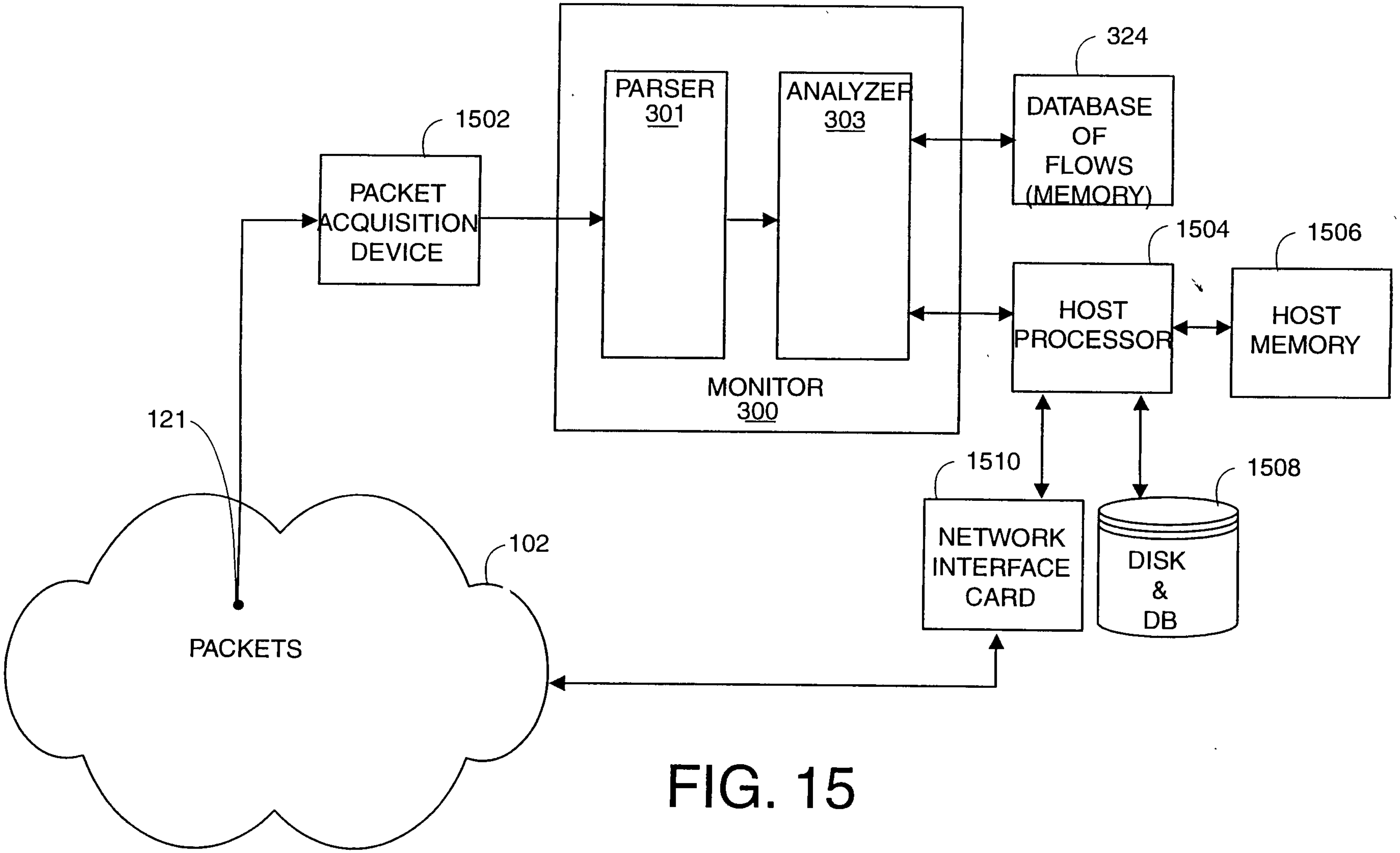

324
DATABASE OF FLOWS (MEMORY)
1502
PACKET ACQUISITION DEVICE
PARSER 301
ANALYZER 303
MONITOR 300
1504
HOST PROCESSOR
1506
HOST MEMORY
121
1510
NETWORK INTERFACE CARD
1508
DISK & DB
102
PACKETS

FIG. 15

001-3

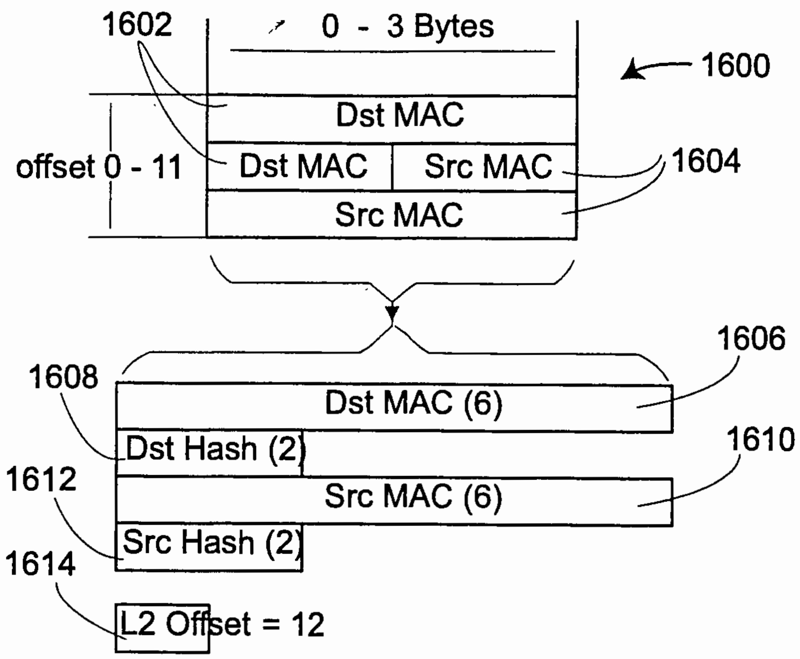

1602
0 - 3 Bytes
1600
Dst MAC
offset 0 - 11
Dst MAC
Src MAC
1604
Src MAC
1606
1608
Dst MAC (6)
Dst Hash (2)
1610
1612
Src MAC (6)
Src Hash (2)
1614
L2 Offset = 12

FIG. 16

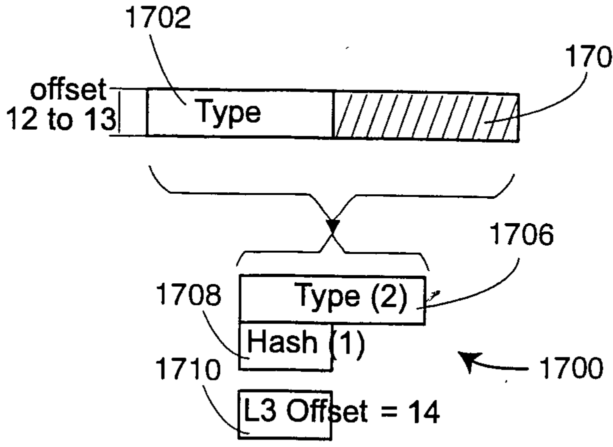


FIG. 17A

1712

IDP	= 0x0600*
IP	= 0x0800*
CHAOSNET	= 0x0804
ARP	= 0x0806
VIP	= 0x0BAD*
VLOOP	= 0x0BAE
VECHO	= 0x0BAF
NETBIOS-3COM	= 0x3C00 - 0x3C0D#
DEC-MOP	= 0x6001
DEC-RC	= 0x6002
DEC-DRP	= 0x6003*
DEC-LAT	= 0x6004
DEC-DIAG	= 0x6005
DEC-LAVC	= 0x6007
RARP	= 0x8035
ATALK	= 0x809B*
VLOOP	= 0x80C4
VECHO	= 0x80C5
SNA-TH	= 0x80D5*
ATALKARP	= 0x80F3
IPX	= 0x8137*
SNMP	= 0x814C#
IPv6	= 0x86DD*
LOOPBACK	= 0x9000
Apple	= 0x080007

* L3 Decoding
# L5 Decoding

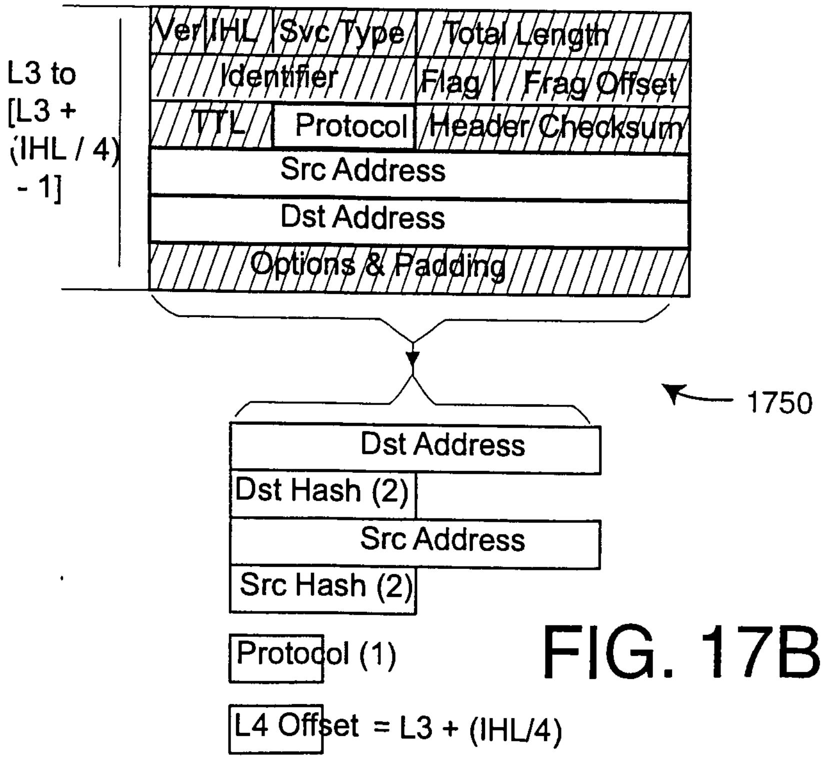


FIG. 17B

1752

ICMP	= 1
IGMP	= 2
GGP	= 3
TCP	= 6*
EGP	= 8
IGRP	= 9
PUP	= 12
CHAOS	= 16
UDP	= 17*
IDP	= 22#
ISO-TP4	= 29
DDP	= 37#
ISO-IP	= 80
VIP	= 83#
EIGRP	= 88
OSPF	= 89

* L4 Decoding
# L3 Re-Decoding

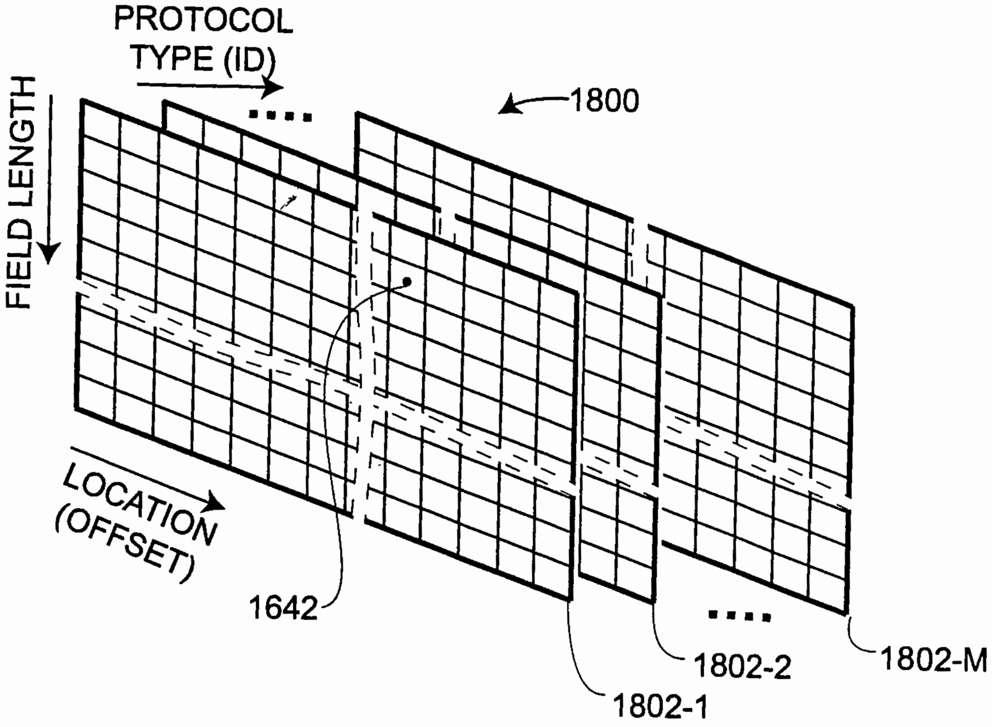


FIG. 18A

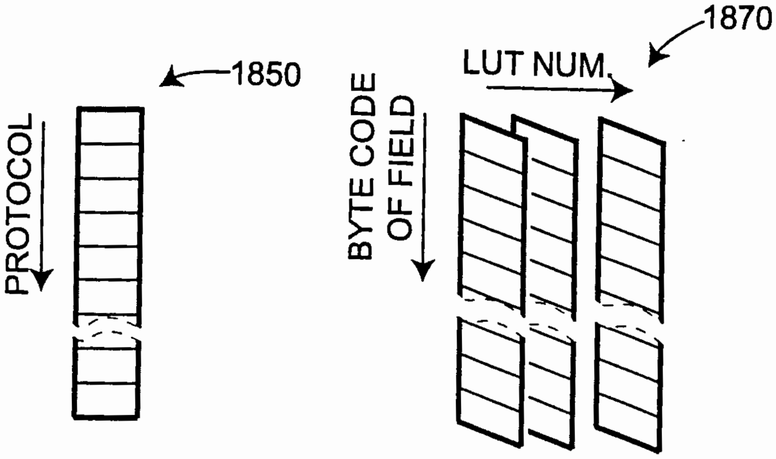


FIG. 18B

UNITED STATES PATENT AND TRADEMARK OFFICE

COMMISSIONER FOR PATENTS
UNITED STATES PATENT AND TRADEMARK OFFICE
WASHINGTON, D.C 20231
www.uspto.gov

APPLICATION NUMBER	FILING/RECEIPT DATE	FIRST NAMED APPLICANT	ATTORNEY DOCKET NUMBER
09/608,126	06/30/2000	Russell S. Dietz	APPT-001-3

FORMALITIES LETTER

OC000000005444855

Dov Rosenfeld
Suite 2
5507 College Avenue
Oakland, CA 94618

Date Mailed: 10/02/2000

NOTICE TO FILE MISSING PARTS OF NONPROVISIONAL APPLICATION

FILED UNDER 37 CFR 1.53(b)

Filing Date Granted

An application number and filing date have been accorded to this application. The item(s) indicated below, however, are missing. Applicant is given TWO MONTHS from the date of this Notice within which to file all required items and pay any fees required below to avoid abandonment. Extensions of time may be obtained by filing a petition accompanied by the extension fee under the provisions of 37 CFR 1.136(a).

- The statutory basic filing fee is missing.
  Applicant must submit **$ 690** *to complete the basic filing fee and/or file a small entity statement claiming such status (37 CFR 1.27).*
- Total additional claim fee(s) for this application is $18.
  - **$18** for **1** total claims over 20.
- The oath or declaration is missing.
  A properly signed oath or declaration in compliance with 37 CFR 1.63, identifying the application by the above Application Number and Filing Date, is required.
- To avoid abandonment, a late filing fee or oath or declaration surcharge as set forth in 37 CFR 1.16(e) of $130 for a non-small entity, must be submitted with the missing items identified in this letter.

- **The balance due by applicant is $ 838.**

A copy of this notice MUST be returned with the reply.

Customer Service Center
Initial Patent Examination Division (703) 308-1202

PART 3 - OFFICE COPY

SEC JDR

#3

Our Ref./Docket No: APPT-0C

Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz, *et al.*

Application No.: 09/608126

Filed: June 30, 2000

Title: RE-USING INFORMATION FROM DATA TRANSACTIONS FOR MAINTAINING STATISTICS IN NETWORK MONITORING

Group Art Unit: 2755

Examiner: (Unassigned)

RESPONSE TO NOTICE TO FILE MISSING PARTS OF APPLICATION

Assistant Commissioner for Patents
Washington, D.C. 20231
Attn: Box Missing Parts

Dear Assistant Commissioner:

This is in response to a Notice to File Missing Parts of Application under 37 CFR 1.53(f). Enclosed is a copy of said Notice and the following documents and fees to complete the filing requirements of the above-identified application:

_X_ Executed Declaration and Power of Attorney. The above-identified application is the same application which the inventor executed by signing the enclosed declaration;

_X_ Executed Assignment with assignment cover sheet.

_X_ A credit card payment form in the amount of $ 898.00 is attached, being for:

X	Statutory basic filing fee:	$ 710
X	Additional claim fee of	$ 18
X	Assignment recordation fee of	$ 40
	Extension Fee (1st Month) of	$ 110
X	Missing Parts Surcharge	$130

_X_ Applicant(s) believe(s) that no Extension of Time is required. However, this conditional petition is being made to provide for the possibility that applicant has inadvertently overlooked the need for a petition for an extension of time.

_____ Applicant(s) hereby petition(s) for an Extension of Time under 37 CFR 1.136(a) of:

_____ one months ($110) _____ two months ($380)

_____ two months ($870) _____ four months ($1360)

If an additional extension of time is required, please consider this as a petition therefor.

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 Assistant Commissioner for Patents, Washington, D.C. 20231 on.

Date: Nov 1 2000

Signed:

Name: Dov Rosenfeld, Reg. No. 38687

_X_ The Commissioner is hereby authorized to charge payment of any missing fees associated with this communication or credit any overpayment to Deposit Account No. 50-0292

(A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED):

Respectfully Submitted,

Nov 1, 2000
Date

Dov Rosenfeld, Reg. No. 38687

Address for correspondence:
Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618
Tel. (510) 547-3378; Fax: (510) 653-7992

PATENT APPLICATION

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION | **ATTORNEY DOCKET NO. APPT-001-3**

As a below named inventor, I hereby declare that:

My residence/post office address and citizenship are as stated below next to my name;

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

RE-USING INFORMATION FROM DATA TRANSACTIONS FOR MAINTAINING STATISTICS IN NETWORK MONITORING

the specification of which is attached hereto unless the following box is checked:

(X) was filed on June 30, 2000 as US Application Serial No. 09/608126 or PCT International Application Number _____ and was amended on ________ (if applicable).

I hereby state that I have reviewed and understood the contents of the above-identified specification, including the claims, as amended by any amendment(s) referred to above. I acknowledge the duty to disclose all information which is material to patentability as defined in 37 CFR 1.56.

Foreign Application(s) and/or Claim of Foreign Priority

I hereby claim foreign priority benefits under Title 35, United States Code Section 119 of any foreign application(s) for patent or inventor(s) certificate listed below and have also identified below any foreign application for patent or inventor(s) certificate having a filing date before that of the application on which priority is claimed:

COUNTRY	APPLICATION NUMBER	DATE FILED	PRIORITY CLAIMED UNDER 35
			YES: ____ NO: ____
			YES: ____ NO: ____

Provisional Application

I hereby claim the benefit under Title 35, United States Code Section 119(e) of any United States provisional application(s) listed below:

APPLICATION SERIAL NUMBER	FILING DATE
60/141,903	June 30, 1999

U.S. Priority Claim

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

APPLICATION SERIAL NUMBER	FILING DATE	STATUS(patented/pending/abandoned)

POWER OF ATTORNEY:

As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) listed below to prosecute this application and transact all business in the Patent and Trademark Office connected therewith:

Dov Rosenfeld, Reg. No. 38,687

Send Correspondence to:
Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618

Direct Telephone Calls To:
Dov Rosenfeld, Reg. No. 38,687
Tel: (510) 547-3378

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.

Name of First Inventor: Russell S. Dietz **Citizenship:** USA

Residence: 6146 Ostenberg Drive, San Jose, CA 95120-2736

Post Office Address: Same

First Inventor's Signature

10/9/00
Date

Declaration and Power of Attorney (Continued)
Case No; «Case CaseNumber»
APPT-001-3
Page 2

ADDITIONAL INVENTOR SIGNATURES:

Name of Second Inventor: Joseph R. Maixner **Citizenship:** USA

Residence: 121 Driftwood Court, Aptos, CA 95003

Post Office Address: Same

______________________________ ______________________________

Inventor's Signature **Date**

Name of Third Inventor: Andrew A. Koppenhaver **Citizenship:** USA

Residence: 10400 Kenmore Drive, Fairfax, VA 22030

Post Office Address: Same

______________________________ ______________________________

Inventor's Signature **Date**

11

PATENT APPLICATION

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION — **ATTORNEY DOCKET NO. APPT-001-3**

As a below named inventor, I hereby declare that:

My residence/post office address and citizenship are as stated below next to my name;

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

RE-USING INFORMATION FROM DATA TRANSACTIONS FOR MAINTAINING STATISTICS IN NETWORK MONITORING

the specification of which is attached hereto unless the following box is checked:

(X) was filed on June 30, 2000 as US Application Serial No. 09/608126 or PCT International Application Number _____ and was amended on ________ (if applicable).

I hereby state that I have reviewed and understood the contents of the above-identified specification, including the claims, as amended by any amendment(s) referred to above. I acknowledge the duty to disclose all information which is material to patentability as defined in 37 CFR 1.56.

Foreign Application(s) and/or Claim of Foreign Priority

I hereby claim foreign priority benefits under Title 35, United States Code Section 119 of any foreign application(s) for patent or inventor(s) certificate listed below and have also identified below any foreign application for patent or inventor(s) certificate having a filing date before that of the application on which priority is claimed:

COUNTRY	APPLICATION NUMBER	DATE FILED	PRIORITY CLAIMED UNDER 35
			YES: ____ NO: ____
			YES: ____ NO: ____

Provisional Application

I hereby claim the benefit under Title 35, United States Code Section 119(e) of any United States provisional application(s) listed below:

APPLICATION SERIAL NUMBER	FILING DATE
60/141,903	June 30, 1999

U.S. Priority Claim

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

APPLICATION SERIAL NUMBER	FILING DATE	STATUS(patented/pending/abandoned)

POWER OF ATTORNEY:

As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) listed below to prosecute this application and transact all business in the Patent and Trademark Office connected therewith:

Dov Rosenfeld, Reg. No. 38,687

Send Correspondence to:	**Direct Telephone Calls To:**
Dov Rosenfeld 5507 College Avenue, Suite 2 Oakland, CA 94618	Dov Rosenfeld, Reg. No. 38,687 Tel: (510) 547-3378

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.

Name of First Inventor: Russell S. Dietz **Citizenship:** USA

Residence: 6146 Ostenberg Drive, San Jose, CA 95120-2736

Post Office Address: Same

______________________ ______________________

First Inventor's Signature **Date**

PATENT APPLICATION

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

ATTORNEY DOCKET NO. APPT-001-3

As a below named inventor, I hereby declare that:

My residence/post office address and citizenship are as stated below next to my name;

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

RE-USING INFORMATION FROM DATA TRANSACTIONS FOR MAINTAINING STATISTICS IN NETWORK MONITORING

the specification of which is attached hereto unless the following box is checked:

(X) was filed on June 30, 2000 as US Application Serial No. 09/608126 or PCT International Application Number _____ and was amended on ________ (if applicable).

I hereby state that I have reviewed and understood the contents of the above-identified specification, including the claims, as amended by any amendment(s) referred to above. I acknowledge the duty to disclose all information which is material to patentability as defined in 37 CFR 1.56.

Foreign Application(s) and/or Claim of Foreign Priority

I hereby claim foreign priority benefits under Title 35, United States Code Section 119 of any foreign application(s) for patent or inventor(s) certificate listed below and have also identified below any foreign application for patent or inventor(s) certificate having a filing date before that of the application on which priority is claimed:

COUNTRY	APPLICATION NUMBER	DATE FILED	PRIORITY CLAIMED UNDER 35
			YES: NO:
			YES: NO:

Provisional Application

I hereby claim the benefit under Title 35, United States Code Section 119(e) of any United States provisional application(s) listed below:

APPLICATION SERIAL NUMBER	FILING DATE
60/141,903	June 30, 1999

U.S. Priority Claim

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

APPLICATION SERIAL NUMBER	FILING DATE	STATUS(patented/pending/abandoned)

POWER OF ATTORNEY:

As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) listed below to prosecute this application and transact all business in the Patent and Trademark Office connected therewith:

Dov Rosenfeld, Reg. No. 38,687

Send Correspondence to:
Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618

Direct Telephone Calls To:
Dov Rosenfeld, Reg. No. 38,687
Tel: (510) 547-3378

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.

Name of First Inventor: Russell S. Dietz **Citizenship:** USA

Residence: 6146 Ostenberg Drive, San Jose, CA 95120-2736

Post Office Address: Same

______________________________ ______________________
First Inventor's Signature **Date**

ADDITIONAL INVENTOR SIGNATURES:

Name of Second Inventor: Joseph R. Maixner **Citizenship:** USA

Residence: 121 Driftwood Court, Aptos, CA 95003

Post Office Address: Same

Joseph R. M

Inventor's Signature

10/23/2000

Date

Name of Third Inventor: Andrew A. Koppenhaver **Citizenship:** USA

Residence: 10400 Kenmore Drive, Fairfax, VA 22030

Post Office Address: Same

Inventor's Signature

Date

PATENT APPLICATION

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION **ATTORNEY DOCKET NO. APPT-001-3**

As a below named inventor, I hereby declare that:

My residence/post office address and citizenship are as stated below next to my name;

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

RE-USING INFORMATION FROM DATA TRANSACTIONS FOR MAINTAINING STATISTICS IN NETWORK MONITORING

the specification of which is attached hereto unless the following box is checked:

(X) was filed on June 30, 2000 as US Application Serial No. 09/608126 or PCT International Application Number _____ and was amended on _________ (if applicable).

I hereby state that I have reviewed and understood the contents of the above-identified specification, including the claims, as amended by any amendment(s) referred to above. I acknowledge the duty to disclose all information which is material to patentability as defined in 37 CFR 1.56.

Foreign Application(s) and/or Claim of Foreign Priority

I hereby claim foreign priority benefits under Title 35, United States Code Section 119 of any foreign application(s) for patent or inventor(s) certificate listed below and have also identified below any foreign application for patent or inventor(s) certificate having a filing date before that of the application on which priority is claimed:

COUNTRY	APPLICATION NUMBER	DATE FILED	PRIORITY CLAIMED UNDER 35
			YES: NO:
			YES: NO:

Provisional Application

I hereby claim the benefit under Title 35, United States Code Section 119(e) of any United States provisional application(s) listed below:

APPLICATION SERIAL NUMBER	FILING DATE
60/141,903	June 30, 1999

U.S. Priority Claim

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

APPLICATION SERIAL NUMBER	FILING DATE	STATUS(patented/pending/abandoned)

POWER OF ATTORNEY:

As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) listed below to prosecute this application and transact all business in the Patent and Trademark Office connected therewith:

Dov Rosenfeld, Reg. No. 38,687

Send Correspondence to:	**Direct Telephone Calls To:**
Dov Rosenfeld 5507 College Avenue, Suite 2 Oakland, CA 94618	Dov Rosenfeld, Reg. No. 38,687 Tel: (510) 547-3378

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.

Name of First Inventor: Russell S. Dietz **Citizenship:** USA

Residence: 6146 Ostenberg Drive, San Jose, CA 95120-2736

Post Office Address: Same

First Inventor's Signature **Date**

Declaration and Power of Attorney (Continued)
Case No; «Case CaseNumber»
Page 2 APPT-001-3

ADDITIONAL INVENTOR SIGNATURES:

Name of Second Inventor: Joseph R. Maixner **Citizenship:** USA

Residence: 121 Driftwood Court, Aptos, CA 95003

Post Office Address: Same

Inventor's Signature **Date**

Name of Third Inventor: Andrew A. Koppenhaver **Citizenship:** USA

Residence: 9325 W. Hinsdale Place, Littleton, CO 80128

Post Office Address: Same

Inventor's Signature **Date** 10/10/2000

Our Ref./Docket No: APPT-0[illegible]3 Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz, *et al.*

Application No.: 09/608126

Filed: June 30, 2000

Title: RE-USING INFORMATION FROM DATA TRANSACTIONS FOR MAINTAINING STATISTICS IN NETWORK MONITORING

Group Art Unit:

Examiner: (Unassigned)

REQUEST FOR RECORDATION OF ASSIGNMENT

Assistant Commissioner for Patents
Washington, D.C. 20231
Attn: Box Assignment

Dear Assistant Commissioner:

Enclosed herewith for recordation in the records of the U.S. Patent and Trademark Office is an original Assignment, an Assignment Cover Sheet, and $40.00. Please record and return the Assignment.

Respectfully Submitted,

Nov 1, 2000
Date

Dov Rosenfeld, Reg. No. 38687

Address for correspondence:

Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618
Tel. (510) 547-3378; Fax: (510) 653-7992

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 Assistant Commissioner for Patents, Washington, D.C. 20231 on.

Date: Nov 1, 2000

Signed:

Name: Dov Rosenfeld, Reg. No. 38687

Page 1 of 2

#3

UNITED STATES PATENT AND TRADEMARK OFFICE

COMMISSIONER FOR PATENTS
UNITED STATES PATENT AND TRADEMARK OFFICE
WASHINGTON, D.C. 20231
www.uspto.gov

APPLICATION NUMBER	FILING/RECEIPT DATE	FIRST NAMED APPLICANT	ATTORNEY DOCKET NUMBER
09/608,126	06/30/2000	Russell S. Dietz	APPT-001-3

FORMALITIES LETTER
OC000000005444855

Dov Rosenfeld
Suite 2
5507 College Avenue
Oakland, CA 94618

Date Mailed: 10/02/2000

NOTICE TO FILE MISSING PARTS OF NONPROVISIONAL APPLICATION

FILED UNDER 37 CFR 1.53(b)

Filing Date Granted

An application number and filing date have been accorded to this application. The item(s) indicated below, however, are missing. Applicant is given TWO MONTHS from the date of this Notice within which to file all required items and pay any fees required below to avoid abandonment. Extensions of time may be obtained by filing a petition accompanied by the extension fee under the provisions of 37 CFR 1.136(a).

- The statutory basic filing fee is missing.
  Applicant must submit **$ ~~690~~** *to complete the basic filing fee and/or file a small entity statement claiming such status (37 CFR 1.27).* 710
- Total additional claim fee(s) for this application is $18.
  - **$18** for **1** total claims over 20.
- The oath or declaration is missing.
  A properly signed oath or declaration in compliance with 37 CFR 1.63, identifying the application by the above Application Number and Filing Date, is required.
- To avoid abandonment, a late filing fee or oath or declaration surcharge as set forth in 37 CFR 1.16(e) of $130 for a non-small entity, must be submitted with the missing items identified in this letter.

- **The balance due by applicant is $ 838.**

858+40 =$898

/09/2000 KZEWDIE 00000036 09608126
FC:101 710.00 OP
FC:105 130.00 OP
FC:103 18.00 OP

A copy of this notice MUST be returned with the reply.

Customer Service Center
Initial Patent Examination Division (703) 308-1202

PART 2 - COPY TO BE RETURNED WITH RESPONSE

Our Docket/Ref. No.: APPT-001-3

Patent ~~2755~~
2151

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz et al.

Serial No.: 09/608126

Filed: June 30, 2000

Title: RE-USING INFORMATION FROM DATA TRANSACTIONS FOR MAINTAINING STATISTICS IN NETWORK MONITORING

Group Art Unit:

Examiner:

Commissioner for Patents
Washington, D.C. 20231

TRANSMITTAL: INFORMATION DISCLOSURE STATEMENT

Dear Commissioner:

Transmitted herewith are:

_X_ An Information Disclosure Statement for the above referenced patent application, together with PTO form 1449 and a copy of each reference cited in form 1449.

__ A check for petition fees.

_X_ Return postcard.

_X_ The commissioner is hereby authorized to charge payment of any missing fee associated with this communication or credit any overpayment to Deposit Account 50-0292.

A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED

Respectfully submitted,

Date: 30 Mar 2002

Dov Rosenfeld
Attorney/Agent for Applicant(s)
Reg. No. 38687

Correspondence Address:
Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618
Telephone No.: +1-510-547-3378

Certificate of Mailing under 37 CFR 1.18

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, Washington, D.C. 20231.

Date of Deposit: 30 Mar 2002 Signature: ________
Dov Rosenfeld, Reg. No. 38,687

#5

Our Docket/Ref. No.: APPT-001-3

Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz et al.

Serial No.: 09/608126

Filed: June 30, 2000

Title: RE-USING INFORMATION FROM DATA TRANSACTIONS FOR MAINTAINING STATISTICS IN NETWORK MONITORING

Group Art Unit: 2755

Examiner:

Commissioner for Patents
Washington, D.C. 20231

INFORMATION DISCLOSURE STATEMENT

Dear Commissioner:

This Information Disclosure Statement is submitted:

X under 37 CFR 1.97(b), or
(Within three months of filing national application; or date of entry of international application; or before mailing date of first office action on the merits; whichever occurs last)

X Applicant(s) submit herewith Form PTO 1449-Information Disclosure Citation together with copies, of patents, publications or other information of which applicant(s) are aware, which applicant(s) believe(s) may be material to the examination of this application and for which there may be a duty to disclose in accordance with 37 CFR 1.56.

X (Certification) Each item of information contained in this information disclosure statement was first cited in a formal communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of this information disclosure statement (written opinion from PCT mailed Jan 11,2002).

It is expressly requested that the cited information be made of record in the application and appear among the "references cited" on any patent to issue therefrom.

As provided for by 37 CFR 1.97(g) and (h), no inference should be made that the information and references cited are prior art merely because they are in this statement and no representation is

Certificate of Mailing under 37 CFR 1.18

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, Washington, D.C. 20231.

Date of Deposit: 30 Mar 2002 Signature:
Dov Rosenfeld, Reg. No. 38,687

being made that a search has been conducted or that this statement encompasses all the possible relevant information.

Date: 30 Mar 2002

Respectfully submitted,

Dov Rosenfeld
Attorney/Agent for Applicant(s)
Reg. No. 38687

Correspondence Address:
Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618
Telephone No.: +1-510-547-3378

..M - 1449 SHEET 1 OF 1.

INFORMATION DISCLOSURE STATEMENT (Use several sheets if necessary)	ATTY. DOCKET NO. APPT-001-3	SERIAL NO. 09/608126
	APPLICANT Dietz et al.	
	FILING DATE 6/30/2000	GROUP ~~2755~~ 2142

U.S. PATENT DOCUMENTS

*EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF APPROPRIATE
Uh	AA	5,703,877	Dec. 30, 1997	Nuber et al.	370	395	Nov. 22, 1995
Uh	AB	5,892,754	Apr.6, 1999	Kompella et al.	370	236	Jun. 7, 1996
	AC						
	AD						
	AE						
	AF						
	AG						
	AH						
	AI						
	AJ						
	AK						
	AL						
	AM						
	AN						

FOREIGN PATENT DOCUMENTS

		DOCUMENT NUMBER	PUBLI-CATION DATE	COUNTRY	CLASS	SUB-CLASS	TRANS-LATION YES	NO
	AO							

OTHER DISCLOSURES (Including Author, Title, Date, Pertinent Pages, Place of Publication, Etc.)

	AP

EXAMINER	DATE CONSIDERED 6/19/03

*EXAMINER: initial if citation considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include a copy of this form with next communication to Applicant.

US005703877A

United States Patent [19]

Nuber et al.

[11] **Patent Number: 5,703,877**

[45] **Date of Patent: *Dec. 30, 1997**

[54] **ACQUISITION AND ERROR RECOVERY OF AUDIO DATA CARRIED IN A PACKETIZED DATA STREAM**

[75] Inventors: **Ray Nuber**, La Jolla; **Paul Moroney**, Olivenhain; **G. Kent Walker**, Escondido, all of Calif.

[73] Assignee: **General Instrument Corporation of Delaware**, Chicago, Ill.

[*] Notice: The term of this patent shall not extend beyond the expiration date of Pat. No. 5,517,250.

[21] Appl. No.: **562,611**

[22] Filed: **Nov. 22, 1995**

[51] **Int. Cl.**^{6} **H04J 3/06**; H04N 7/12

[52] **U.S. Cl.** **370/395**; 370/510; 370/514; 375/366; 348/423; 348/462; 348/466; 348/467

[58] **Field of Search** 370/389, 395, 370/503, 509, 510, 514, 516; 375/362, 365, 366, 368, 371; 348/423, 461, 462, 464, 466, 467

[56] **References Cited**

U.S. PATENT DOCUMENTS

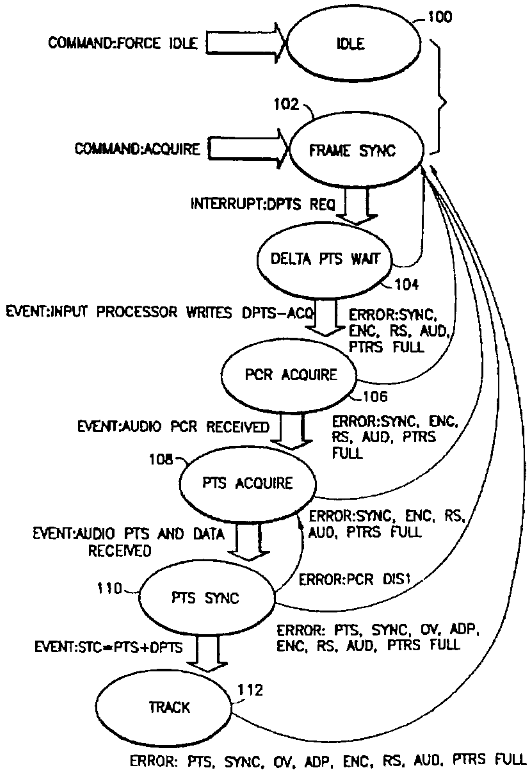
5,365,272	11/1994	Siracusa	348/461
5,376,969	12/1994	Zdepski	348/466
5,467,342	11/1995	Logston et al.	370/253
5,517,250	5/1996	Hoogenboom et al.	348/467
5,537,409	7/1996	Moriyama et al.	370/471

Primary Examiner—Alpus H. Hsu
Attorney, Agent, or Firm—Barry R. Lipsitz

[57] **ABSTRACT**

Audio data is processed from a packetized data stream carrying digital television information in a succession of fixed length transport packets. Some of the packets contain a presentation time stamp (PTS) indicative of a time for commencing the output of associated audio data. After the audio data stream has been acquired, the detected audio packets are monitored to locate subsequent PTS's for adjusting the timing at which audio data is output, thereby providing proper lip synchronization with associated video. Errors in the audio data are processed in a manner which attempts to maintain synchronization of the audio data stream while masking the errors. In the event that the synchronization condition cannot be maintained, for example in the presence of errors over more than one audio frame, the audio data stream is reacquired while the audio output is concealed. An error condition is signaled to the audio decoder by altering the audio synchronization word associated with the audio frame in which the error has occurred.

25 Claims, 4 Drawing Sheets


100
COMMAND:FORCE IDLE
IDLE
102
COMMAND:ACQUIRE
FRAME SYNC
INTERRUPT:DPTS REQ
DELTA PTS WAIT
104
EVENT:INPUT PROCESSOR WRITES DPTS-ACQ
ERROR:SYNC, ENC, RS, AUD, PTRS FULL
PCR ACQUIRE
106
EVENT:AUDIO PCR RECEIVED
ERROR:SYNC, ENC, RS, AUD, PTRS FULL
108
PTS ACQUIRE
ERROR:SYNC, ENC, RS, AUD, PTRS FULL
EVENT:AUDIO PTS AND DATA RECEIVED
ERROR:PCR DIS1
110
PTS SYNC
ERROR: PTS, SYNC, OV, ADP, ENC, RS, AUD, PTRS FULL
EVENT:STC=PTS+DPTS
112
TRACK
ERROR: PTS, SYNC, OV, ADP, ENC, RS, AUD, PTRS FULL

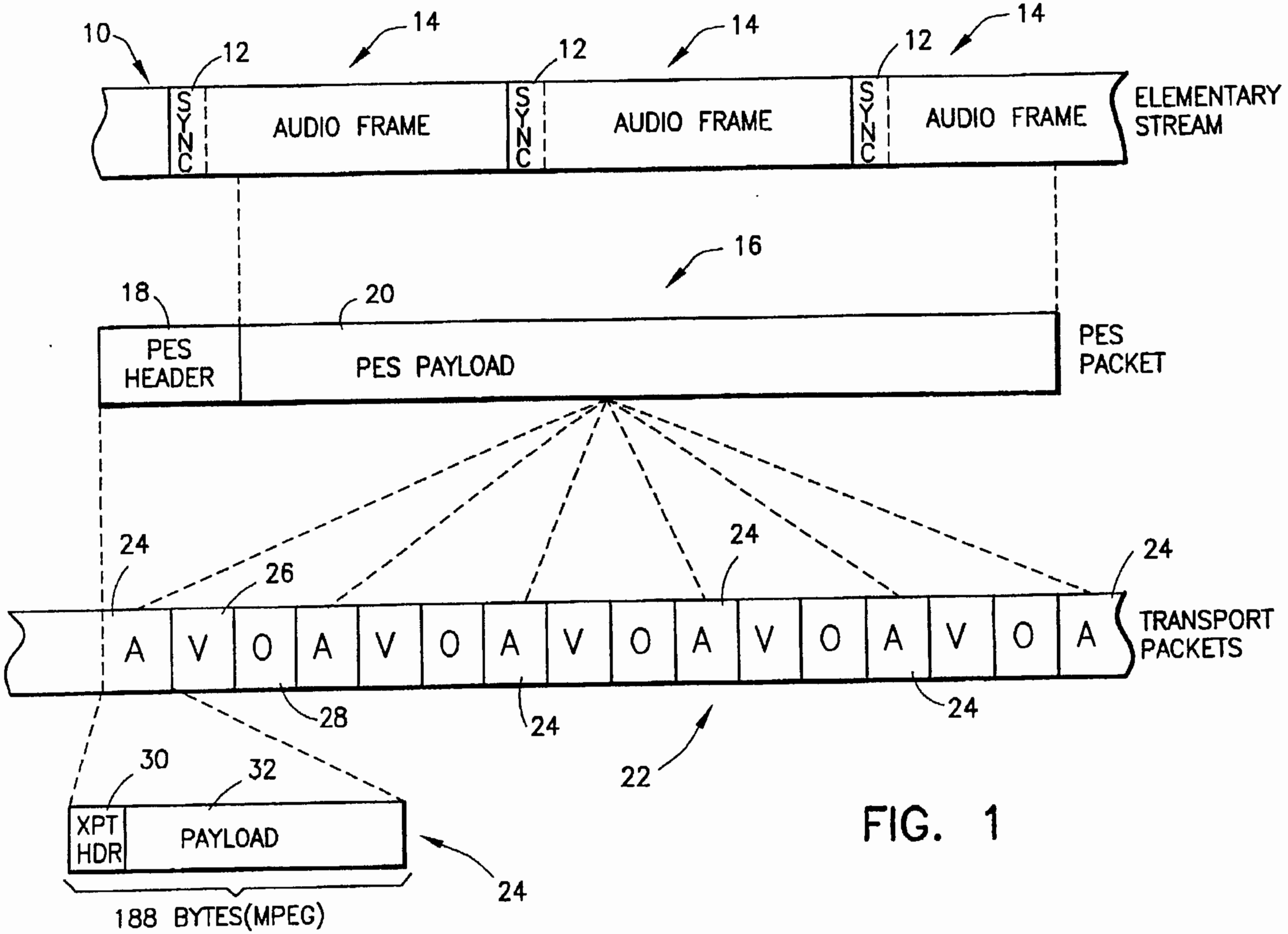

10
12
14
12
14
12
14
SYNC
AUDIO FRAME
SYNC
AUDIO FRAME
SYNC
AUDIO FRAME
ELEMENTARY STREAM
16
18
20
PES HEADER
PES PAYLOAD
PES PACKET
24
26
24
24
A
V
O
A
V
O
A
V
O
A
V
O
A
V
O
A
TRANSPORT PACKETS
28
24
22
24
30
32
XPT HDR
PAYLOAD
24
188 BYTES(MPEG)

FIG. 1

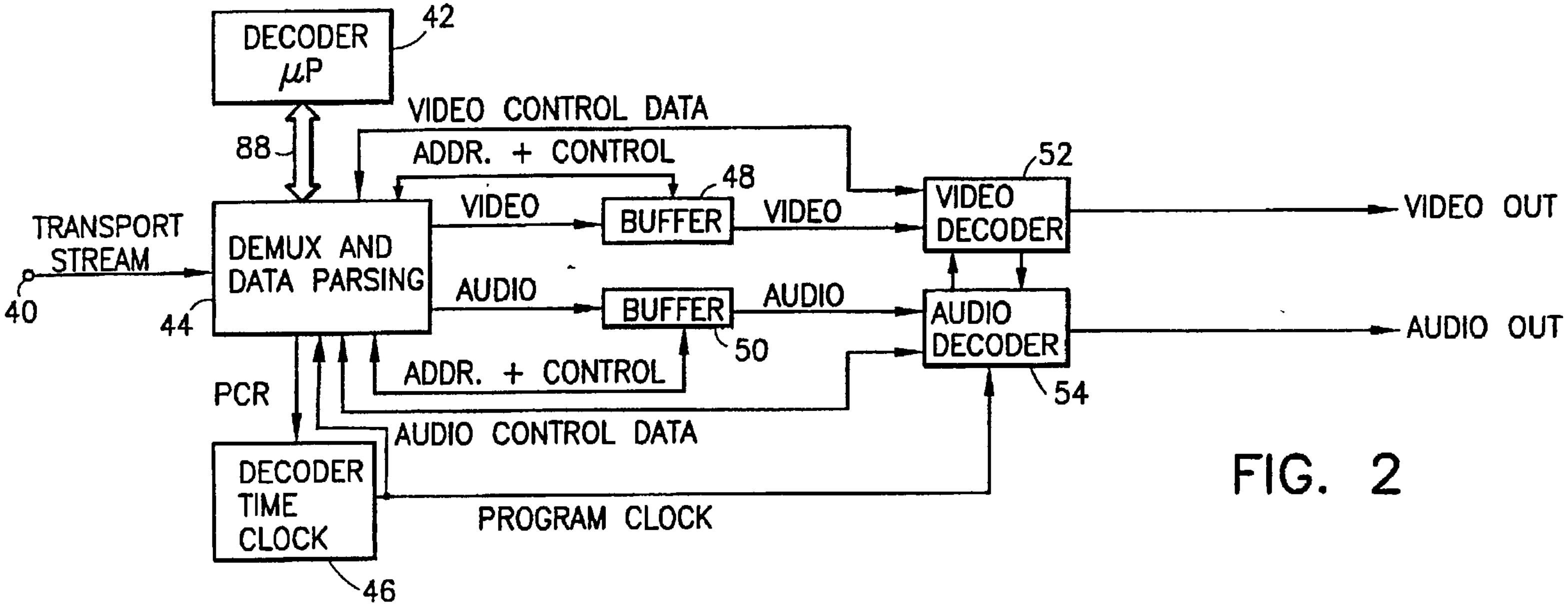

DECODER
μP
42
88
VIDEO CONTROL DATA
ADDR. + CONTROL
48
52
TRANSPORT
STREAM
DEMUX AND
DATA PARSING
VIDEO
BUFFER
VIDEO
VIDEO
DECODER
VIDEO OUT
40
44
AUDIO
BUFFER
AUDIO
AUDIO
DECODER
AUDIO OUT
50
54
PCR
ADDR. + CONTROL
AUDIO CONTROL DATA
DECODER
TIME
CLOCK
PROGRAM CLOCK
46

FIG. 2

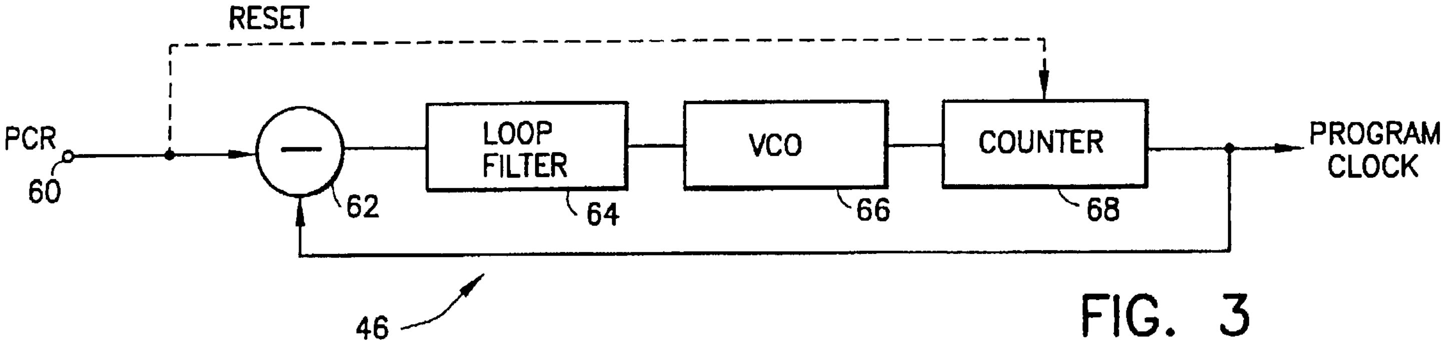
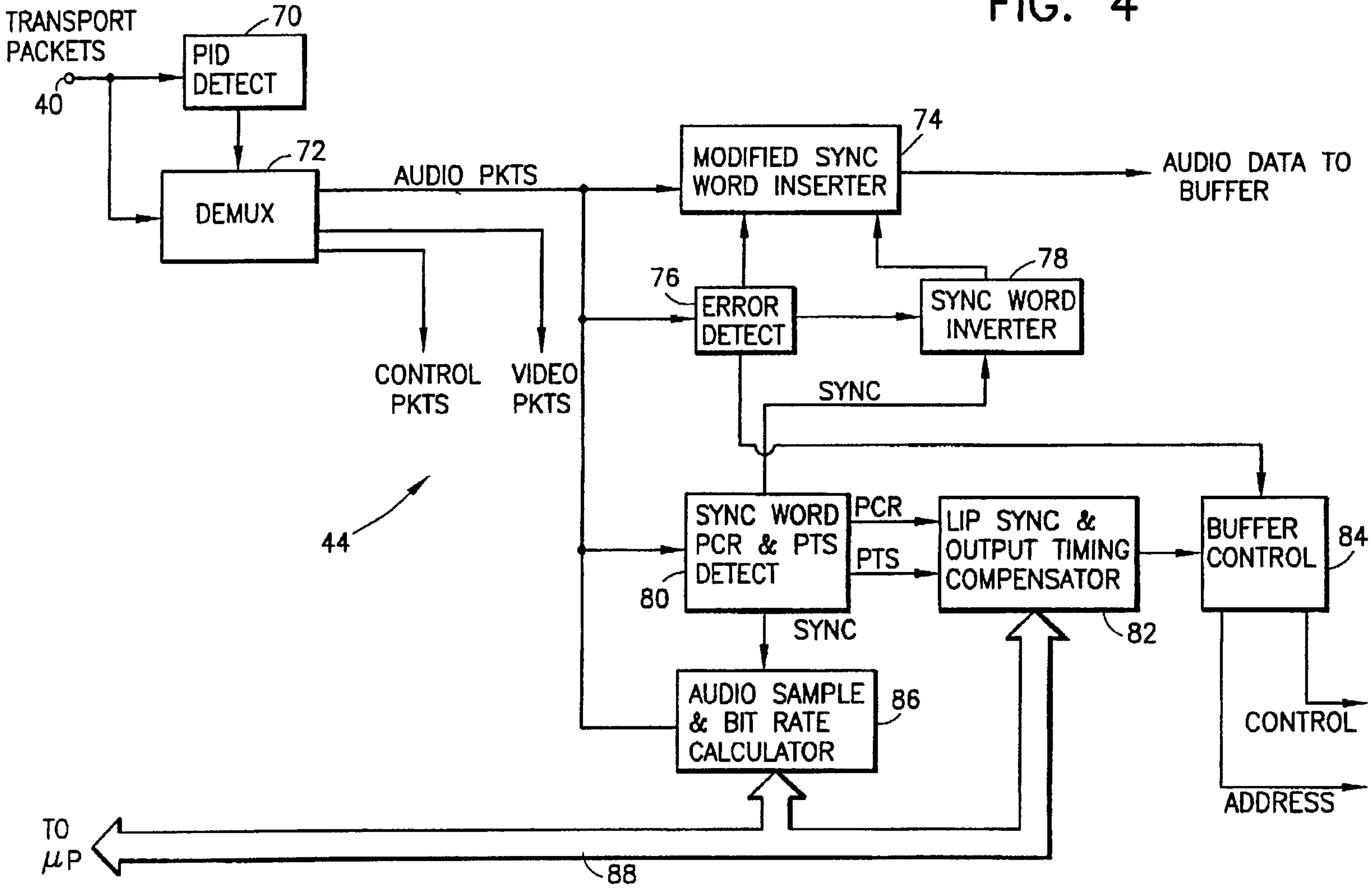

RESET
PCR
60
62
LOOP
FILTER
64
VCO
66
COUNTER
68
PROGRAM
CLOCK
46

FIG. 3

FIG. 4


TRANSPORT PACKETS
40
70
PID DETECT
72
DEMUX
AUDIO PKTS
CONTROL PKTS
VIDEO PKTS
74
MODIFIED SYNC WORD INSERTER
AUDIO DATA TO BUFFER
76
ERROR DETECT
78
SYNC WORD INVERTER
SYNC
44
SYNC WORD PCR & PTS DETECT
80
PCR
PTS
LIP SYNC & OUTPUT TIMING COMPENSATOR
82
BUFFER CONTROL
84
SYNC
AUDIO SAMPLE & BIT RATE CALCULATOR
86
CONTROL
ADDRESS
TO μP
88

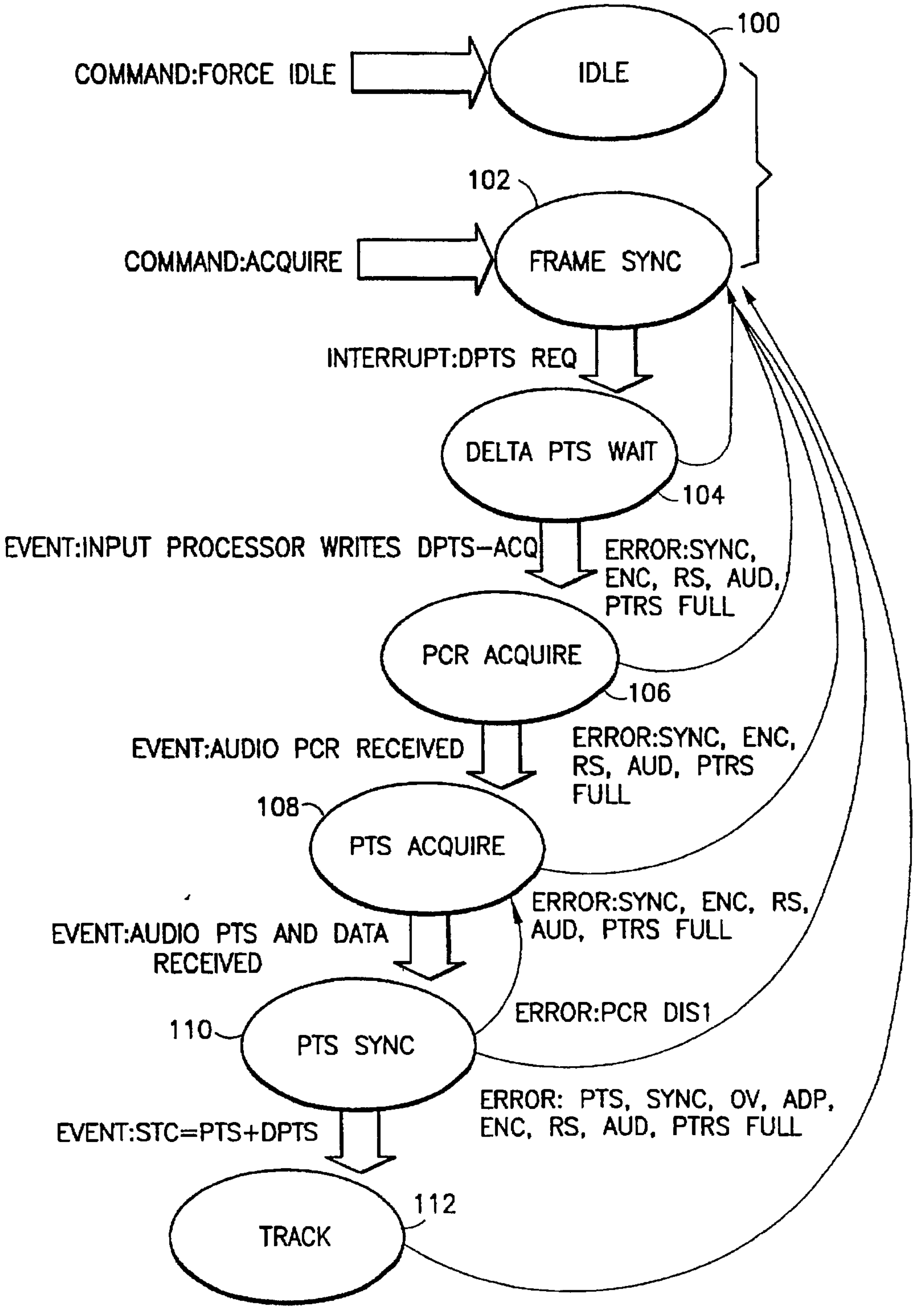

100
COMMAND:FORCE IDLE
IDLE
102
COMMAND:ACQUIRE
FRAME SYNC
INTERRUPT:DPTS REQ
DELTA PTS WAIT
104
EVENT:INPUT PROCESSOR WRITES DPTS-ACQ
ERROR:SYNC, ENC, RS, AUD, PTRS FULL
PCR ACQUIRE
106
EVENT:AUDIO PCR RECEIVED
ERROR:SYNC, ENC, RS, AUD, PTRS FULL
108
PTS ACQUIRE
ERROR:SYNC, ENC, RS, AUD, PTRS FULL
EVENT:AUDIO PTS AND DATA RECEIVED
ERROR:PCR DIS1
110
PTS SYNC
ERROR: PTS, SYNC, OV, ADP, ENC, RS, AUD, PTRS FULL
EVENT:STC=PTS+DPTS
112
TRACK
ERROR: PTS, SYNC, OV, ADP, ENC, RS, AUD, PTRS FULL

FIG. 5

ACQUISITION AND ERROR RECOVERY OF AUDIO DATA CARRIED IN A PACKETIZED DATA STREAM

BACKGROUND OF THE INVENTION

The present invention relates to a method and apparatus for acquiring audio data from a packetized data stream and recovery from errors contained in such data.

Various standards have emerged for the transport of digital data, such as digital television data. Examples of such standards include the Moving Pictures Experts Group (MPEG) standards and the DigiCipher® II standard proprietary to General Instrument Corporation of Chicago, Ill., U.S.A., the assignee of the present invention. The DigiCipher® II standard extends the MPEG-2 systems and video standards, which are widely known and recognized as transport and video compression specifications specified by the International Standards Organization (ISO) in Document series ISO 13818. The MPEG-2 specification's systems "layer" provides a transmission medium independent coding technique to build bitstreams containing one or more MPEG programs. The MPEG coding technique uses a formal grammar ("syntax") and a set of semantic rules for the construction of bitstreams. The syntax and semantic rules include provisions for demultiplexing, clock recovery, elementary stream synchronization and error handling.

The MPEG transport stream is specifically designed for use with media that can generate data errors. Many programs, each comprised of one or more elementary streams, may be combined into a transport stream. Examples of services that can be provided using the MPEG format are television services broadcast over terrestrial, cable television and satellite networks as well as interactive telephony-based services. The primary mode of information carriage in MPEG broadcast applications will be the MPEG-2 transport stream. The syntax and semantics of the MPEG-2 transport stream are defined in International Organisation for Standardisation, ISO/IEC 13818-1, International Standard, 1994 entitled "Generic Coding of Moving Pictures and Associated Audio: Systems," recommendation H.222, incorporated herein by reference.

Multiplexing according to the MPEG-2 standard is accomplished by segmenting and packaging elementary streams such as compressed digital video and audio into packetized elementary stream (PES) packets which are then segmented and packaged into transport packets. As noted above, each MPEG transport packet is fixed at 188 bytes in length. The first byte is a synchronization byte having a specific eight-bit pattern, e.g., 01000111. The sync byte indicates the beginning of each transport packet.

Following the sync byte is a three-byte field which includes a one-bit transport packet error indicator, a one-bit payload unit start indicator, a one-bit transport priority indicator, a 13-bit packet identifier (PID), a two-bit transport scrambling control, a two-bit adaptation field control, and a four-bit continuity counter. The remaining 184 bytes of the packet may carry the data to be communicated. An optional adaptation field may follow the prefix for carrying both MPEG related and private information of relevance to a given transport stream or the elementary stream carried within a given transport packet. Provisions for clock recovery, such as a program clock reference (PCR), and bitstream splicing information are typical of the information carried in the adaptation field. By placing such information in an adaptation field, it becomes encapsulated with its associated data to facilitate remultiplexing and network routing operations. When an adaptation field is used, the payload is correspondingly shorter in length.

The PCR is a sample of the system time clock (STC) for the associated program at the time the PCR bytes are received at the decoder. The decoder uses the PCR values to synchronize a decoder system time clock (STC) with the encoder's system time clock. The lower nine bits of a 42-bit STC provide a modulo-300 counter that is incremented at a 27 MHz clock rate. At each modulo-300 rollover, the count in the upper 33 bits is incremented, such that the upper bits of the STC represent time in units of a 90 kHz clock period. This enables presentation time stamps (PTS) and decode time stamps (DTS) to be used to dictate the proper time for the decoder to decode access units and to present presentation units with the accuracy of one 90 kHz clock period. Since each program or service carried by the data stream may have its own PCR, the programs can be multiplexed asynchronously.

Synchronization of audio, video and data presentation within a program is accomplished using a time stamp approach. Presentation time stamps (PTSs) and/or decode time stamps (DTSs)are inserted into the transport stream for the separate video and audio packets. The PTS and DTS information is used by the decoder to determine when to decode and display a picture and when to play an audio segment. The PTS and DTS values are relative to the same system time clock sampled to generate the PCRs.

All MPEG video and audio data must be formatted into a packetized elementary stream (PES) formed from a succession of PES packets. Each PES packet includes a PES header followed by a payload. The PES packets are then divided into the payloads of successive fixed length transport packets.

PES packets are of variable and relatively long length. Various optional fields, such as the presentation time stamps and decode time stamps may be included in the PES header. When the transport packets are formed from the PES, the PES headers immediately follow the transport packet headers. A single PES packet may span many transport packets and the subsections of the PES packet must appear in consecutive transport packets of the same PID value. It should be appreciated, however, that these transport packets may be freely multiplexed with other transport packets having different PIDs and carrying data from different elementary streams within the constraints of the MPEG-2 Systems specification.

Video programs are carried by placing coded MPEG video streams into PES packets which are then divided into transport packets for insertion into a transport stream. Each video PES packet contains one or more coded video pictures, referred to as video "access units." A PTS and/or a DTS value may be placed into the PES packet header that encapsulates the associated access units. The DTS indicates when the decoder should decode the access unit into a presentation unit. The PTS is used to actuate the decoder to present the associated presentation unit.

Audio programs are provided in accordance with the MPEG Systems specification using the same specification of the PES packet layer. PTS values may be included in those PES packets that contain the first byte of an audio access unit (sync frame). The first byte of an audio access unit is part of an audio sync word. An audio frame is defined as the data between two consecutive audio sync words, including the preceding sync word and not including the succeeding sync word.

In DigiCipher® II, audio transport packets include one or both of an adaptation field and payload field. The adaptation field may be used to transport the PCR values. The payload field transports the audio PES, consisting of PES headers and PES data. PES headers are used to transport the audio PTS values. Audio PES data consists of audio frames as specified, e.g., by the Dolby® AC-3 or Musicam audio syntax specifications. The AC-3 specifications are set forth in a document entitled Digital Audio Compression (AC-3), ATSC Standard, Doc. A/52, United States Advanced Television Systems Committee. The Musicam specification can be found in the document entitled "Coding of Moving Pictures and Associated Audio for Digital Storage Media at Up to About 1.5 MBIT/s," *Part* 3 Audio, 11172-3 (MPEG-1) published by ISO. Each syntax specifies an audio sync frame as audio sync word, followed by audio information including audio sample rate, bit rate and/or frame size, followed by audio data.

In order to reconstruct a television signal from the video and audio information carried in an MPEG/DigiCipher® II transport stream, a decoder is required to process the video packets for output to a video decompression processor (VDP) and the audio packets for output to an audio decompression processor (ADP). In order to properly process the audio data, the decoder is required to synchronize to the audio data packet stream. In particular, this is required to enable audio data to be buffered for continuous output to the ADP and to enable the audio syntax to be read for audio rate information necessary to delay the audio output to achieve proper lip synchronization with respect to the video of the same program.

Several events can result in error conditions with respect to the audio processing. These include loss of audio transport packets due to transmission channel errors. Errors will also result from the receipt of audio packets which are not properly decrypted or are still encrypted. A decoder must be able to handle such errors without significantly degrading the quality of the audio output.

The decoder must also be able to handle changes in the audio sample rate and audio bit rate. The audio sample rate for a given audio elementary stream will rarely change. The audio bit rate, however, can often change at program boundaries, and at the start and end of commercials. It is difficult to maintain synchronization to the audio stream through such rate changes, since the size of the audio sync frames is dependent on the audio sample rate and bit rate. Handling undetected errors in the audio stream, particularly in systems where error detection is weak, complicates the tracking of the audio stream through rate changes. When a received bitstream indicates that an audio rate has changed, the rate may or may not have actually changed. If the decoder responds to an indication from the bitstream that the audio rate has changed when the indication is in error and the rate has not changed, a loss of audio synchronization will likely occur. This can result in an audio signal degradation that is noticeable to an end user.

To support an audio sample rate change, the audio clock rates utilized by the decoder must be changed. This process can take significant time, again degrading the quality of the audio output signal. Still further, such a sample rate change will require the audio buffers to be cleared to establish a different sample-rate-dependent lip sync delay. Thus, it may not be advantageous to trust a signal in the received bitstream indicating that the audio sample rate has changed.

With respect to bit rate changes, the relative frequency of such changes compared to undetected errors in the bit rate information will be dominated by whether the receiver has adequate error detection. Thus, it would be advantageous to provide a decoder having two modes of operation. In a robust error detection environment such as for satellite communications or cable media, where error detection is robust, a seamless mode of operation can be provided by trusting a bit rate change indication provided by the data. In a less robust error detection environment, indications of bit rate changes can be ignored, at the expense of requiring resynchronization of the audio in the event that the bit rate has actually changed.

It would be further advantageous to provide an audio decoder in which synchronization to the audio bitstream is maintained when the audio data contains errors. Such a decoder should conceal the audio for those sync frames in which an error has occurred, to minimize the aural impact of audio data errors.

It would be still further advantageous to provide a decoder in which the timing at which audio data is output from the decoder's audio buffer is adjusted on an ongoing basis. The intent of such adjustments would be to insure correct presentation time for audio elementary streams.

The present invention provides methods and apparatus for decoding digital audio data from a packetized transport stream having the aforementioned and other advantages.

SUMMARY OF THE INVENTION

In accordance with the present invention, a method is provided for processing digital audio data from a packetized data stream carrying television information in a succession of fixed length transport packets. Each of the packets includes a packet identifier (PID). Some of the packets contain a program clock reference (PCR) value for synchronizing a decoder system time clock (STC). Some of the packets contain a presentation time stamp (PTS) indicative of a time for commencing the output of associated data for use in reconstructing a television signal. In accordance with the method, the PID's for the packets carried in the data stream are monitored to identify audio packets associated with the desired program. The audio packets are examined to locate the occurrence of at least one audio synchronization word therein for use in achieving a synchronization condition. The audio packets are monitored after the synchronization condition has been achieved to locate an audio PTS. After the PTS is located, the detected audio packets are searched to locate the next audio synchronization word. Audio data following the next audio synchronization word is stored in a buffer. The stored audio data is output from the buffer when the decoder system time clock reaches a specified time derived from the PTS. The detected audio packets are continually monitored to locate subsequent audio PTS's for adjusting the timing at which the stored audio data is output from the buffer on an ongoing basis.

A PTS pointer can be provided to maintain a current PTS value and an address of the buffer identifying where the sync word of an audio frame referred to by the current PTS is stored. In order to provide the timing adjustment, the PTS value in the PTS pointer is replaced with a new PTS value after data stored at the address specified by the PTS pointer has been output from the buffer. The address specified by the PTS pointer is then replaced with a new address corresponding to the sync word of an audio frame referred to by the new PTS value. The output of data from the buffer is suspended when the new buffer address is reached during the presentation process. The output of data from the buffer is recommenced when the decoder's system time clock reaches a specified time derived from the new PTS value.

In an illustrated embodiment, the output of data from the buffer is recommenced when the decoder's system time clock reaches the time indicated by the sum of the new PTS value and an offset value. The offset value provides proper lip synchronization by accounting for any decoder video signal processing delay. In this manner, after the audio and video data has been decoded, the audio data can be presented synchronously with the video data so that, for example, the movement of a person's lips in the video picture will be sufficiently synchronous to the sound reproduced.

The method of the present invention can comprise the further step of commencing a reacquisition of the audio synchronization condition if the decoder's system time clock is beyond the specified time derived from the new PTS value before the output of data from the buffer is recommenced. Thus, if a PTS designates that an audio frame should be presented at a time which has already passed, reacquisition of the audio data will automatically commence to correct the timing error, thus minimizing the duration of the resultant audio artifact.

In the illustrated embodiment, two consecutive audio synchronization words define an audio frame therebetween, including the preceding sync word, but not including the succeeding sync word. The occurrence of errors may be detected in the audio packets. Upon detecting a first audio packet of a current audio frame containing an error, the write pointer for the buffer is advanced by the maximum number of bytes (N) contained in one of the fixed length transport packets. At the same time, the current audio frame is designated as being in error. The subsequent audio packets of the current audio frame are monitored for the next audio synchronization word after the error has been detected. If the synchronization word is not received at the expected point in the audio elementary stream, subsequent data is not stored in the buffer until the sync word is located. Storage of audio data into the buffer is resumed with the next sync word if the next audio synchronization word is located within N bytes after the commencement of the search therefor. If the next audio synchronization word is not located within N bytes after the commencement of the search therefor, a reacquisition of the synchronization condition is commenced. These steps will insure the buffer is maintained at the correct fullness when as many as one transport packet is lost per audio sync frame, even with the sync frame size changes such as with a sample rate of 44.1 ksps, and will resynchronize the audio when too many audio transport packets are lost.

Whenever the audio data from which the television audio is being reconstructed is in error, it is preferable to conceal the error in the television audio. In the illustrated embodiment, a current audio frame is designated as being in error by altering the audio synchronization word for that frame. For example, every other bit of the audio synchronization word can be inverted. The error in the television audio for the corresponding audio frame may then be concealed in response to an altered synchronization word during the decoding and presentation process. This method allows the buffering and error detection process to signal the decoding and presentation process when errors occur via the data itself, without the need for additional interprocess signals.

The audio data can include information indicative of an audio sample rate and audio bit rate, at least one of which is variable. In such a situation, it is advantageous to maintain synchronization within the audio elementary stream during a rate change indicated by the audio data. This can be accomplished by ignoring an audio sample rate change indicated by the audio data on the assumption that the sample rate has not actually changed, and concealing the audio frame containing the data indicative of an audio sample rate change while attempting to maintain the synchronization condition. This strategy will properly respond to an event in which the audio sample rate change or bit rate change indication is the result of an error in the indication itself, as opposed to an actual rate change.

Similarly, audio data can be processed in accordance with a new rate indicated by the audio data in the absence of an error indication pertaining to the audio frame containing the new rate, while attempting to maintain the synchronization condition. The audio data is processed without changing the rate if an error indication pertains to the audio frame containing the new rate. At the same time, the audio frame to which the error condition pertains is concealed while the decoder attempts to maintain the synchronization condition. If the synchronization condition cannot be maintained, a reacquisition of the synchronization condition is commenced, as desired when the sample rate actually changes.

Apparatus in accordance with the present invention acquires audio information carried by a packetized data stream. The apparatus also handles errors contained in the audio information. Means are provided for identifying audio packets in the data stream. An audio elementary stream is recovered from the detected audio packets for storage in a buffer. An audio presentation time stamp (PTS) is located in the detected audio packets. Means responsive to the PTS are provided for commencing the output of audio data from the buffer at a specified time. Means are provided for monitoring the detected audio packets after the output of audio data from the buffer has commenced, in order to locate subsequent audio PTS's for use in governing the output of audio data from the buffer to insure audio is presented synchronous to any other elementary streams of the same program and to maintain correct buffer fullness.

The apparatus can further comprise means for maintaining a PTS pointer with a current PTS value and an address of the buffer identifying where a portion of audio data referred to by the current PTS is stored. Means are provided for replacing the PTS value in the PTS pointer with a new current PTS value after data stored at the address set forth in the PTS pointer has been output from the buffer. The address in the PTS pointer is then replaced with a new address corresponding to a portion of audio data referred to by the new current PTS value. Means responsive to the PTS pointer are provided for suspending the output of data from the buffer when the new address is reached. Means are provided for recommencing the output of data from the buffer at a time derived from the new current PTS value. In the event that the new current PTS value is outside a predetermined range, means provided in the apparatus conceal the audio signal and reestablish synchronization.

In an illustrated embodiment, the audio transport packets have a fixed length of M bytes. The transport packets carry a succession of audio frames each contained wholly or partially in said packets. The audio frames each begin with an audio synchronization word. Means are provided for detecting the occurrence of errors in the audio packets. A write pointer for the buffer is advanced by the maximum number of audio frame bytes per audio transport packet (N) and a current audio frame is designated as being in error upon detecting an error in an audio packet of the current audio frame. Means are provided for monitoring the detected audio packets of the current audio frame for the next audio synchronization word after the error has been detected. If the synchronization word is not received where expected within the audio elementary stream, subsequent audio data is not buffered until the next audio synchronization word is received. This process compensates for too many audio bytes having been buffered when the errored audio packet was detected. Such an event will occur each time the lost packet does not carry the maximum number of possible audio data bytes. Means are provided for resuming the storage of audio data in the buffer if the next audio synchronization word is located within N bytes after the commencement of the search therefor. If the next audio synchronization word is not located within said N bytes after the commencement of the search therefor, the audio timing will be reacquired. In this manner, the size of the sync frames buffered will be maintained including for those frames that are marked as being in error, unless the next sync word is not located where expected in the audio elementary stream to recover from the error before buffering any of the next successive frame. This algorithm allows the decode and presentation processes to rely on buffered audio frames being the correct size in bytes, even when data errors result in the loss of an unknown amount of audio data.

Means can also be provided for concealing error in an audio signal reproduced from data output from the buffer when the data output from the buffer is in error. Means are further provided for altering the audio synchronization word associated with a current audio frame, to signal the decode and presentation process that a particular frame is in error. The concealing means are responsive to altered synchronization words for concealing audio associated with the corresponding audio frame.

Decoder apparatus in accordance with the invention acquires audio information carried by a packetized data stream and handles errors therein. Means are provided for identifying audio packets in the data stream. The successive audio frames are extracted from the audio transport packets. Each audio frame is carried by one or more of the packets, and the start of each audio frame is identified by an audio synchronization word. Means responsive to the synchronization words obtain a synchronization condition enabling the recovery of audio data from the detected audio packets for storage in a buffer. Means are provided for detecting the presence of errors in the audio data. Means responsive to the error detecting means control the flow of data through the buffer when an error is present, to attempt to maintain the synchronization condition while masking the error. Means are provided for reestablishing the audio timing if the controlling means cannot maintain the synchronization condition.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a diagrammatic illustration showing how audio transport packets are formed from an elementary stream of audio data;

FIG. 2 is a block diagram of decoder apparatus that can be used in accordance with the present invention;

FIG. 3 is a more detailed block diagram of the decoder system time clock (STC) illustrated in FIG. 2;

FIG. 4 is a more detailed block diagram of the demultiplexing and data parsing circuit of FIG. 2; and

FIG. 5 is a state diagram illustrating the processing of audio data in accordance with the present invention.

DETAILED DESCRIPTION OF THE INVENTION

FIG. 1 is a diagrammatic illustration showing how one or more digital programs can be multiplexed into a stream of transport packets. Multiplexing is accomplished by segmenting elementary streams such as coded video and audio into PES packets and then segmenting these into transport packets. The figure is illustrative only, since a PES packet, such as PES packet 16 illustrated, will commonly translate into other than the six transport packets 24 illustrated.

In the example of FIG. 1, an elementary stream generally designated 10 contains audio data provided in audio frames 14 delineated by synchronization words 12. Similar elementary streams will be provided for video data and other data to be transported.

The first step in forming a transport packet stream is to reconfigure the elementary stream for each type of data into a corresponding packetized elementary stream (PES) formed from successive PES packets, such as packet 16 illustrated. Each PES packet contains a PES header 18 followed by a PES payload 20. The payload comprises the data to be communicated. The PES header 18 will contain information useful in processing the payload data, such as the presentation time stamp (PTS).

The header and payload data from each PES packet are encapsulated into transport packets 24, each containing a transport header 30 and payload data 32. The payload data of the transport packet 24 will contain a portion of the payload data 20 and/or PES header 18 from PES packet 16. In an MPEG implementation, the transport header 30 will contain the packet identifier (PID) which identifies the transport packet, such as an audio transport packet 24, a video transport packet 26, or other data packet 28. In FIG. 1, only the derivation of the audio transport packets 24 is shown. In order to derive video packets 26 and other packets 28, corresponding elementary streams (not shown) are provided which are processed into PES packets and transport packets in essentially the same manner illustrated in FIG. 1 with respect to the formation of the audio transport packets

Each MPEG transport packet contains 188 bytes of data, formed from the four-byte transport header 30 and payload data 32, which can be up to 184 bytes. In the MPEG implementation, an adaptation field of, e.g., eight bytes may be provided between the transport header 30 and payload 32. The variable length adaptation field can contain, for example, the program clock reference (PCR) used for synchronization of the decoder system time clock (STC).

The plurality of audio transport packets 24, video transport packets 26 and other packets 28 is multiplexed as illustrated in FIG. 1 to form a transport stream 22 that is communicated over the communication channel from the encoder to the decoder. The purpose of the decoder is to demultiplex the different types of transport packets from the transport stream, based on the PID's of the individual packets, and to then process each of the audio, video and other components for use in reconstructing a television signal.

FIG. 2 is a block diagram of a decoder for recovering the video and audio data. The transport stream 22 is input to a demultiplexer and data parsing subsystem 44 via terminal 40. The demultiplexing and data parsing subsystem communicates with a decoder microprocessor 42 via a data bus 88. Subsystem 44 recovers the video and audio transport packets from the transport packet stream and parses the PCR, PTS and other necessary data therefrom for use by other decoder components. For example, PCR's are recovered from adaptation fields of transport packets for use in synchronizing a decoder system time clock (STC) 46 to the system time clock of the encoder. Presentation time stamps for the video and audio data streams are recovered from the respective PES packet headers and communicated as video or audio control data to the video decoder 52 and audio decoder 54, respectively.

The decoder time clock 46 is illustrated in greater detail in FIG. 3. An important function of the decoder is the reconstruction of the clock associated with a particular program. This clock is used to reconstruct, for example, the proper horizontal scan rate for the video. The proper presentation rate of audio and video presentation units must also be assured. These are the audio sample rate and the video frame rate. Synchronization of the audio to the video, referred to as "lip sync", is also required.

In order to generate a synchronized program clock, the decoder system time clock (STC) 46 receives the PCR's via terminal 60. Before the commencement of the transport stream decoding, a PCR value is used to preset a counter 68 for the decoder system time clock. As the clock runs, the value of this counter is fed back to a subtracter 62. The local feedback value is then compared with subsequent PCR's in the transport stream as they arrive at terminal 60. When a PCR arrives, it represents the correct STC value for the program. The difference between the PCR value and the STC value, as output from subtracter 62, is filtered by a loop filter 64 and used to drive the instantaneous frequency of a voltage controlled oscillator 66 to either decrease or increase the STC frequency as necessary. The STC has both a 90 kHz and 27 MHz component, and the loop filter 64 converts this to units in the 27 Mhz domain. The output of the VCO 66 is a 27 MHz oscillator signal which is used as the program clock frequency output from the decoder system time clock. Those skilled in the art will recognize that the decoder time clock 46 illustrated in FIG. 3 is implemented using well known phase locked loop (PLL) techniques.

Before beginning audio synchronization, the decoder of FIG. 2, and particularly subsystem 44, will remain idle until it is configured by decoder microprocessor 42. The configuration consists of identifying the type of audio data stream to be processed (e.g., Dolby AC-3 or Musicam audio), identifying the PID of packets from which the audio PCR values are to be extracted, and identifying the PID for audio packets.

During the idle state, subsystem 44 will instruct audio decoder 54 to conceal the audio output. Concealment can be accomplished by zeroing all of the audio samples. Subsequent digital signal processing will result in a smooth aural transition from no sound to sound, and back to no sound. The concealment of the audio output will be terminated when the synchronization process reaches a tracking state. Decoder microprocessor 42 configures the audio format as AC-3 or Musicam, depending on whether audio decoder 54 is an AC-3 or Musicam decoder. Microprocessor 42 determines the audio PID and audio PCR PID from program map information provided in the transport stream. The program map information is essentially a directory of PID's, and is identified via its own PID.

Once the demultiplexer and data parsing subsystem 44 is commanded to enter a Frame Sync state via an acquire command, it will begin searching for two consecutive audio sync words and will supply the decoder microprocessor 42 with the audio sampling rate and audio bit rate indicated within the audio elementary stream. To locate the sync words, subsystem 44 will receive transport packets on the audio PID and extract the PES data, searching for the occurrence of the audio sync word, which is a predetermined, fixed word. For example, the AC-3 audio sync word is 0000 1011 0111 0111 (16 bits) while the Musicam sync word is 1111 1111 1111 (12 bits).

The number of bits between the first bit of two consecutive audio sync words is referred to as the frame size. The frame size depends on whether the audio stream is AC-3 or Musicam and has a different value for each combination of audio sample and bit rate. In a preferred embodiment, subsystem 44 is required to synchronize to AC-3 and Musicam sample rates of 44.1 ksps and 48 ksps. The AC-3 audio syntax conveys the audio sample rate and audio frame size while the Musicam audio syntax conveys the audio sample rate and audio bit rate. Both AC-3 and Musicam specify one sync frame size for each bit rate when the sample rate is 48 ksps. However, AC-3 and Musicam specify two sync frame sizes for each bit rate when the sample rate is 44.1 ksps, a fact which complicates synchronization, especially through packet loss. When the sample rate is 44.1 ksps, the correct sync frame size between the two possibilities is indicated by the least significant bit of the AC-3 frame size code or by a Musicam padding bit.

Once two consecutive audio sync words have been received with the correct number of bytes in between, as specified by the sync frame size, subsystem 44 will store the audio sample rate and audio bit rate implied by the audio syntax for access by the decoder microprocessor 42, interrupting the microprocessor to indicate that subsystem 44 is waiting for the microprocessor to supply it with an audio PTS correction factor. The correction factor is necessary in order to know when to output audio data to the audio decoder 54 during initial acquisition and during tracking for proper lip synchronization. The value is denoted as dPTS. The lip sync value used for tracking is slightly less than that used for initial acquisition to allow for time errors which will exist between any two PTS values, namely that which is used for acquisition and those which are used for tracking.

Decoder microprocessor 42 sets the correction factors such that audio and video-will exit the decoder with the same time relationship as it entered the encoder, thus achieving lip synchronization. These correction factors are determined based on audio sample rate and video frame rate (e.g., 60 Hz or 50 Hz). These dependencies exist because the audio decompression processing time required by audio decoder 54 potentially depends on audio sample and bit rate while the video decompression implemented by video decoder 52 potentially depends on video frame rate and delay mode. In a preferred implementation, the PTS correction factors consist of 11 bits, representing the number of 90 kHz clock periods by which audio data is to be delayed before output to the audio decoder 54. With 11 bit values, the delay can be as high as 22.7 milliseconds.

Once the demultiplexing and data parsing subsystem 44 requests the decoder microprocessor 42 to supply the correction factors, it will monitor reception of consecutive sync words at the expected positions within the audio elementary stream. If an error condition occurs during this time, subsystem 44 will transition to searching for two consecutive audio sync words with the correct number of data bytes in between. Otherwise, subsystem 44 remains in State dPTS-wait until the decoder microprocessor services the interrupt from subsystem 44 by writing dPTS$_{acq}$ to subsystem 44.

Once subsystem 44 is provided with the PTS correction factors, it checks whether a transport packet has been received on the audio PCR PID containing a PCR value, carried in the adaptation field of the packet. Until this has occurred, reception of consecutive sync words will continue [State=PCR Acquire]. If an error condition occurs during this time, subsystem 44 will transition to searching for two consecutive audio sync words [State=Frame Sync]. Otherwise, it will remain in State=PCR Acquire until it receives a PCR value on the audio PCR PID.

After a PCR has been acquired, subsystem **44** will begin searching for a PTS [State=PTS Acquire], which is carried in the PES header of the audio transport packets. Until this has occurred, subsystem **44** will monitor the reception of consecutive sync words. If an error condition occurs during this time, it will transition to an error handling algorithm [State=Error Handling]. Otherwise, it will remain in the PTS acquire state until it receives a PTS value on the audio PID.

When subsystem **44** receives an audio PTS value, it will begin searching for reception of the next audio sync word. This is important since the PTS defines the time at which to output the data which begins with the next audio frame. Since audio frames are not aligned with the audio PES, the number of bytes which will be received between the PTS and the next audio sync word varies with time. If an error condition occurs before reception of the next audio sync word, subsystem **44** returns to searching for audio frame synchronization [State=Frame Sync]. It should be appreciated that since audio sync frames and PES headers are not aligned, it is possible for a PES header, and the PTS which it may contain, to be received between the 12 or 16 bits which form an audio sync word. In this case, the sync word to which the PTS refers is not the sync word which is split by the PES header, but rather the following sync word.

When subsystem **44** receives the next sync word, it has acquired PTS. At this point, it will store the received PTS and the PES data (starting with the sync word which first followed the PTS) into an audio buffer **50**, together with the buffer address at which it writes the sync word. This stored PTS/buffer address pair will allow subsystem **44** to begin outputting audio PES data to the audio decoder **54** at the correct time, starting with the audio sync word. In a preferred embodiment, the buffer **50** is implemented in a portion of dynamic random access memory (DRAM) already provided in the decoder.

Once subsystem **44** begins buffering audio data, a number of parameters must be tracked which will allow it to handle particular error conditions, such as loss of an audio transport packet to transmission errors. These parameters can be tracked using audio pointers including a PTS pointer, a DRAM offset address pointer, and a valid flag pointer discussed in greater detail below.

After PTS is acquired, subsystem **44** begins waiting to synchronize to PTS [State=PTS Sync]. In this state, the demultiplexer and data parsing subsystem **44** continues to receive audio packets via terminal **40**, writes their PES data into buffer **50**, and maintains the error pointers. When this state is entered, subsystem **44** compares its audio STC to the correct output start time, which is the PTS value in the PTS pointer plus the acquisition PTS correction factor ($dPTS_{acq}$). If subsystem **44** discovers that the correct time has passed, i.e., $PCR>PTS+dPTS_{acq}$, one or more of the three values is incorrect and subsystem **44** will flag decoder microprocessor **42**. At this point, the state will revert to State=Frame Sync, and subsystem **44** will return to searching for two consecutive audio sync words. Otherwise, until $PCR=PTS+dPTS_{acq}$, subsystem **44** will continue to receive audio packets, write their PES data into the buffer **50**, maintain the error pointers, and monitor the reception of consecutive sync words.

When $PCR=PTS+dPTS_{acq}$, subsystem **44** has synchronized to PTS and will begin tracking the audio stream [State=Track]. At this time, subsystem **44** will begin transferring the contents of the audio buffer to the audio decoder **54** upon the audio decoder requesting audio data, starting with the sync word located at the buffer address pointed to by the PTS pointer. In the tracking state, subsystem **44** will continue to receive audio packets, write their PES data into the buffer **50**, maintain the error pointers, and monitor reception of consecutive sync words. If an error condition occurs during this time, subsystem **44** will transition to error processing. Otherwise, it will remain in State=Track until an error occurs or microprocessor **42** commands it to return to the idle state.

As subsystem **44** outputs the sync word of each sync frame to the audio decoder **54** as part of the "audio" referred to in FIG. **2**, it will signal the error status of each audio sync frame to the audio decoder using the sync word. The sync word of audio sync frames in which subsystem **44** knows of no errors will be output as specified by the Dolby AC-3 or Musicam specification, as appropriate. The sync word of audio sync frames in which subsystem **44** knows of errors will be altered relative to the correct sync words. As an example, and in the preferred embodiment, every other bit of the sync word of sync frames to which an error pointer points will be inverted, starting with the most significant bit of the sync word. Thus, the altered AC-3 sync word will be 1010 0001 1101 1101 while the altered Musicam sync word will be 0101 0101 0101. Only the bits of the sync word will be altered. The audio decoder **54** will conceal the audio errors in the sync frame which it receives in which the sync word has been altered in this manner. However, the audio decoder will continue to maintain synchronization with the audio bitstream. Synchronization will be maintained assuming the audio bit rate did not change, and knowing that two sync frame sizes are possible when the audio sample rate is 44.1 ksps.

In accordance with the preferred embodiment, audio decoder **54** will maintain synchronization through sample and bit rate changes if this feature is enabled by the decoder microprocessor **42**. If the microprocessor disables sample rate changes, audio decoder **54** will conceal the audio errors in each sync frame received with a sample rate that does not match the sample rate of the sync frame on which the audio decoder last acquired, and will assume that the sample rate did not change in order to maintain synchronization. The audio decoder is required to process through bit rate changes. If an error in the bit rate information is indicated, e.g., through the use of a cyclic redundancy code (CRC) as well known in the art, audio decoder **54** will assume that the bit rate of the corresponding sync frame is the same bit rate as the previous sync frame in order to maintain synchronization. If the decoder microprocessor **42** has enabled rate changes, the audio decoder **54** will assume that the rates indicated in the sync frame are correct, will process the sync frame, and use the appropriate sync frame size in maintaining synchronization with the audio bitstream.

Demultiplexer and data parsing subsystem **44** will also aid microprocessor **42** in checking that audio data continues to be output at the correct time by resynchronizing with the PTS for some PTS values received. To accomplish this, when a PTS value is received it will be stored in the PTS pointer, along with the audio offset address at which the next sync word is written in audio buffer **50**, if the PTS pointer is not already occupied. In doing this, subsystem **44** will ensure that the next sync word is received at the correct location in the audio PES bitstream. Otherwise, the PTS value will not be stored and subsystem **44** will defer resynchronization until the next successful PTS/DRAM offset address pair is obtained. Subsystem **44** will store the PTS/DRAM offset address pair in the PTS pointer until it begins to output the associated audio sync frame. Once it begins outputting audio data to the audio decoder **54**, subsystem **44** will continue to service the audio decoder's requests for audio data, outputting each audio sync frame in sequence. This will continue until the sync frame pointed to by the PTS pointer is reached. When this occurs, subsystem 44 will stop outputting data to the audio decoder 54 until PCR=PTS+dPTS$_{track}$. This will detect audio timing errors which may have occurred since the last resynchronization by this method.

If PCR>PTS+dPTS$_{acq}$ when subsystem 44 completes output of the previous sync frame, the audio decoder 54 is processing too slow or an undetected error has occurred in a PCR or PTS value. After this error condition, subsystem 44 will flag microprocessor 42, stop the output to the audio decoder 54, clear audio buffer 50 and the pointers, and return to searching for two consecutive sync words separated by the correct number of audio data bytes. If the audio decoder 54 is not requesting data when the buffer read pointer equals the address pointed to by the PTS pointer, an audio processing error has occurred and subsystem 44 will maintain synchronization with the audio stream, clear its audio buffer and pointers, and return to searching for two consecutive audio sync words [State=Frame Sync].

In order to handle errors, subsystem 44 sets a unique error flag for each error condition, which is reset when microprocessor 42 reads the flag. Each error condition which interrupts microprocessor 42 will be maskable under control of the microprocessor. Table 1 lists the various error conditions related to audio synchronization and the response by subsystem 44. In this table, "Name" is a name assigned to each error condition as referenced in the state diagram of FIG. 5. "Definition" defines the conditions indicating that the corresponding error has occurred. "INT" is an interrupt designation which, if "yes", indicates that subsystem 44 will interrupt microprocessor 42 when this error occurs. "Check State" and "Next State" designate the states in which the error will be detected (checked) and the audio processor will enter, respectively, with the symbol ">" that the designated error will be detected when the audio processing state of subsystem 44 is higher than the designated state. The audio processing state hierarchy, from lowest to highest, is:

1. Idle
2. Frame Sync
3. dPTS$_{wait}$
4. PCR$_{acq}$
5. PTS$_{acq}$
6. PTS Sync
7. Track

The symbol "≧" preceding a state indicates that the error will be detected when the audio processing state of subsystem 44 is equal to or higher than the designated state. The designated state(s) indicate(s) that the error will be detected in this state or that the audio processing of subsystem 44 will proceed to this state after the associated actions are carried out. The designation "same" indicates that the audio processing of subsystem 44 will stay in the same state after the associated actions are carried out.

The heading "Buffer Action" indicates whether the audio buffer is to be flushed by setting its read and write pointers to be equal to the base address of the audio buffer. The designation "none" indicates no change from normal audio buffer management.

The heading "Pointer Action" indicates by the term "reset" that the PTS pointer, error pointers or both will be returned to the state specified as if subsystem 44 had been reset. The designation "none" indicates no change from normal pointer management. The designation "see other actions" indicates that other actions under the "Other Actions" heading may indicate a pointer to be set or reset. The "Other Actions" heading states any additional actions required of the subsystem 44 as a result of the error.

TABLE 1

SUMMARY OF ERRORS, EXCEPTIONS, AND ACTIONS.

Name	Definition	Int	Check State	Next State	Buffer Action	Pointer Action	Other Actions
pts_err	PCR > PTS + $dPTS_{acq}$	yes	pts_sync	frame_sync	flush	reset	none
pts_err	PCR > PTS + $dPTS_{acq}$	yes	track	frame_sync	flush	reset	Stop output to Audio Decoder (ADP).
sync_err	Input processor loses sync with input audio frames	yes	>idle	frame_sync	flush	reset	Stop output to ADP.
ov_err	Audio Buffer overflows	yes	≥pts_sync	frame_sync	flush	reset	Input processor maintains synchronization with the audio bitstream. Stop output to ADP.
under_err	Audio Buffer underflows	no	track	same	none	none	Input processor maintains synchronization with the audio bitstream. Stop output to ADP.
fs_err	Input processor reaches Audio PBS data which indicates the audio sample rate has changed since the current PID was acquired	yes	>frame_sync	same	none	none	Continue proccesing as if the audio sample rate had not changed.
fb_err	Input processor receives Audio PES data which indicates the audio bit rate has changed relative to the last audio sync frame reached	yes	>frame_sync	same	none	none	If bit rate changes are enabled, input processor will continue processing, trusting that the bit rate in fact changed and using the appropriate sync frame size to maintain synchronization. If bit rate changes are not enabled, input processor will continue processing using the bit rate indicated by the last audio sync frame received.
pts_miss	Sync word not found due to loss of audio data after a PTS is received	no	≥pts_acquire	same	none	none	None but other error conditions may also apply in this case
pcr_dis1	Input processor reaches a transport packet on the Audio PCR PID with the discontinuity_indicator bit of its adaptation_field set	no	pts_sync	pts acquire	flush	pts:reset error:none	Input processor stops storing PTS values in the PTS pointer until after reception of the next Audio PCR value.
pcr_dis2	Input processor receives a transport packet on the Audio PCR PID with the discontinuity_indicator bit of its adaptation_field set	no	track	same	none	pts:reset error:none	Input processor stops storing PTS values in the PTS pointer until after reception or the next Audio PCR value.
aud_err1a	Audio data of one transport packet of the current input sync frame is lost due to errors	See other actions	>idle	same or frame_sync; see other actions	none	pts:none error:see other actions	Maintain Audio Buffer fullness by advancing the FIFO write pointer by 184 bytes (MPEG), use an error pointer to mark the current sync frame as in error, and continue processing without generating an interrupt. If it is possible that more than one audio sync word was lost with the missing audio transport packet, such as when supporting Musicam Layer II at less than 64 kbps or AC-3 at less than 48 kbps, return to the Frame Sync state and generate an interrupt. If the next audio sync word is not received when expected, begin a byte-by-byte search for the audio sync word during the reception of subsequent audio data. Once the sync byte search is started, stop storing audio data in the buffer until the sync word in found. Do not store the first byte examined during the search. Resume storing audio data when the sync byte is found, starting with the sync word itself. If the sync word is not found during the first 184 bytes searched, return to the Frame Sync state[1] and generate an interrupt

TABLE 1-continued

SUMMARY OF ERRORS, EXCEPTIONS, AND ACTIONS.

Name	Definition	Int	Check State	Next State	Buffer Action	Pointer Action	Other Actions
aud_err1b	Audio data of one transport packet of the current input sync frame is lost due to errors after aud_err1a has occurred during the same input sync frame	yes	>idle	frame_sync	flush	pts:reset error:none	none
aud_err2	Audio data of more than one transport packet of the current input sync frame is lost due to errors	yes	>idle	frame_sync	flush	pts:reset error:see other actions	Use an error pointer to mark the current sync frame as in error.
ptrs_full	Audio data of one transport packet is lost while Error Mode is Unprotected	yes	≧pts_sync	frame_sync	flush	reset	Input processor maintains synchronization with the audio bitstream. Stop output to ADP.

[1] To implement the above error processing for MPEG or DigiCipher II implementations, the Input Processor can maintain an audio frame byte count by:
setting a counter's value so the sync frame size in bytes as each sync word is received,
decrementing the counter as each received audio byte is stored in the Audio Buffer (FIFO),
decrementing the counter by 184 bytes when a single audio transport packet is lost to compensate for the advancement of the FIFO write pointer by 184,
incrementing the counter by the smaller of the two sync frame sizes in bytes corresponding to the current bit rate if the above decrement resulted in a negative counter value (indicating the lost transport packet possibly contained the next audio sync word and accounting for the possibility that the audio sample rate is 44.1 Ksps and the sync frame size has changed from the larger value to the smaller value),
returning to the Frame Sync state if the above increment resulted in a counter value which was still negative (indicating the lost transport packet possibly contained more than one audio sync word), and
beginning the byte-by-byte sync word search when the counter is zero.

As indicated above, the demultiplexing and data parsing subsystem 44 of FIG. 2 maintains several pointers to support audio processing. The PTS pointer is a set of parameters related to a PTS value, specifically a PTS value, a DRAM offset address, and a validity flag. In the illustrated embodiment, the PTS value comprises the 17 least significant bits of the PTS value received from the audio PES header. This value is associated with the audio sync frame pointed to by the pointer's DRAM offset address field. The use of 17 bits allows this field to specify a 1.456 second time window ($(2^{17}-1)/90$ kHz), which exceeds the maximum audio time span which the audio buffer 50 is sized to store.

The DRAM offset address maintained by the PTS pointer is a 13-bit offset address, relative to the audio buffer base address, into the DRAM at which the first byte of the audio sync frame associated with the pointer's PTS value is stored. The 13 bits allows the pointer to address an audio buffer as large as 8192 bytes.

The PTS pointer validity flag is a one-bit flag indicating whether or not this PTS pointer contains a valid PTS value and DRAM offset address. Since MPEG does not require PTS values to be transported more often than every 700 milliseconds, subsystem 44 may find itself not having a valid PTS value for some intervals of time.

After the decoder is reset, the valid flag of the PTS pointer is set to invalid. When a new PTS value is received, if the valid flag is set, the newly received PTS value is ignored. If the valid flag is not set, the newly received PTS value is stored into the PTS pointer but its valid flag is not yet set to valid. After a new PTS value is stored into the PTS pointer, the processing of audio data is continued and each audio data byte is counted. If the next audio sync frame is received and placed into the buffer correctly, the DRAM offset address (which corresponds to the buffer address into which the first byte of the sync word of this sync frame is stored) is stored into the pointer's DRAM offset address field. Then, the pointer's valid flag is set to valid. The next audio sync frame is received and placed into the buffer correctly when no data is lost for any reason between reception of the PTS value and reception of a subsequent sync word before too many audio bytes (i.e., the number of audio bytes per sync frame) are buffered. If the next audio, sync frame is not received or placed into the buffer correctly, the valid flag is not set to valid.

After the PTS pointer is used to detect any audio timing errors which may have occurred since the last resynchronization, the valid flag is set to invalid to allow subsequent PTS pointers to be captured and used. This occurs whether the PTS pointer is in the PTS sync or tracking state.

The error pointers are parameters related to an audio sync frame currently in the buffer and known to contain errors. The error pointers comprise a DRAM offset address and a validity flag. The DRAM offset address is a 13-bit offset address, relative to the audio buffer base address, into the DRAM at which the first byte of the audio sync frame known to contain errors is stored. Thirteen bits allows the pointer to address an audio buffer as large as 8192 bytes. The validity flag is a one-bit flag indicating whether or not this error pointer contains a valid DRAM offset address. When receiving data from a relatively error free medium, subsystem 44 will find itself not having any valid error pointers for some intervals of time.

Subsystem 44 is required to maintain a total of two error pointers and one error mode flag. After reset, the validity flag is set to invalid and the error mode is set to "protected." When a sync word is placed into the audio buffer, if the valid flag of one or more error pointers is not set, the buffer address of the sync word is recorded into the DRAM offset address of one of the invalid error pointers. At the same time, the error mode is set to protected. If the validity flag of both error pointers is set when a sync word is placed into the buffer, the error mode is set to unprotected but the DRAM offset address of the sync word is not recorded.

When audio data is placed into the buffer and any error is discovered in the audio data, such as due to the loss of an audio transport packet or the reception of audio data which has not been properly decrypted, subsystem 44 will revert to the PTS acquire state if the error mode is unprotected. Otherwise, the validity bit of the error pointer which contains the DRAM offset address of the sync word which starts the sync frame currently being received is set. In the rare event that an error is discovered in the data for an audio sync frame during the same clock cycle that the sync word for the sync frame is removed from the buffer, the sync word will be corrupted as indicated above to specify that the sync frame is known to contain an audio error. At the same time, the validity bit is cleared such that it does not remain set after the sync frame has been output. This avoids the need to reset subsystem 44 in order to render the pointer useful again.

When audio data is being removed from the audio buffer, the sync word is corrupted if the DRAM offset address of any error pointer matches that of the data currently being removed from the buffer. At the same time, the validity bit is set to invalid.

The decoder of FIG. 2 also illustrates a video buffer 58 and video decoder 52. These process the video data at the same time the audio data is being processed as described above. The ultimate goal is to have the video and audio data output together at the proper time so that the television signal can be reconstructed with proper lip synchronization.

FIG. 4 is a block diagram illustrating the demultiplexing and data parsing subsystem 44 of FIG. 2 in greater detail. After the transport packets are input via terminal 40, the PID of each packet is detected by circuit 70. The detection of the PIDs enables demultiplexer 72 to output audio packets, video packets and any other types of packets carried in the data stream, such as packets carrying control data, on separate lines.

The audio packets output from demultiplexer 72 are input to the various circuits necessary to implement the audio processing as described above. Circuit 74 modifies the sync word of each audio frame known to contain errors. The modified sync words are obtained using a sync word inverter 78, which inverts every other bit in the sync words output from a sync word, PCR and PTS detection circuit 80, in the event that the audio frame to which the sync word corresponds contains an error. Error detection is provided by error detection circuit 76.

The sync word, PCR and PTS detection circuit 80 also outputs the sync word for each audio frame to an audio sample and bit rate calculator 86. This circuit determines the audio sample and bit rate of the audio data and passes this information to decoder microprocessor 42 via data bus 88.

The PCR and PTS are output from circuit 80 to a lip sync and output timing compensator 82. Circuit 82 also receives the dPTS values from microprocessor 42, and adds the appropriate values to the PTS in order to provide the necessary delay for proper lip synchronization. Compensator 82 also determines if the delayed presentation time is outside of the acceptable range with respect to the PCR, in which case an error has occurred and resynchronization will be required.

Buffer control 84 provides the control and address information to the audio output buffer 50. The buffer control 84 is signaled by error detection circuit 76 whenever an error occurs that requires the temporary suspension of the writing of data to the buffer. The buffer control 84 also receives the delay values from lip sync and output timing compensator 82 in order to control the proper timing of data output from the buffer.

FIG. 5 is a state diagram illustrating the processing of audio data and response to errors as set forth in Table 1. The idle state is represented by box 100. Acquisition of the audio data occurs during the frame sync state 102. The dPTS-wait state is indicated by box 104. Boxes 106, 108 and 110 represent the PCR_{acq}, PTS_{acq}, and PTS sync states, respectively. Once audio synchronization has occurred, the signal is tracked as indicated by the tracking state of box 112. The outputs of each of boxes 104, 106, 108, 110 and 112 indicate the error conditions that cause a return to the frame synchronization state 102. The error PCR DIS1 during the PTS sync state 110 will cause a return to the PTS acquire state, as indicated in the state diagram of FIG. 5.

It should now be appreciated that the present invention provides methods and apparatus for acquiring and processing errors in audio data communicated via a transport packet scheme. Transport packet errors are handled while maintaining audio synchronization. During such error conditions, the associated audio errors are concealed. Corrupted data in an audio frame is signaled by altering the sync pattern associated with the audio frame. PTS's are used to check the timing of processing and to correct audio timing errors.

Although the invention has been described in connection with various specific embodiments, it should be appreciated and understood that numerous adaptations and modifications may be made thereto, without departing from the spirit and scope of the invention as set forth in the claims.

We claim:

1. A method for processing digital audio data from a packetized data stream carrying digital television information in a succession of fixed length transport packets, each of said packets including a packet identifier (PID), some of said packets containing a program clock reference (PCR) value for synchronizing a decoder system time clock (STC), and some of said packets containing a presentation time stamp (PTS) indicative of a time for commencing the output of associated data for use in reconstructing a television signal, said method comprising the steps of:

monitoring the PID's for the packets carried in said data stream to detect audio packets, some of said audio packets carrying an audio PTS;

storing audio data from the detected audio packets in a buffer for subsequent output;

monitoring the detected audio packets to locate audio PTS's;

comparing a time derived from said STC with a time derived from the located audio PTS's to determine whether said audio packets are too early to decode, too late to decode, or ready to be decoded; and

adjusting the time at which said stored audio data is output from said buffer on an ongoing basis in response to said comparing step.

2. A method in accordance with claim 1 wherein a PTS pointer is provided to maintain a current PTS value and an address of said buffer identifying where a portion of audio data referred to by said current PTS is stored, said timing adjustment being provided by the further steps of:

replacing said PTS value in said PTS pointer with a new current PTS value after data stored at said address has been output from said buffer;

replacing said address in said PTS pointer with a new address corresponding to a portion of audio data referred to by said new current PTS value;

suspending the output of data from said buffer when said new address is reached; and

recommencing the output of data from said buffer when said decoder system time clock reaches a presentation time derived from said new current PTS value.

3. A method in accordance with claim 2 wherein said presentation time is determined from the sum of said new current PTS value and an offset value that provides proper lip synchronization by accounting for a video signal processing delay.

4. A method in accordance with claim 1 wherein the time at which the audio data is output from said buffer is dependent on an offset value added to said PTS for providing proper lip synchronization by accounting for a video signal processing delay.

5. A method in accordance with claim 1 comprising the further steps of:

examining the detected audio packets to locate the occurrence of at least one audio synchronization word therein for use in achieving a synchronization condition prior to locating said audio PTS's;

commencing a reacquisition of said synchronization condition if said comparing step determines that said audio packets are too late to decode.

6. A method in accordance with claim 5 wherein two consecutive audio synchronization words with a correct number of audio data bytes in between define an audio frame, said audio frame including only one of said two consecutive audio synchronization words, said method comprising the further steps of:

detecting the occurrence of errors in said audio packets;

upon detecting a first audio packet of a current audio frame containing an error, advancing a write pointer for said buffer by the maximum number of payload bytes (N) contained in one of said fixed length transport packets and designating said current audio frame as being in error;

monitoring the detected audio packets of said current audio frame for the next audio synchronization word after said error has been detected, and if said synchronization word is not received where expected in the audio stream, discarding subsequent audio data while searching for said synchronization word rather than storing the subsequent audio data into said buffer;

resuming the storage of audio data in said buffer upon detection of said next audio synchronization word if said next audio synchronization word is located within N bytes after the commencement of the search therefor; and

if said next audio synchronization word is not located within said N bytes after the commencement of the search therefor, commencing a reacquisition of said synchronization condition.

7. A method in accordance with claim 6 comprising the further step of concealing television audio errors whenever the audio data from which said television audio is being reconstructed is in error.

8. A method in accordance with claim 7 wherein:

a current audio frame is designated as being in error by altering the audio synchronization word for that frame; and

said concealing step is responsive to an altered synchronization word for concealing audio associated with the corresponding audio frame.

9. A method for processing digital audio data from a packetized data stream carrying digital television information in a succession of transport packets having a fixed length of N bytes, each of said packets including a packet identifier (PID), some of said packets containing a program clock reference (PCR) value for synchronizing a decoder system time clock, and some of said packets containing a presentation time stamp (PTS) indicative of a time for commencing the output of associated data for use in reconstructing a television signal, said method comprising the steps of:

monitoring the PID's for the packets carried in said data stream to detect audio packets;

examining the detected audio packets to locate the occurrence of audio synchronization words for use in achieving a synchronization condition, each two consecutive audio synchronization words defining an audio frame therebetween;

monitoring the detected audio packets after said synchronization condition has been achieved to locate an audio PTS;

searching the detected audio packets after locating said audio PTS to locate the next audio synchronization word;

storing audio data following said next audio synchronization word in a buffer;

detecting the occurrence of errors in said audio packets;

upon detecting a first audio packet of a current audio frame containing an error, advancing a write pointer for said buffer by N bytes and designating said current audio frame as being in error;

monitoring the detected audio packets of said current audio frame for the next audio synchronization word after said error has been detected, and if said synchronization word is not received where expected in the audio stream, discarding subsequent audio data while searching for said synchronization word rather than storing the subsequent audio data into said buffer;

resuming the storage of audio data in said buffer upon detection of said next audio synchronization word if said next audio synchronization word is located within N bytes after the commencement of the search therefor; and

if said next audio synchronization word is not located within said N bytes after the commencement of the search therefor, commencing a reacquisition of said synchronization condition.

10. A method in accordance with claim 9 comprising the further step of concealing television audio errors whenever the audio data from which said television audio is being reconstructed is in error.

11. A method in accordance with claim 10 wherein:

a current audio frame is designated as being in error by altering the audio synchronization word for that frame; and

said concealing step is responsive to an altered synchronization word for concealing audio associated with the corresponding audio frame.

12. A method in accordance with claim 9 wherein said audio data includes information indicative of an audio sample rate and audio bit rate, at least one of said audio sample rate and audio bit rate being variable, said method comprising the further step of attempting to maintain synchronization of said audio packets during a rate change indicated by said audio data by:

ignoring a rate change indicated by said audio data on the assumption that the rate has not actually changed;

concealing the audio frame containing the data indicative of an audio sample rate change while attempting to maintain said synchronization condition; and

commencing a reacquisition of said synchronization condition if said condition cannot be maintained.

13. A method in accordance with claim 9 wherein said audio data includes information indicative of an audio sample rate and audio bit rate, at least one of said audio sample rate and audio bit rate being variable, said method comprising the further step of attempting to maintain synchronization of said audio packets during a rate change indicated by said audio data by:

processing said audio data in accordance with a new rate indicated by said audio data in the absence of an error indication pertaining to the audio frame containing the new rate, while attempting to maintain said synchronization condition;

processing said audio data without changing the rate if an error indication pertains to the audio frame containing the new rate, while concealing the audio frame to which said error condition pertains and attempting to maintain said synchronization condition; and

commencing a reacquisition of said synchronization condition if said condition cannot be maintained.

14. Apparatus for acquiring audio information carried by a packetized data stream and processing errors therein, comprising:

means for detecting audio transport packets in said data stream;

means for recovering audio data from said detected audio transport packets for storage in a buffer;

means for locating an audio presentation time stamp (PTS) in said detected audio transport packets;

means responsive to said PTS for commencing the output of audio data from said buffer at a specified time;

means for monitoring the detected audio transport packets after the output of audio data from said buffer has commenced, to locate subsequent audio PTS's;

means for comparing a time derived from a decoder system time clock (STC) to a time derived from the subsequent audio PTS's to determine whether audio data stored in said buffer is too early to decode, too late to decode, or ready to be decoded; and

means responsive to said comparing means for adjusting the time at which said stored audio data is output from said buffer.

15. Apparatus in accordance with claim 14 further comprising:

means for maintaining a PTS pointer with a current PTS value and an address of said buffer identifying where a portion of audio data referred to by said current PTS is stored;

means for replacing said PTS value in said PTS pointer with a new current PTS value after data stored at said address has been output from said buffer, and for replacing said address in said PTS pointer with a new address corresponding to a portion of audio data referred to by said new current PTS value;

means responsive to said PTS pointer for suspending the output of data from said buffer when said new address is reached; and

means for recommencing the output of data from said buffer at a time derived from said new current PTS value.

16. Apparatus in accordance with claim 15 further comprising:

means for concealing error in an audio signal reproduced from data output from said buffer and reestablishing the detection of said audio transport packets if the time derived from said new current PTS value is outside a predetermined range.

17. Apparatus in accordance with claim 14 wherein said audio transport packets each contain a fixed number N of payload bytes, said packets being arranged into successive audio frames commencing with an audio synchronization word, said apparatus further comprising:

means for detecting the occurrence of errors in said audio packets;

means for advancing a write pointer for said buffer by N bytes and designating a current audio frame as being in error upon detecting an error in an audio transport packet of said current audio frame;

means for monitoring the detected audio transport packets of said current audio frame for the next audio synchronization word after said error has been detected, and if said synchronization word is not received where expected in the audio stream, discarding subsequent audio data while searching for said synchronization word rather than storing the subsequent audio data into said buffer;

means for resuming the storage of audio data in said buffer upon detection of said next audio synchronization word if said next audio synchronization word is located within said fixed number N of bytes after the commencement of the search therefor; and

means for reestablishing the detection of said audio transport packets if said next audio synchronization word is not located within said fixed number N of bytes after the commencement of the search therefor.

18. Apparatus in accordance with claim 17 further comprising:

means for concealing error in an audio signal reproduced from data output from said buffer when the data output from said buffer is in error.

19. Apparatus in accordance with claim 18 further comprising:

means for altering the audio synchronization word associated with a current audio frame to designate that frame as being in error;

wherein said concealing means are responsive to altered synchronization words for concealing errors in audio associated with the corresponding audio frame.

20. Apparatus for acquiring audio information carried by a packetized data stream and processing errors therein, comprising:

means for detecting audio transport packets in said data stream, said packets being arranged into successive audio frames commencing with an audio synchronization word;

means responsive to said synchronization words for obtaining a synchronization condition enabling the recovery of audio data from said detected audio transport packets for storage in a buffer;

means for detecting the presence of errors in said audio data;

means responsive to said error detecting means for controlling the flow of data through said buffer when an error is present, to attempt to maintain said synchronization condition while masking said error; and

means for reestablishing the detection of said audio transport packets if said controlling means cannot maintain said synchronization condition.

21. Apparatus in accordance with claim 20 wherein said audio transport packets each contain a fixed number N of payload bytes, and said means responsive to said error detecting means comprise:

means for advancing a write pointer for said buffer by said fixed number N of bytes and designating a current audio frame as being in error upon the detection of an error in an audio transport packet thereof;

means for monitoring the detected audio transport packets of said current audio frame for the next audio synchronization word after said error has been detected, and if said synchronization word is not received where expected in the audio stream, discarding subsequent audio data while searching for said synchronization word rather than storing the subsequent audio data into said buffer; and

means for resuming the storage of audio data in said buffer upon detection of said next audio synchronization word if said next audio synchronization word is located within said fixed number N of bytes after the commencement of the search therefor.

22. Apparatus in accordance with claim 20 further comprising:

means for concealing error in an audio signal reproduced from data output from said buffer when the data output from said buffer is in error.

23. Apparatus in accordance with claim 22 further comprising:

means for altering the audio synchronization word associated with an audio frame containing a data error to designate that frame as being in error;

wherein said concealing means are responsive to altered synchronization words for concealing errors in audio associated with the corresponding audio frame.

24. A method for managing errors in data received in bursts from a packetized data stream carrying digital information in a succession of fixed length transport packets, at least some of said packets containing a presentation time stamp (PTS) indicative of a time for commencing the fixed rate presentation of presentation units from a buffer into which they are temporarily stored upon receipt, said method comprising the steps of:

monitoring received packets to locate associated PTS's, said received packets carrying presentation units to be presented;

synchronizing the presentation of said presentation units from said buffer to a system time clock (STC) associated with the packetized data stream using timing information derived from the PTS's located in said monitoring step; and

identifying discontinuity errors resulting from a loss of one or more transmitted packets between successive ones of the received packets and, if a discontinuity of no more than one packet is identified, advancing a write pointer of said buffer by a suitable number of bits to compensate for the discontinuity, while maintaining the synchronization of said presentation with respect to said STC.

25. A method in accordance with claim 24 wherein said transport packets each contain a fixed number N of payload bytes, said method comprising the further steps of:

advancing said write pointer by said fixed number N of bytes upon the detection of a discontinuity error;

continuing said monitoring step after said discontinuity error has been detected in order to search for a synchronization word, and if said synchronization word is not located where expected, discarding subsequent presentation units while searching for said synchronization word rather than storing said subsequent presentation units in said buffer; and

resuming the storage of presentation units in said buffer upon the detection of said synchronization word if said synchronization word is located within said fixed number N of bytes after the commencement of the search therefor.

* * * * *

US005892754A

United States Patent [19]

Kompella et al.

[11] **Patent Number: 5,892,754**

[45] **Date of Patent: Apr. 6, 1999**

[54] **USER CONTROLLED ADAPTIVE FLOW CONTROL FOR PACKET NETWORKS**

[75] Inventors: **Vachaspathi P. Kompella**, Cary; **James P. Gray**, Chapel Hill; **Frank D. Smith**, Chapel Hill; **Kevin Jeffay**, Chapel Hill, all of N.C.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **660,317**

[22] Filed: **Jun. 7, 1996**

[51] **Int. Cl.**^{6} **H04L 12/56**

[52] **U.S. Cl.** **370/236**; 370/252; 370/406; 370/410

[58] **Field of Search** 370/231, 232, 370/235, 236, 252, 253, 400, 406, 410

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,317,563	5/1994	Oouchi et al.	370/253
5,367,523	11/1994	Chang et al.	370/253
5,442,624	8/1995	Bonomi et al.	370/253
5,636,345	6/1997	Valdevit	370/253
5,701,292	12/1997	Chiussi et al	370/253

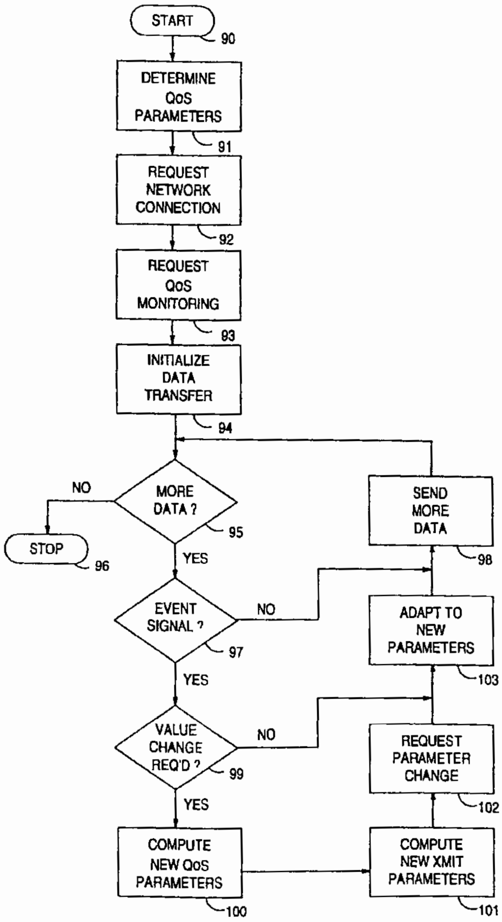
Primary Examiner—Min Jung

Attorney, Agent, or Firm—Gerald R Woods; The University of North Carolina at Chapel Hill

[57] **ABSTRACT**

A flow control system for packet transmission networks is centered in the user applications supplying data to the network. Changes in control are responsive to changes in the transmission parameters of the network, measured in the network and transmitted to the user application. The user application specifies desired ranges of Quality of Service parameters and, when the measured network parameters fall outside of the desired range, the user application modifies the transmission strategy to match the available transmission parameters. Measurements of network parameters are made over a pre-selected observation period to average the values of the transmission parameters.

25 Claims, 5 Drawing Sheets



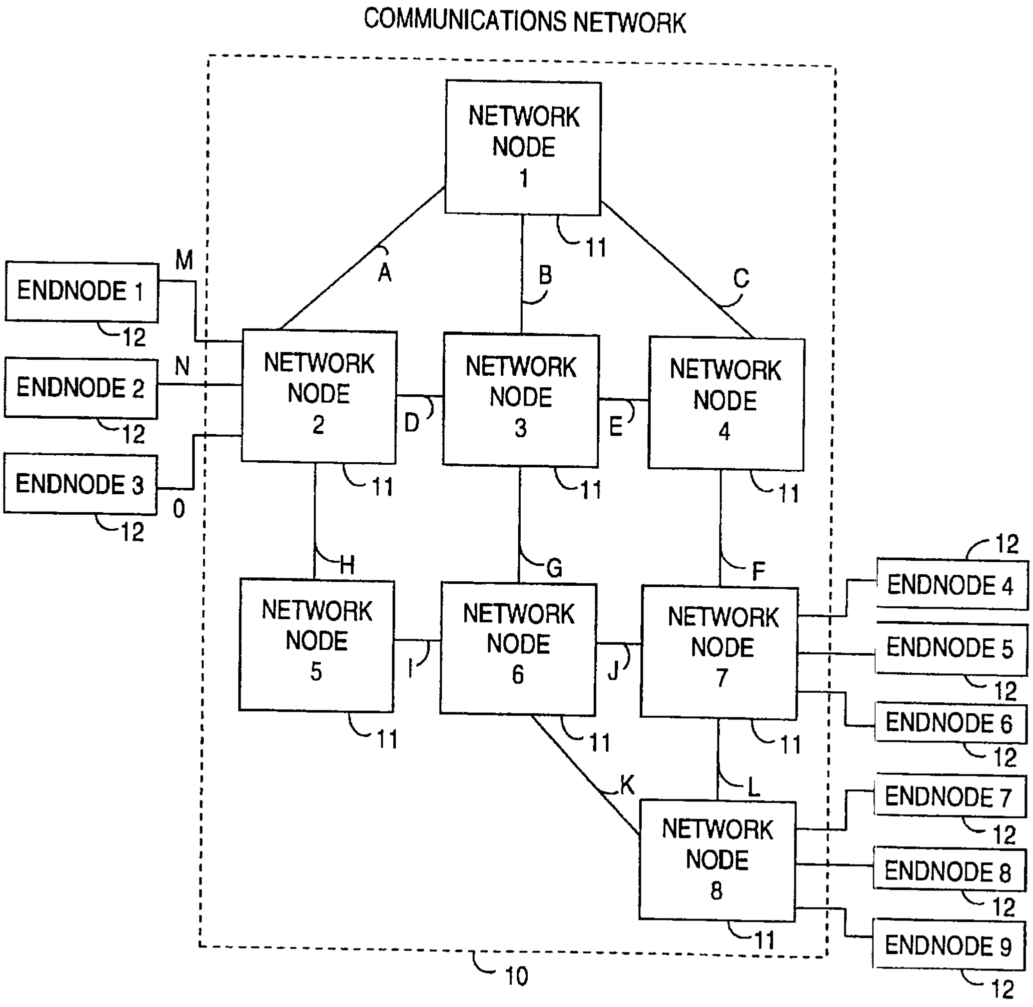


FIG. 1
(PRIOR ART)

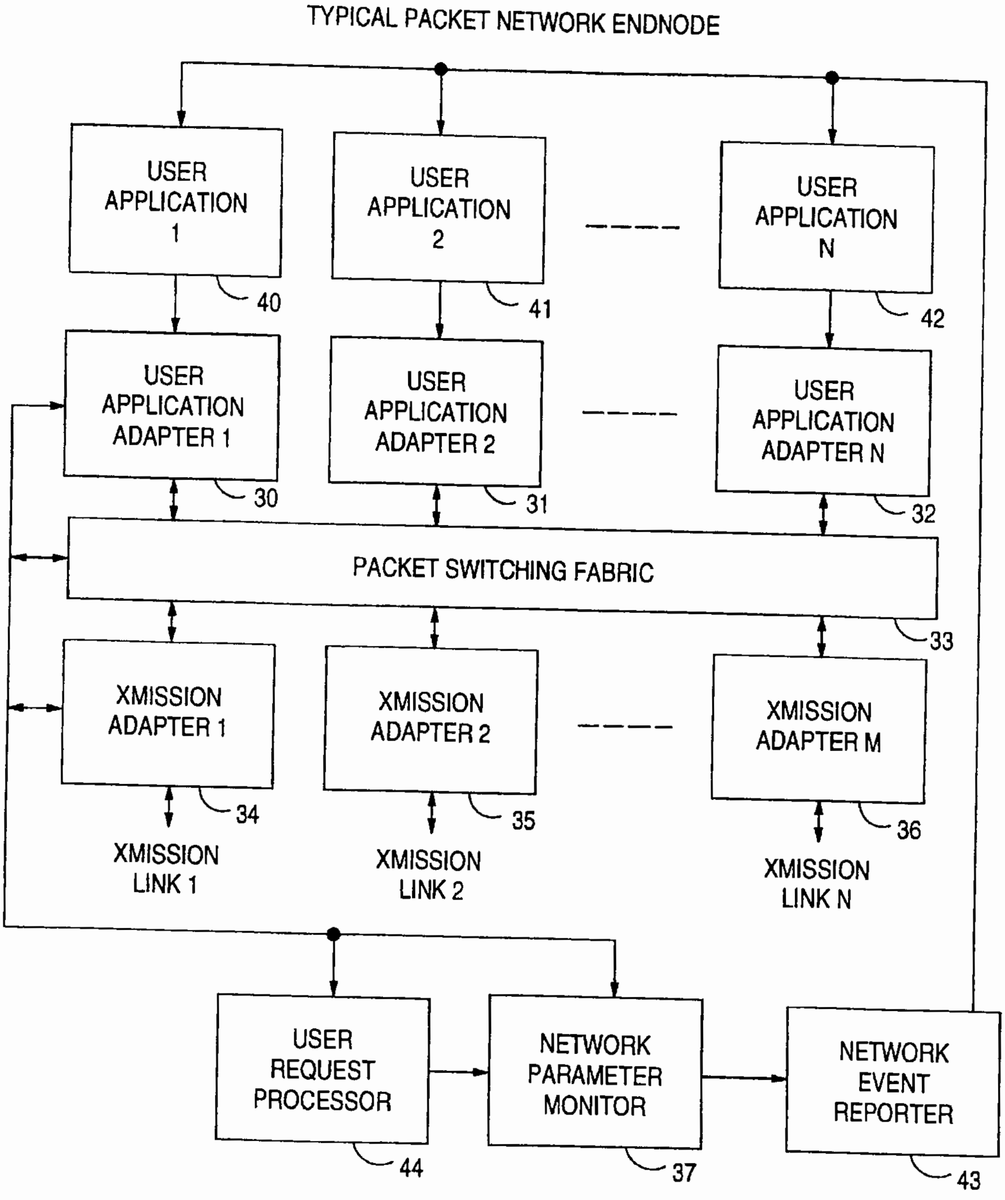

TYPICAL PACKET NETWORK ENDNODE
USER APPLICATION 1
40
USER APPLICATION 2
41
USER APPLICATION N
42
USER APPLICATION ADAPTER 1
30
USER APPLICATION ADAPTER 2
31
USER APPLICATION ADAPTER N
32
PACKET SWITCHING FABRIC
33
XMISSION ADAPTER 1
34
XMISSION ADAPTER 2
35
XMISSION ADAPTER M
36
XMISSION LINK 1
XMISSION LINK 2
XMISSION LINK N
USER REQUEST PROCESSOR
44
NETWORK PARAMETER MONITOR
37
NETWORK EVENT REPORTER
43

FIG. 2

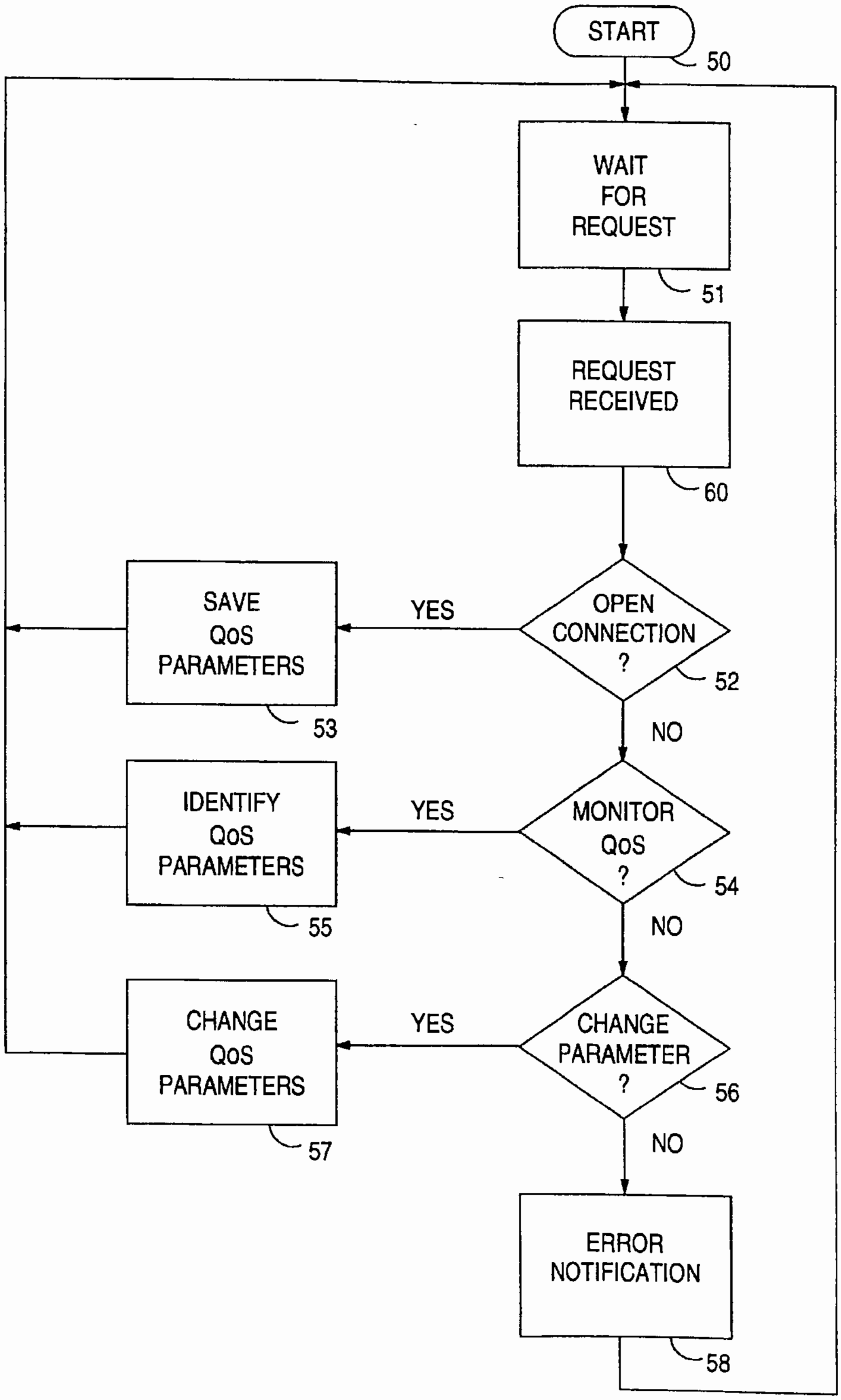

START
50
WAIT FOR REQUEST
51
REQUEST RECEIVED
60
OPEN CONNECTION ?
52
YES
SAVE QoS PARAMETERS
53
NO
MONITOR QoS ?
54
YES
IDENTIFY QoS PARAMETERS
55
NO
CHANGE PARAMETER ?
56
YES
CHANGE QoS PARAMETERS
57
NO
ERROR NOTIFICATION
58

FIG. 3

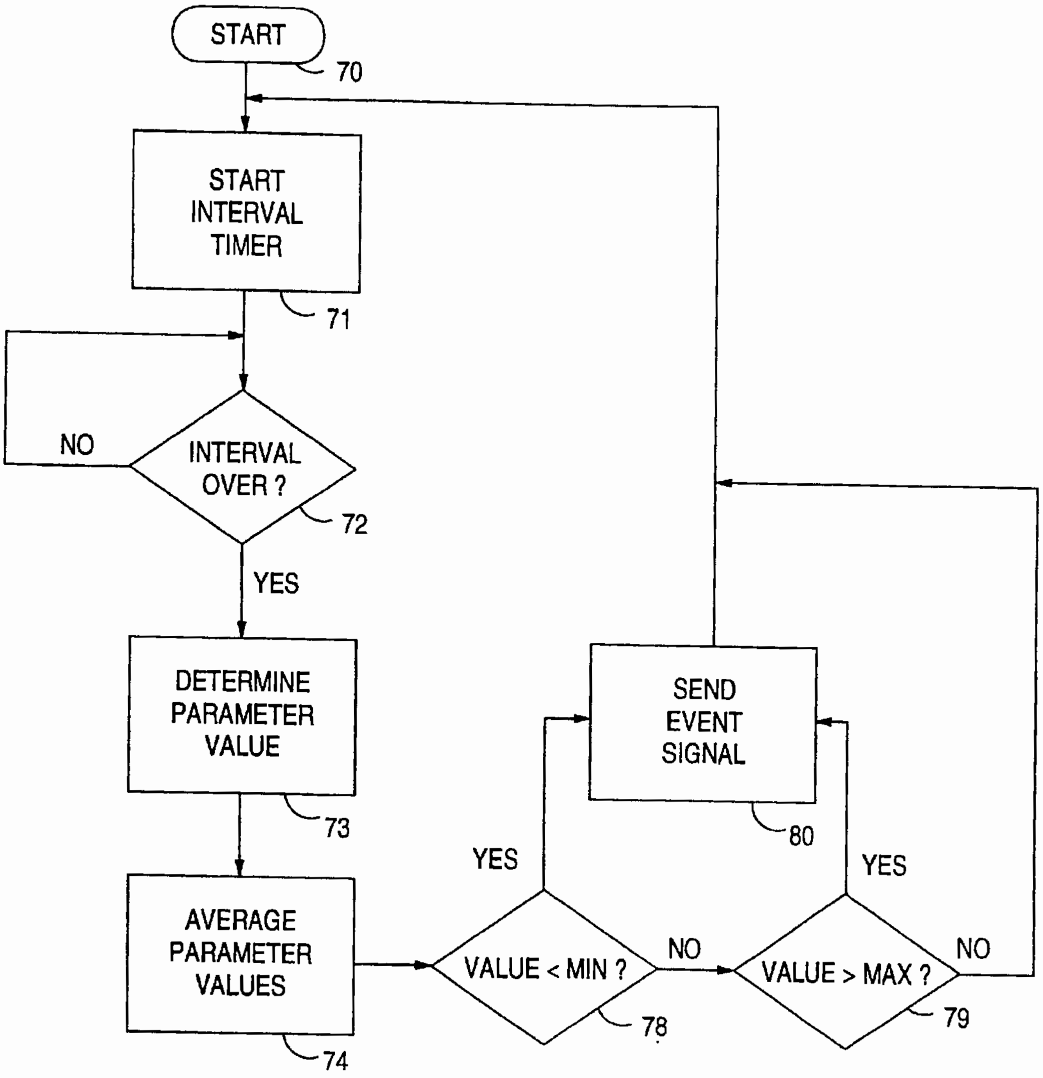

START
70
START INTERVAL TIMER
71
NO
INTERVAL OVER ?
72
YES
DETERMINE PARAMETER VALUE
73
AVERAGE PARAMETER VALUES
74
VALUE < MIN ?
78
YES
NO
SEND EVENT SIGNAL
80
VALUE > MAX ?
79
YES
NO

FIG. 4

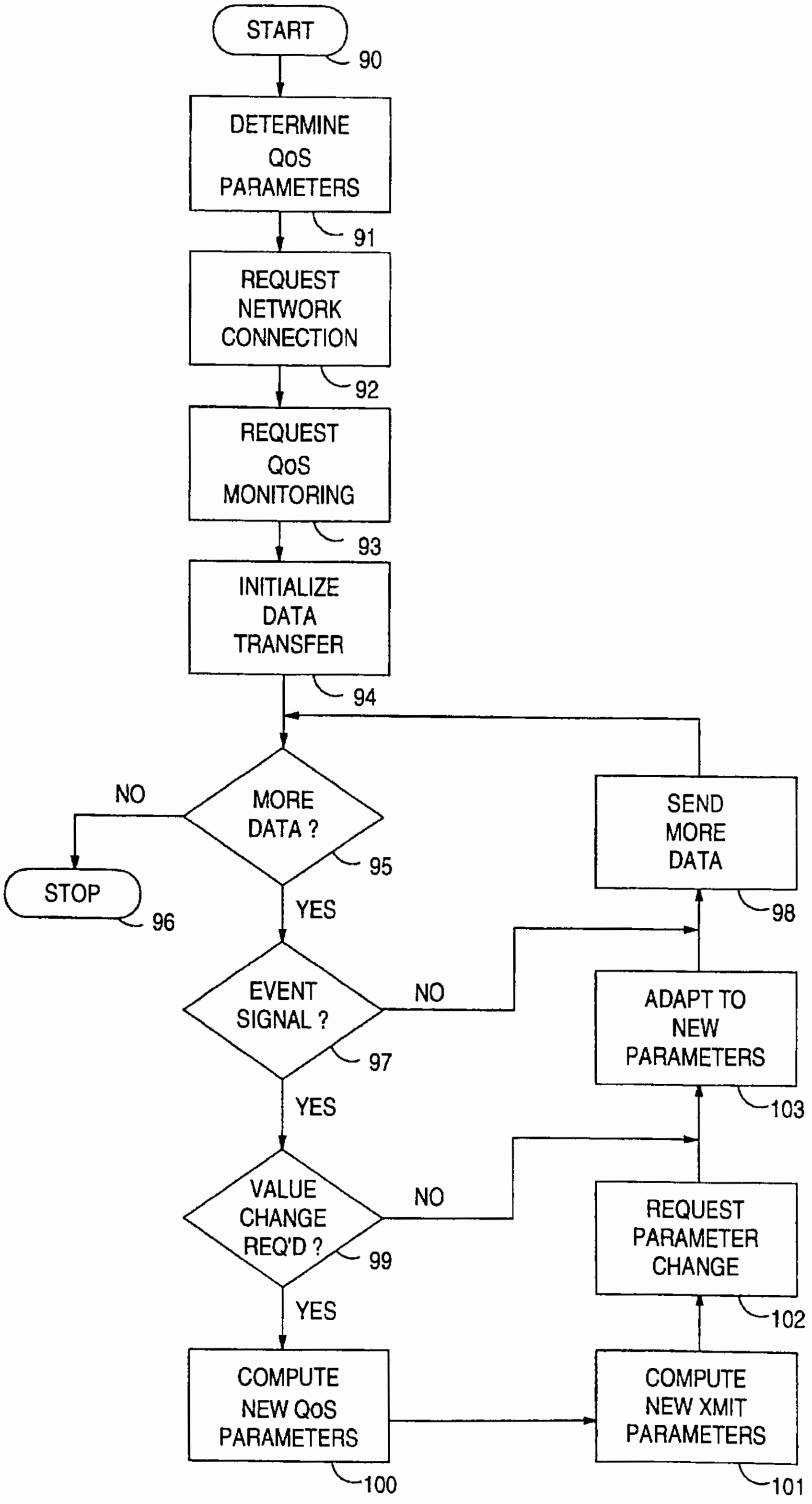

START
90
DETERMINE QoS PARAMETERS
91
REQUEST NETWORK CONNECTION
92
REQUEST QoS MONITORING
93
INITIALIZE DATA TRANSFER
94
MORE DATA ?
95
NO
STOP
96
YES
EVENT SIGNAL ?
97
NO
YES
VALUE CHANGE REQ'D ?
99
NO
YES
COMPUTE NEW QoS PARAMETERS
100
COMPUTE NEW XMIT PARAMETERS
101
REQUEST PARAMETER CHANGE
102
ADAPT TO NEW PARAMETERS
103
SEND MORE DATA
98

FIG. 5

USER CONTROLLED ADAPTIVE FLOW CONTROL FOR PACKET NETWORKS

TECHNICAL FIELD

This invention relates to packet communications systems and, more particularly, to traffic flow control in such systems.

BACKGROUND OF THE INVENTION

Numerous types of flow control have been devised for packet transmission systems. Such control mechanisms regulate a user application's behavior with respect to the transmission of data into the network and are typically implemented in the operating system and in the network protocol software. For example, if a user application attempts to send a large quantity of data to the network, and the network is overloaded, the network software buffers store the data that cannot be transmitted and attempts to deliver the data that can be transmitted. When there are no more buffers available, or if the buffers allocated to this application have been exhausted, the operating system typically suspends the user application, preventing the application from transmitting any more data until buffer space becomes available. The network protocol may also slow down the transmission of data because the receiving application cannot keep up with the data flow. These types of control mechanisms are known as flow control mechanisms. These network-based mechanisms are clearly not optimized for any particular user application, but are simply imposed on all user applications by the network.

Some flow control mechanisms in network protocols, such as the Transmission Control Protocol (TCP), are window oriented. That is, the receiving application will permit the transmitting user application to send only a certain amount of data (a "window") and, until the receiving application opens up the window further, the sending application is not allowed to transmit data. In TCP, the sending station backs off from its transmitting rate exponentially if acknowledgments from the receiving application do not arrive fast enough (before a local timer expires). These types of flow control mechanisms operate independently of the applications and often do not interact well with application requirements.

Another type of flow control mechanism is the so-called rate-based flow control, and includes High Performance Routing (HPR) in the Advanced Peer-to-Peer Network (APPN). These rate-based flow control mechanisms monitor the round-trip time of data flow and adjust the rate at which data is released from the transmitting application in response to the flow rate. That is, the rate-based flow control mechanism only allows data to enter into the network at a rate it (the network) has deemed sustainable over the long term, usually based on measurements of a test message sent to the receiving application. The application is thus constrained to transmit at this predetermined rate over the long term, even though transient rates may be greater due to buffering. Clearly, these constraints on the sending application are never optimal for the particular data being transmitted.

Having the network software act as a moderator of data flow into the network has significant advantages. The network is able to monitor its own behavior and thus determine overload situations. As taught in U.S. Pat. No. 5,326,523, the adaptive rate-based (ARB) flow control mechanism in HPR allows the data outflow to be controlled by the congestion status of the network, in effect allowing the data to flow out of a node at a rate commensurate with the actual congestion experienced in the network. For time-insensitive applications such as E-mail and file transfer, the rate-based adaptation of the network is excellent, relieving network overload without adding significant complexity to the user applications.

Unfortunately, for time-sensitive applications such as multimedia, audio and video conferencing and video-on-demand, network-implemented flow control mechanisms are totally inadequate. For example, if a video source, transmitting at thirty frames per second, is network flow controlled to deliver only twenty frames per second, the receiving application can either play the twenty frames that it receives (with gaps), and discard the ten frames that arrive late, or it may attempt to play all of the frames, but at the price of introducing substantial latency into the system. Neither of these results is particularly desirable since the quality or real time delivery of the picture is significantly degraded.

Many types of data applications are capable of performing satisfactorily at a number of different operating points in the multidimensional space defined by the network transmission parameters such as throughput, latency, latency variations, i.e., jitter, error rates, and so forth. For simplicity, the terms "transmission parameters" and "Quality of Service parameters" are used interchangeably in this application. In the above example, the video source could transmit fewer frames per second, obviating the need for transmitting the ten frames later, and providing a picture quality better than transmitting twenty of thirty frames and discarding the other ten frames. In general, user applications are capable of adapting to changing network conditions such as congestion in a variety of different ways such as using different coding, using data compression, different image sizes, different color representation, different frame rates, forward error correction, and so forth. None of these adaptations to network conditions can be used when adaptation is controlled solely by the network software. Similarly audio signals can be sensibly adapted to different transmission conditions by re-scaling the audio signals.

SUMMARY OF THE INVENTION

In accordance with the illustrative embodiment of the present invention, the state of congestion in a packet communications system is made available to the user applications utilizing that communications system. That is, the network facilities monitor the network so as to obtain the best possible information concerning the values of all of the network transmission parameters, including throughput, latency, jitter and so forth. However, since the network does not have the best information concerning how best to adapt to changes in these transmission parameters, these transmission parameter values are made available to each user application. More particularly, a programming interface with user applications is provided with extensions which enable the network software to inform the user applications of the values of these transmission parameters. The user application can be provided with a system call to inquire about the network transmission parameters, or the network software can asynchronously supply the user application with signals indicating the occurrence of events affecting transmission parameters. These event signals can be handled like other external event signals such as timer events, semaphore events, user signals, and so forth, which are already part of most operating systems. The latter technique, advising user applications of transmission parameter affecting events, is the preferred alternative since the user application may not know the best times to query the network for transmission parameters.

In accordance with one feature of the present invention, a certain amount of hysteresis in introduced into the event reporting process to prevent the application from responding to transient changes which do not persist over the long term. In particular, each application notifies the network of the Quality of Service (QoS) specifications required for that application. Such QoS specifications consist of a lower bound, an upper bound and an operating level for that parameter. The lower bound is the value of the parameter below which the application would like an input signal, the upper bound is the value of the parameter above which the application would like an input signal, and the operating level is the value at which the application would prefer to operate over the long term. The operating level need not be midway between the upper and lower bounds, but merely between these maximum and minimum values. The user application will then receive transmission parameter input signals only when the value of the parameter falls outside of the upper or lower bound. The provision of both upper and lower bounds is necessary to insure that the application can return to the preferred operating level after congestion has abated.

In accordance with another feature of the present invention, an observation period is specified for each transmission parameter. That is, each transmission parameter is monitored at the end of an observation period. If the monitored value of the parameter lies outside of the specified bounds, its value can be sent to the user application. In the alternative, if the instantaneous value of the parameter is unstable, some computed function of the parameter value may be used, such as an average or an exponential average, both to ensure that the value is actually within or outside of the bounds, and as the better value to be passed to the user application. If the user application realizes that the operating levels or bounds on any parameter are no longer suitable for the current network status, new operating points and bound values can be passed to the network, overriding the previous values.

BRIEF DESCRIPTION OF THE DRAWINGS

Acomplete understanding of the present invention may be gained by considering the following detailed description in conjunction with the accompanying drawings, in which:

FIG. 1 shows a general block diagram of a packet communications network in which a user-controlled flow control mechanism in accordance with the present invention might find use;

FIG. 2 shows a more detailed block diagram of typical endnode in the network of FIG. 1 at which point packets may enter the network to be forwarded along the route to a destination for each packet, and in which transmission parameter observation and user application notification of parameter variations in accordance with the present invention might find use;

FIG. 3 shows a flow chart of the processing of user requests for opening a connection, and monitoring and controlling transmission parameters in processor 44 of FIG. 2, all in accordance with the present invention;

FIG. 4 shows a flow chart of the processing of transmission parameter violations in monitor 37 and reporter 43 of FIG. 2 in accordance with the present invention; and

FIG. 5 shows a general flow chart of the process of adapting transmission parameters to changes in the quality of service provided by the network of FIG. 1, such process taking place in a user application such as applications 40, 41 and 42 of FIG. 2 in accordance with the present invention.

To facilitate reader understanding, identical reference numerals are used to designate elements common to the figures.

DETAILED DESCRIPTION

Referring more particularly to FIG. 1, there is shown a general block diagram of a packet transmission system 10 comprising eight network nodes 11 numbered 1 through 8. Each of network nodes 11 is linked to others of the network nodes 11 by one or more communication links A through L. Each such communication link may be either a permanent connection or a selectively enabled (dial-up) connection. Any or all of network nodes 11 may be attached to end nodes, network node 2 being shown as attached to end nodes 1, 2 and 3, network node 7 being shown as attached to end nodes 4, 5 and 6, and network node 8 being shown as attached to end nodes 7, 8 and 9. Network nodes 11 each comprise a data processing system which provides data communications services to all connected nodes, network nodes and end nodes, as well as providing decision points within the node. The network nodes 11 each comprise one or more decision points within the node, at which point incoming data packets are selectively routed on one or more of the outgoing communication links terminated within that node or at another node. Such routing decisions are made in response to information in the header of the data packet. The network node also provides ancillary services such as the calculation of new routes or paths between terminal nodes, the provision of access control to packets entering the network at that node, and the provision of directory services and topology database maintenance at that node. In accordance with the present invention, one or more of network nodes 11 can also comprise a centralized route management system.

Each of end nodes 12 comprises either a source of digital data to be transmitted to another end node, a utilization device for consuming digital data received from another end node, or both. Users of the packet communications network 10 of FIG. 1 may utilize an end node device 12 connected to the local network node 11 for access to the packet network 10. The local network node 11 translates the user's data into packets formatted appropriately for transmission on the packet network of FIG. 1 and generates the header which is used to route the packets through the network 10. In accordance with the present invention, one or more of nodes 11 and 12 of FIG. 1 is equipped to provide user-controlled data flow control for access to the network of FIG. 1.

In order to transmit packets on the network of FIG. 1, it is necessary to calculate a feasible path or route through the network from the source node to the destination node for the transmission of such packets. To avoid overload on any of the links on this route, the route is calculated in accordance with an algorithm that insures that adequate bandwidth is available on each leg of the new connection. One such optimal route calculating systems is disclosed in U.S. Pat. No. 5,233,604 granted Aug. 3, 1993. Once such a route is calculated, a connection request message is launched on the network, following the computed route and updating the bandwidth occupancy of each link along the route to reflect the new connection. Data packets may then be transmitted along the calculated route from the originating node to the destination node (and from the destination node to the originating node) by placing this route in the header of the data packet. In prior art systems, if the network of FIG. 1 became congested, the network would detect this condition and limit the access of traffic to the system. While this procedure protected the system against overload, it was not always the best way to transmit the user's data, particularly multimedia video data requiring real time delivery.

In FIG. 2 there is shown a general block diagram of a network endnode control circuit which might be found in the nodes 12 of FIG. 1. The endnode control circuit of FIG. 2 comprises a high speed packet switching fabric 33 onto which packets arriving at the node are entered. Such packets arrive over transmission links from network nodes of the network, such as links M-O of FIG. 1 via transmission interfaces 34, 35 or 36, or are originated locally via local user interfaces 30, 31 or 32. Switching fabric 33, under the control of route controller 39, connects each of the incoming data packets to the appropriate one of the outgoing transmission link interfaces 34–36 or to the appropriate one of the local user interfaces 30–32, all in accordance with well known packet network operations. Indeed, network management control messages are also launched on, and received from, the packet network in the same fashion as data packets. That is, each network packet, data or control message, transmitted on the network of FIG. 1 can be routed by way of switching fabric 30, as shown in FIG. 2.

Routes or paths through the network of FIG. 1 are calculated to satisfy the Quality of Service (QoS) parameters determined to be necessary to adequately transmit a particular data stream as taught in the afore-mentioned U.S. Pat. No. 5,233,604. These Quality of Service parameters include such things as throughput (bandwidth), latency (path delay) and jitter (latency variations). If, due to changes in traffic loading or outages, the selected path is no longer capable of providing the desired QoS parameters, it is customary to restrict the access to the network in such a way as to reduce the load on the system. Such restricted access was imposed on input data streams regardless of the degradation thereby introduced into the transmitted signals.

In accordance with the present invention, some input signals to a packet communications network can be better accommodated in a network with reduced capability by the user application source of those input signals than by the network management facilities. Video and audio signals, for example, depend on real time delivery of the successive video frames for realistic reproduction of the moving picture. Delayed transmissions enforced by the network can degrade the video signals in such a fashion as to render the signal useless. The user application, on the other hand, can choose to reduce the frame rate of a video signal and thereby produce a useable, albeit degraded, video signal. The present invention provides a mechanism which allows the user application to control the flow control access to a network such as that of FIG. 1 by passing information about the state of the network to the user application, and allowing that user application to use this information to control the rate of data delivery to the network.

In accordance with the present invention, a user request processor 44 is provided in FIG. 2 to receive and process flow control requests from user applications 40–42. Such requests can include requests to access the network, requests to monitor certain Quality of Service parameters, and requests to change a particular Quality of Service parameter in response to changes in the network requiring flow control intervention. In response to a request processed in processor 44, network parameter monitor 37 uses prior art methods to monitor the desired parameter. As will be described hereinafter, this monitoring is particularized for a given network parameter and is averaged over a specified observation interval. Results of such monitoring are reported, using prior art signaling methods, to the user applications 40–42 by network event reporter 43. In response to these network events, user applications 40–42 control the flow of data from their respective applications into the network. The detailed processes which take place in blocks 44, 37, 43 and 40–42 are shown in the flow charts of FIGS. 3 through 5.

The processes of FIGS. 3 through 5 can, of course, be implemented by designing appropriate special purpose circuits. In the preferred embodiment, however, the processes of FIGS. 3–5 are implementing by programming a general purpose computer of the type normally used to control user stations in packet or cell transmission networks. Such programming is obvious to persons skilled in the network node control and operation arts and will not be further described here.

Referring then to FIG. 3, there is shown a flow chart of the processes taking place in the user request processor 44 of FIG. 2. Starting in start box 50, box 51 is entered where the processor waits for the next request from a user application. In box 60, it is detected that a request is received and, in decision box 52, it is determined whether or not the request is to open a new connection. If so, box 53 is entered where the Quality of Service parameters associated with the new connection are saved. These parameters are used to select a route for the new connection capable of satisfying these parameters. Once such a route is determined, the user requesting the new connection can begin transmitting data to the network for transmission along that route. At this time, the application has not specified which QoS parameter violations of which it would like to be notified.

If the new request is not for an open connection, as determined in decision box 52, then decision box 54 is entered to determine whether the request is to monitor a certain QoS parameter. If the request is to monitor a QoS parameter, box 55 is entered where the identification of the QoS parameters are ascertained (from the request) and passed on to network parameter monitor 37 of FIG. 2. At this time, the network is informed which QoS parameter violations are of interest to the application. Box 51 is then re-entered to await the next request from a user application.

If the new request is not to monitor a particular QoS parameter, as determined by decision box 54, decision box 56 is entered to determine if the new request is to change one of the QoS parameters currently being used for a particular connection from a particular user application. If so, box 57 is entered where the new value of that QoS parameter is substituted for the previously stored value from box 53 or from a previous action in box 57. After the QoS parameter is changed in box 57, box 51 is re-entered to await the next request from a user application.

If the new request is not to change the value of a QoS parameter, as determined by decision box 56, box 58 is entered where an error notification is sent to the user application and to the network manager. That is, if the user request is not for a new connection or to monitor a QoS parameter or to change a QoS parameter, then an error has occurred and the user application is so notified. Box 51 is then re-entered to await the next request from a user application.

In FIG. 4 there is shown a flow chart of the processing of QoS parameter violations detected in network parameter monitor 37 of FIG. 2. Before proceeding to a description of FIG. 4, it is first necessary to describe the operation of the flow control system of the present invention. Each user application, at the time of establishing a new connection, notifies the endnode 12 (FIG. 2) of the Quality of Service parameters required to properly transmit the data stream to be launched from that user application. Rather than simply sending the values of each parameter, the user application supplies the network with a triplet of values for each QoS parameter consisting of (1) the preferred operating value of that parameter, (2) a lower bound on the value of the parameter below which the user application wants to be notified so as to exercise a flow control option, and (3) an upper bound on the value of the parameter above which the user application wants to be notified so as to exercise a flow control option. In addition, for each QoS parameter, the application supplies an observation interval that determines, for the respective parameter, the frequency of monitoring that parameter. The user application is thus able to ignore small transitory changes in a parameter value and react only to larger, persistent changes. Thus, a certain amount of "hysteresis" is built into the flow control process, smoothing the application adaptation changes. With this in mind, FIG. 4 can now be described. As previously noted, Quality of Service parameters can include such metrics as bandwidth, latency and jitter. For the purposes of simplicity, FIG. 4 describes the monitoring of only a single QoS parameter. Those skilled in the art can extend FIG. 4 to accommodate the monitoring of any of the other possible parameters. Furthermore, the method of measuring the QoS parameters can be implemented in ways well known in the prior art and will not be specifically disclosed herein. The implementation of these other measurements is well known to anyone of ordinary skill in the art and can be implemented without any undue experimentation.

In FIG. **4**, starting in start box **70**, box **71** is entered where an observation interval timer is started. For simplicity, it is assumed that a separate interval timer is provided for each QoS parameter that is to be monitored. For efficiency, however, a plurality of different QoS parameters could be monitored simultaneously, using a common interval timer. The interval timer is used to sample the QoS parameter periodically, rather than continuously, in order to reduce the measurement overhead. After starting the interval timer in box **71**, decision box **72** is entered to determine whether or not the observation interval is over, i.e., the interval timer has timed out. If not, decision box is re-entered to await the termination of the interval. When the observation timer does time out, the interval is recognized as being over and box **73** is entered to measure or determine the current value of the QoS parameter in question. This particular QoS parameter may be measured over the particular observation interval, such as accumulating jitter on a per data packet basis, or a measurement may be taken at the end of the observation interval, such as measuring latency by computing the round trip delay of a test message. The implementation of these measurement techniques are well known to those of ordinary skill in the art and will not be further described here.

In box **74**, the measured or computed value of the parameter is smoothed by computing an average or exponential average or by using some other user-specified smoothing function. The resulting smoothed value is then used to test against the user-specified lower bound in decision box **78**. If the smoothed value from box **74** is less than the lower bound set for that parameter, box **80** is entered to send an event signal to the user application notifying the user application of the violation of the lower bound and the actual smoothed value of the parameter. The user application can then use this value to determine the changes it will make in its transmission strategy to accommodate the new value of the QoS parameter. This process will be taken up in connection with FIG. 5.

If the smoothed value of the parameter is not below the minimum bound, as determined by decision box **78**, decision box **79** is entered to determine if the smoothed value of the parameter is greater than the upper bound set for that parameter. If so, box **80** is entered to send an event signal to the user application notifying the user application of the violation of the upper bound and the actual measured value of the parameter. The user application uses this value to determine the changes it will make in the transmission strategy to accommodate the new parameter value. Box **71** is then reentered to start a new observation interval. If the measured value does not fall outside of the specified range, as determined by decision boxes **78** and **79**, box **71** is re-entered to start the next measurement interval.

In FIG. **5** there is shown a flow chart of the processing of Quality of Service parameters by a user application, such as one of applications **40**–**42** of FIG. 2. Starting in start box **90**, box **91** is entered the determine the desired Quality of Service parameters, and their respective allowable range of values, for a data stream to be transmitted over a desired new network connection. In box **92**, a new network connection is requested (see FIG. 3) and, in box **93**, the desired Quality of Service parameters are requested for the new connection. The network of FIG. 1 utilizes the specified Quality of Service parameters to select a route through the network of FIG. 1 which satisfies all of the specified parameters, all as taught in the above-mentioned U.S. Pat. No. 5,233,604. Next, box **93** is entered where the user application notifies the network which QoS parameters to monitor. Box **94** is then entered to initialize the transfer of data, for example, video or audio frames. Decision box **95** is then entered to determine if there is any more data signals to be transmitted. If not, the transmission is over and stop box **96** is entered to terminate the transmission process and the connection.

If more data is available for transmission, as determined by decision box **95**, decision box **97** is entered to determine whether or not a QoS parameter violation event signal, transmitted in box **80** of FIG. **4**, has been received. If not, box **98** is entered to transmit one data frame through the network of FIG. **1**, along the selected route. Decision box **95** is then re-entered to determine if the transmission of any more data frames is required. If a QoS parameter violation event signal has been received, as determined decision box **97**, then decision box **99** is entered to determine whether or not the transmission parameters of the user application should be changed in response to the parameter violation. If a change is necessary, box **100** is entered where the user application determines the best action to take in response to the parameter violation, depending on the type of data signal being transmitted, e.g. changing the coding method to reduce bandwidth utilization or packing more signal samples into the same packet to reduce the effects of jitter. The QoS parameters required for change in transmission strategy are computed in box **100** and the resulting new transmission parameters are computed in box **101**. Box **102** is then entered to request the necessary changes in the QoS parameters as shown in FIG. 3. Box **103** is then entered to make the actual changes in the transmission strategy which are necessary to accommodate the violation of the previous parameters. When the transmission adaptations have been effected in box **103**, box **98** is re-entered to transmit the next data frame over the connection, using the new transmission strategy. Decision box **95** is then re-entered to continue transmitting data using the new strategy.

If no transmission parameter changes are necessary, as determined by decision box **99**, but a violation event signal has been received, as determined by decision box **97**, then box **103** is entered to make the necessary adaptation to the violation, but using all of the previously established QoS parameters. Transmission then continues, using the new adaptive strategy. It can be seen that the process of FIG. 5 permits the user application to adapt the flow of information into the network to maximize the use of the available network path parameters. Since the user application is in a better position to optimize the transmission of the data stream originating at that user application than is the network manager, superior flow control results from giving the user application control over the data flow into the network. This is in distinct contrast to prior art, network-controlled data flow mechanisms applied uniformly for all data streams regardless of the special requirements of the particular data stream.

What is claimed is:

1. A packet transmission network comprising

a plurality of transmission nodes interconnected by transmission links,

a plurality of user applications for transmitting data streams on said network, said data streams having at least two different modes of transmission requiring different transmission parameters,

means for selecting a data path through said network between two of said user applications to satisfy the transmission parameters of one of said two different modes of transmission,

means for detecting changes in the transmission parameters available on said selected data path,

means for notifying said user applications of said changes in the transmission parameters, and

means, responsive to said means for notifying, for changing to the other of said two different modes of transmission at said user application.

2. The packet transmission network according to claim 1 further comprising

means at each of said user applications for specifying a range of values of said transmission parameters within which said one mode of transmission remains unchanged.

3. The packet transmission network according to claim 1 further comprising

means, in each said user application, for requesting changes in said transmission parameters for a particular connection.

4. The packet transmission network according to claim 1 further comprising

means for storing the quality of service transmission parameters requested by each of said user applications for each requested connection to said user application.

5. The packet transmission network according to claim 1 further comprising

means for computing a smoothing function of the transmission parameter values on each connection through said network for a predetermined observation interval.

6. The packet transmission network according to claim 1 further comprising

means for transmitting an event signal to said user applications when said transmission parameters fall outside of said specified range of values.

7. A method for operating a packet transmission network comprising the steps of

interconnecting a plurality of transmission nodes by transmission links,

transmitting a plurality of data streams from user applications connected to said network, said data streams having at least two different modes of transmission requiring different transmission parameters,

selecting a data path through said network between two of said user applications to satisfy the transmission parameters of one of said two different modes of transmission,

detecting changes in the transmission parameters available on said selected data path,

notifying said user applications of said changes in the transmission parameters, and

in response to said step of notifying, changing to the other of said two different modes of transmission at said user application.

8. The method according to claim 7 further comprising the step of

at each of said user applications, specifying a range of values of said transmission parameters within which said one mode of transmission remains unchanged.

9. The method according to claim 7 further comprising the step of

in each said user application, requesting changes in said transmission parameters for a particular connection.

10. The method according to claim 7 further comprising the step of

storing the quality of service transmission parameters requested by each of said user applications for each requested connection to said user application.

11. The method according to claim 7 further comprising the step of

smoothing the values of transmission parameters for each connection through said network.

12. The method according to claim 7 further comprising the step of

transmitting an event signal to said user applications when said transmission parameters fall outside of said specified range of values.

13. A data flow control system for packet communications systems connected to a plurality of user applications comprising

means in said packet communications system for measuring the transmission parameters of at least one route from one of said user applications to another of said user applications,

means in each of said user applications, responsive to said means for measuring, for changing the flow rate of data transmitted over said at least one route,

means in said user applications for specifying a range of permissible values for each of said transmission parameters, and

means in each of said user applications for requesting changes in the requested transmission parameters for said at least one route.

14. The data flow control system according to claim 13 further comprising

means for storing the quality of service parameters for said at least one route in said packet communications system.

15. The data flow control system according to claim 13 further comprising

means in said packet communications system for smoothing the values of said transmission parameters.

16. A method for controlling data flow into a packet communication system connected to a plurality of user applications comprising the steps of

in said packet communications system, measuring the transmission parameters of at least one route from one of said user applications to another of said user applications,

in each of said user applications, in response to said means for measuring, changing the flow rate of data transmitted over said at least one route,

in each of said user applications, specifying a range of permissible values for each of said transmission parameters, and in each of said user applications, requesting changes in the requested transmission parameters for said at least one route.

17. The method according to claim 16 further comprising the step of

storing the quality of service parameters for said at least one route in said packet communications system.

18. The method according to claim 16 further comprising

in said packet communications system, smoothing the values of said transmission parameters over a predetermined measuring interval.

19. A packet transmission network comprising

a plurality of transmission nodes interconnected by transmission links,

a plurality of user applications for transmitting data streams on said network, said data streams having at least two different modes of transmission requiring different transmission parameters,

means for selecting a data path through said network between two of said user applications to satisfy the transmission parameters of one of said two different modes of transmission,

means for detecting changes in the transmission parameters available on said selected data path,

means for notifying said user applications of said changes in the transmission parameters,

means, responsive to said means for notifying, for changing to the other of said two different modes of transmission at said user application, means in each said user application for requesting a new connection satisfying a specified range of transmission parameters,

means in each said user application for requesting said network to monitor specified network parameters for each said connection, and

means in each said user application for requesting changes in said specified transmission parameters.

20. A method for operating a packet transmission network comprising the steps of

interconnecting a plurality of transmission nodes by transmission links,

transmitting a plurality of data streams from user applications connected to said network, said data streams having at least two different modes of transmission requiring different transmission parameters,

selecting a data path through said network between two of said user applications to satisfy the transmission parameters of one of said two different modes of transmission,

detecting changes in the transmission parameters available on said selected data path,

notifying said user applications of said changes in the transmission parameters,

in response to said step of notifying, changing to the other of said two different modes of transmission at said user application,

in each said user application, requesting a new connection satisfying a specified range of transmission parameters,

in each said user application, requesting said network to monitor specified network parameters for each said connection, and

in each said user application, requesting changes in said specified transmission parameters.

21. A data flow control system for packet communications systems connected to a plurality of user applications comprising

means in said packet communications system for measuring the transmission parameters of at least one route from one of said user applications to another of said user applications, and

in each of said user applications,

means responsive to said means for measuring, for changing the flow rate of data transmitted over said at least one route,

means for requesting a new route through said packet communications system satisfying specified ranges of values of transmission parameters,

means for requesting the measurement of specific transmission parameters over a specified route, and

means for requesting a change in a previously specified transmission parameter.

22. A method for controlling data flow into a packet communications system connected to a plurality of user applications comprising the steps of

in said packet communications system, measuring the transmission parameters of at least one route from one of said user applications to another of said user applications,

in each of said user applications, in response to said means for measuring,

changing the flow rate of data transmitted over said at least one route,

requesting a new route through said packet communications system satisfying specified ranges of values of transmission parameters,

requesting the measurement of specific transmission parameters over a specified route, and

requesting a change in a previously specified transmission parameter.

23. In a user application for use with a packet transmission network having a plurality of transmission nodes interconnected by transmission links, a network route selector for selecting a data path through said network between said user application and a second user application, said user application transmitting a data stream on said network in one of at least two different modes of transmission requiring different transmission parameters, a network parameter monitor for detecting changes in transmission parameters available on a selected data path and a network event reporter for providing notification of detected changes in transmission parameters, a flow control system comprising:

means for receiving a notification of a change in transmission parameters on the selected data path from the network event reporter; and

means responsive to the receipt of said notification to change a different one of the different modes of transmission.

24. A flow control system as defined in claim 23 further including means for requesting changes in transmission parameters for the selected data path.

25. A flow control system as defined in claim 23 further including:

means for requesting a new connection to a second user application satisfying a specified range of transmission parameters,

means for specifying network parameters to be monitored by the network parameter monitor, and

means for requesting changes in the network in transmission parameters established for a connection.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address COMMISSIONER FOR PATENTS
PO 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

7590 07/10/2003

Dov Rosenfeld
Suite 2
5507 College Avenue
Oakland, CA 94618

EXAMINER	
VU, THONG H	
ART UNIT	PAPER NUMBER
2142	5

DATE MAILED: 07/10/2003

Please find below and/or attached an Office communication concerning this application or proceeding.

PTO-90C (Rev. 07-01)

PRG

Office Action Summary	**Application No.** 09/608,126	**Applicant(s)** DIETZ ET AL.
	Examiner Thong H Vu	**Art Unit** 2142

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the penod for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133)
- Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any eamed patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on 30 June 2000 .

2a)□ This action is **FINAL**. 2b)☒ This action is non-final.

3)□ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) 1-21 is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)□ Claim(s) _____ is/are allowed.

6)☒ Claim(s) 1-21 is/are rejected.

7)□ Claim(s) _____ is/are objected to.

8)□ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)□ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on 30 June 2000 is/are: a)☒ accepted or b)□ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

11)□ The proposed drawing correction filed on _____ is: a)□ approved b)□ disapproved by the Examiner.

If approved, corrected drawings are required in reply to this Office action.

12)□ The oath or declaration is objected to by the Examiner.

Priority under 35 U.S.C. §§ 119 and 120

13)□ 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:

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)).

* See the attached detailed Office action for a list of the certified copies not received.

14)□ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application).

a) □ The translation of the foreign language provisional application has been received.

15)□ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) □ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☒ Information Disclosure Statement(s) (PTO-1449) Paper No(s) 4 .

4) □ Interview Summary (PTO-413) Paper No(s). _____ .

5) □ Notice of Informal Patent Application (PTO-152)

6) □ Other: .

1. Claims 1-21 are pending .

2. The numbering of claims is not in accordance with 37 CFR 1.126 which requires the original numbering of the claims to be preserved throughout the prosecution. When claims are canceled, the remaining claims must not be renumbered. When new claims are presented, they must be numbered consecutively beginning with the number next following the highest numbered claims previously presented (whether entered or not).

Misnumbered of paragraphs in claims 1 and 10 have been renumbered (a), (b) (c) (d) for claim 1 and (c), (d) for claim 10.

3. Claim 1 is objected to because of the following informalities: a flow-entry database comprises **none** or more flow-entries. Examine consider as **one** or more flow entries. Appropriate correction is required.

Claim Rejections - 35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

4. Claims 1-21 are rejected under 35 U.S.C. § 102(e) as being anticipated by Anderson et al [Anderson 5,850,388]

5. As per claim 1, Anderson discloses a method of analyzing a flow of packets (or frames) passing through a connection point (protocol analyzer) on a computer network [col 4 line 49-col 6 line 19], the method comprising:

(a) receiving a packet from a packet acquisition device [protocol analyzer, col 8 line 26-col 9 line 13];

(b) looking up a flow-entry database [database, col 5 lines 24-46, col 9 lines 30-40, col 23 lines 35-45, col 24 lines 6-20,57-col 25 line 50; lookup table, col 18 lines 29-37] comprising one or more flow-entries for previously encountered conversational flows, the looking up to determine if the received packet is of an existing flow [previous session, col 24 lines 6-13; prior entries, col 28 lines 26-43];

(c) if the packet is of an existing flow, updating the flow-entry of the existing flow including storing one or more statistical measures kept in the flow-entry [col 17 lines 15-23, col 25 lines 22-47, col 27 lines 24-34, col 28 lines 49-67]; and

(d) if the packet is of a new flow, storing a new flow-entry for the new flow in the flow-entry database [updat new information, col 27 lines 10-53], including storing one or more statistical measures kept in the flow-entry [statistics, col 27 lines 10-34], wherein every packet passing though the connection point is received by the packet acquisition device [protocol analyzer col 8 line 26-col 9 line 13].

6. Claim 17 contains the similar limitations set forth of method claim 1. Therefore, claim 17 is rejected for the similar rationale set forth in claim 1

7. As per claim 2, Anderson discloses extracting identifying portions from the packet, wherein the looking up uses a function of the identifying portions [information is extracted from a frame, col 9 line 42-col 10 line 18].

8. As per claim 3, Anderson discloses the steps are carried out in real time on each packet passing through the connection point [col 4 line 58-col 5 line 46].

9. As per claim 4, Anderson discloses the one or more statistical measure includes selected from the set of consisting of the total packet count for the flow, the time and a differential time from the last entered time to the present time [col 28 lines 58-67].

10. As per claim 5, Anderson discloses including one or more metrics (parameters) related to the flow of a flow entry from one or more of the statistical measure in the flow entry [col 10 lines 20-40, col 19 lines 30-45,col 22 lines 16-65].

11. As per claim 6, Anderson discloses the metrics include one or more quality of service (QOS) metrics (id, tiem, length col 22 lines 16-23].

12. As per claim 7, Anderson discloses 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. As per claim 8, Anderson discloses calculating one or more quality of service (QOS) metrics from the base metrics [col 14 lines 39-60, col15 lines 32-46,col 17 lines 45-57].

14. As per claim 9, Anderson discloses 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 [col 28 lines 58-67].

15. As per claim 10, Anderson discloses

(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 [between the last update and the present update, col 26 lines 6-40];

(d) includes if the packet is of a new flow, performing any state operations required for the initial state of the new flow [new data and user initial slect how often information on station statistics was to update, col 26 lines 6-15].

16. As per claim 11, Anderson discloses 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 [col 30 line 58-col 31 line 10].

17. As per claim 12, Anderson discloses 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 [col 30 line 58-col 31 line 10].

18. As per claim 13, Anderson discloses reporting is part of the state operations for the state of the flow [col 30 line 58-col 31 line 10].

19. As per claim 14, Anderson discloses updating the flow-entry, including storing identifying information for future packets to be identified with the flow-entry [col 16 lines 47-54, col19 lines 17-24, col 22 line 66-col 23 line 6] .

20. As per claim 15, Anderson discloses receiving further packets, wherein the state processing of each received packet of a flow furthers the identifying of the application program of the flow as inherent of new data received [col 28 lines 58-67].

21. As per claim 16, Anderson discloses 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 as inherent feature of parameters [col 22 lines 16-65].

22. As per claim 20, Anderson discloses 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 [software performes statistical calculations ,col 7 lines 33-53].

23. As per claim 21, Anderson discloses the statistical processor determines and reports the one or more metrics from time to time [col 30 line 58-col 31 line 10].

24. Any inquiry concerning this communication or earlier communications from the examiner should be directed to examiner Thong Vu, whose telephone number is (703)-305-4643.
The examiner can normally be reached on Monday-Thursday from 8:00AM- 4:30PM.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Mark Powell, can be reached at (703) 305-9703.

Any inquiry of a general nature or relating to the status of this application should be directed to the Group receptionist whose telephone number is (703) 305-9700.

Any response to this action should be mailed to: Commissioner of Patent and Trademarks, Washington, D.C. 20231 or faxed to :

After Final	(703) 746-7238
Official:	(703) 746-7239
Non-Official	(703) 746-7240

Hand-delivered responses should be brought to Crystal Park 11,2121 Crystal Drive, Arlington. VA., Sixth Floor (Receptionist).

Thong Vu
Patent Examiner
Art Unit 2142

Notice of References Cited	Application/Control No. 09/608,126	Applicant(s)/Patent Under Reexamination DIETZ ET AL.	
	Examiner Thong H Vu	Art Unit 2142	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-US005850388A	12-1998	ANderson et al	370/252
	B	US-US006097699A	08-2000	Chen et al	370/231
	C	US-US006269330B1	07-2001	Cidon et al	704/43
	D	US-US006453345B2	09-2002	Trcka et al	709/224
	E	US-US006381306B1	04-2002	Lawson et al	379/32
	F	US-US006282570B1	08-2001	Leung et al	709/224
	G	US-US005761429A	06-1998	Thompson	709/224
	H	US-5799154	08-1998	Kuriyan	709/223
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	Advanced Methods for Storage and Retrieval in Image ; http://www.cs.tulane.edu/www/Prototype/proposal.html; 1998
	V	Measurement and analysis of the digital DECT propagation channel; IEEE 1998
	W	
	X	

*A copy of this reference is not being furnished with this Office action (See MPEP § 707 05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

US005850388A

United States Patent [19]

Anderson et al.

[11] **Patent Number: 5,850,388**

[45] **Date of Patent: Dec. 15, 1998**

[54] **PROTOCOL ANALYZER FOR MONITORING DIGITAL TRANSMISSION NETWORKS**

[75] Inventors: **Craig D. Anderson**, Durham; **Mark B. Anderson**, Chapel Hill; **Eugene N. Cookmeyer**, Apex; **Ralph A. Daniels**, Clayton; **Lee E. Wheat**, Durham; **Roger A. Lingle**, Raleigh, all of N.C.

[73] Assignee: **Wandel & Goltermann Technologies, Inc.**

[21] Appl. No.: **742,093**

[22] Filed: **Oct. 31, 1996**

Related U.S. Application Data

[60] Provisional application No. 60/023,459, Aug. 2, 1996.

[51] **Int. Cl.**^{6} **H04L 12/26**

[52] **U.S. Cl.** **370/252**; 371/20.1; 395/183.15

[58] **Field of Search** 370/241, 252, 370/253; 371/20.1; 395/183.15, 200.94

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,437,184	3/1984	Cork et al.	371/19
4,550,407	10/1985	Couasnon et al.	371/29
4,672,611	6/1987	Fukuhara et al.	371/59
4,680,755	7/1987	Reames	370/85
4,775,973	10/1988	Tomberlin et al.	370/60
4,792,753	12/1988	Iwai	324/73
4,887,260	12/1989	Carden et al.	370/60
4,916,694	4/1990	Roth	370/94
5,040,111	8/1991	Al-Salameh et al.	364/200
5,090,014	2/1992	Polich et al.	371/15.1
5,097,469	3/1992	Douglas	371/20.1
5,187,708	2/1993	Nakatani et al.	370/85.1
5,197,127	3/1993	Waclawski et al.	395/200
5,260,970	11/1993	Henry et al.	375/10
5,276,529	1/1994	Williams	358/406
5,276,802	1/1994	Yamaguchi et al.	395/164
5,278,836	1/1994	Iimura et al.	370/112
5,282,194	1/1994	Harley, Jr. et al.	370/17
5,287,506	2/1994	Whiteside	395/650
5,293,384	3/1994	Keeley et al.	371/16.3
5,303,344	4/1994	Yokoyama et al.	395/200
5,309,507	5/1994	Hosaka et al.	379/96
5,317,725	5/1994	Smith et al.	395/575
5,317,742	5/1994	Bapat	395/700
5,325,528	6/1994	Klein	395/650
5,333,302	7/1994	Hensley et al.	395/575
5,345,396	9/1994	Yamaguchi	395/500
5,347,524	9/1994	I'Anson	371/29.1
5,373,346	12/1994	Hocker	364/550
5,375,126	12/1994	Wallace	371/20.1
5,375,159	12/1994	Williams	379/23
5,377,196	12/1994	Godlew et al.	371/20.1
5,418,972	5/1995	Takeuchi et al.	395/800

(List continued on next page.)

OTHER PUBLICATIONS

Uyless Black, "OSI: A Model for Computer Communications Standards", Prentice-Hall, Inc., pp. 8–11 and 54–56, 1991.

Weaver, Alfred C. and McNabb, James F., "A Real-Time Monitor for Token Ring Networks," MILCOM '89: Bridging the Gap, pp. 794–798, 1989.

Brochure, Network General Corporation, Products and Services, dated May, 1995, face, back page, and pp. 1–10.

Primary Examiner—Melvin Marcelo

Attorney, Agent, or Firm—Moore & Van Allen, PLLC; William G. Dosse

[57] **ABSTRACT**

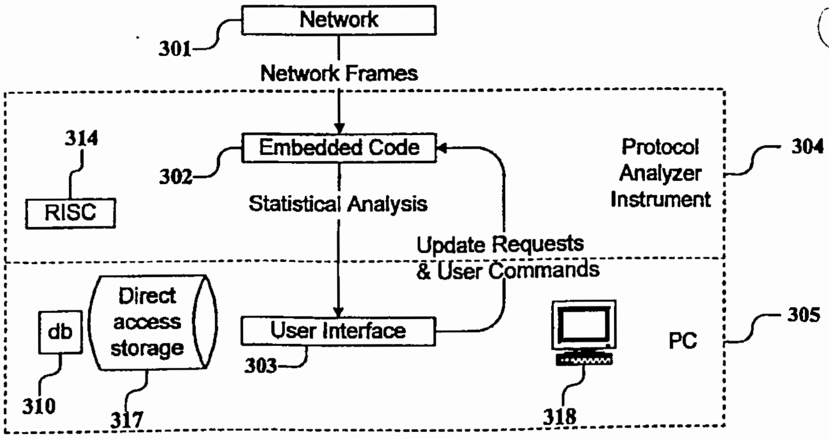
A new and improved protocol analyzer for monitoring digital transmission networks is disclosed. The protocol analyzer of the present invention is capable of displaying station level statistics, network statistics, real-time event information, and protocol distribution. The protocol analyzer of the present invention is additionally capable of creating baseline network performance information and displaying the baseline information simultaneously with real-time performance information, pre-programming monitoring sessions, and generating presentation-quality reports in conjunction with analyzing digital transmission networks, all in real time.

1 Claim, 18 Drawing Sheets

(22) ma[illegible] flow of [illegible]

(9) Frame A[illegible]

[illegible]



U.S. PATENT DOCUMENTS

5,434,845	7/1995	Miller	370/13
5,440,719	8/1995	Hanes et al.	395/500
5,442,737	8/1995	Smith	395/135
5,442,741	8/1995	Hughes et al.	395/142
5,444,706	8/1995	Osaki	370/94.1
5,446,874	8/1995	Waclawski et al.	395/575
5,457,729	10/1995	Hamann et al.	379/2
5,469,463	11/1995	Polich et al.	395/182.18
5,473,551	12/1995	Sato et al.	364/496
5,475,732	12/1995	Pester, III	379/34
5,477,531	12/1995	McKee et al.	370/17
5,481,548	1/1996	Wallace	371/20.1
5,490,199	2/1996	Fuller et al.	379/1
5,504,736	4/1996	Cubbison, Jr.	370/13
5,701,400	12/1997	Amado	395/76

OTHER PUBLICATIONS

Distributed Sniffer System, "Seven–Layer Analysis on all Segments Quickly Pinpoints Problems And Recommends Solutions", Network General Corporation, dated May, 1996, 6 pages.

Sniffer Network Analyzer, "Seven–Layer Analysis on all Segments Quickly Pinpoints Problems And Recommends Solutions", Network General Corporation, dated Jul. 1996, 6 pages.

Product Brochure, DominoLAN™ Internetwork Analyzer, DA–320, Wandel & Goltermann, 6 pages.

User Guide, Link View™ 1000 Network Analyzer, Tinwald Networking Technologies Inc., dated Jan. 1, 1996.

FIG. 1

LAYER 7 - APPLICATION LAYER
LAYER 6 - PRESENTATION LAYER
LAYER 5 - SESSION LAYER
LAYER 4 - TRANSPORT LAYER
LAYER 3 - NETWORK LAYER
LAYER 2 - DATA LINK LAYER
LAYER 1 - PHYSICAL LAYER

FIG. 2

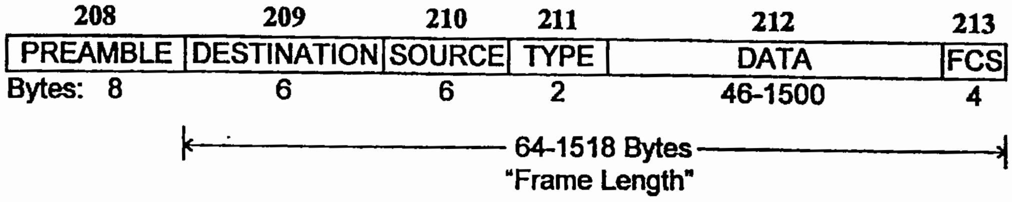


FIG. 3

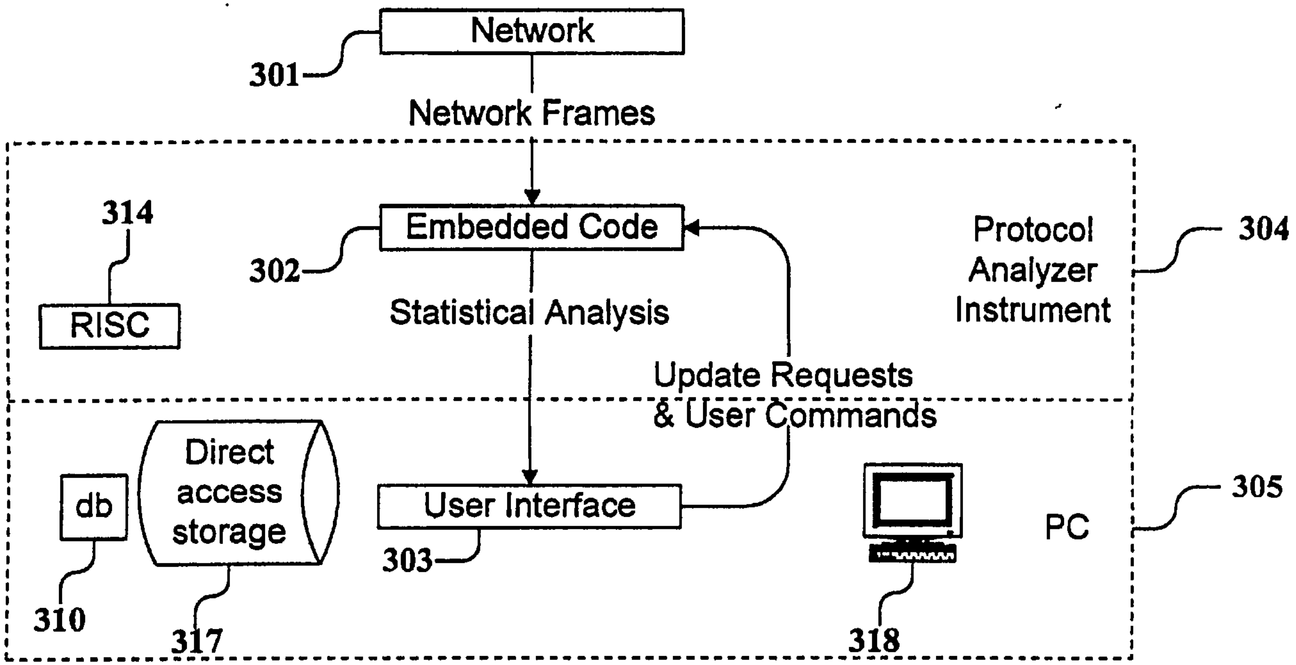

Network
301
Network Frames
314
Embedded Code
302
Protocol
Analyzer
Instrument
304
RISC
Statistical Analysis
Update Requests
& User Commands
Direct
access
storage
db
User Interface
303
PC
305
310
317
318

FIG. 4

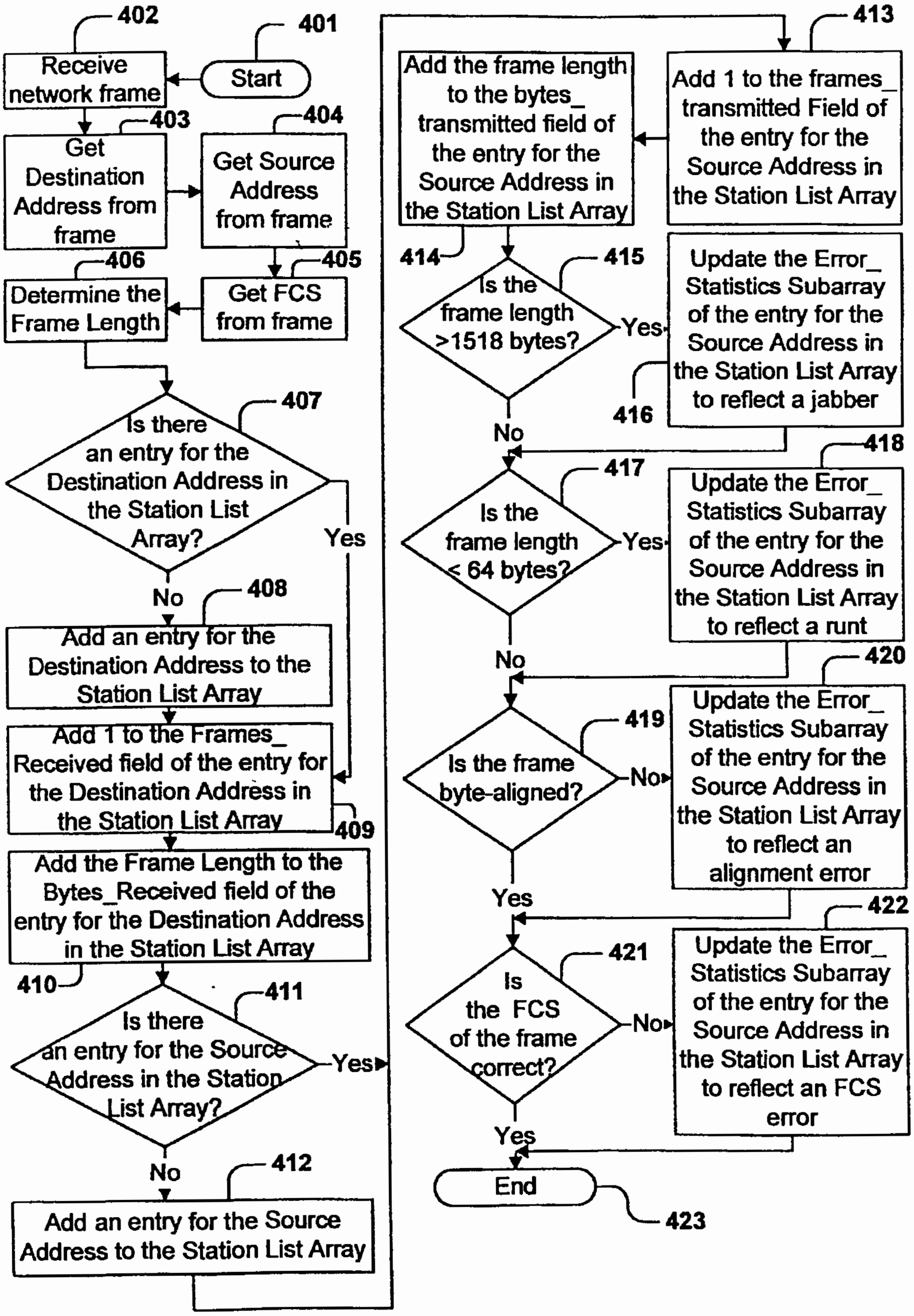

401
Start
402
Receive network frame
403
Get Destination Address from frame
404
Get Source Address from frame
405
Get FCS from frame
406
Determine the Frame Length
407
Is there an entry for the Destination Address in the Station List Array?
Yes
No
408
Add an entry for the Destination Address to the Station List Array
Add 1 to the Frames_ Received field of the entry for the Destination Address in the Station List Array
409
Add the Frame Length to the Bytes_Received field of the entry for the Destination Address in the Station List Array
410
411
Is there an entry for the Source Address in the Station List Array?
Yes
No
412
Add an entry for the Source Address to the Station List Array
413
Add 1 to the frames_ transmitted Field of the entry for the Source Address in the Station List Array
Add the frame length to the bytes_ transmitted field of the entry for the Source Address in the Station List Array
414
415
Is the frame length >1518 bytes?
Yes
No
416
Update the Error_ Statistics Subarray of the entry for the Source Address in the Station List Array to reflect a jabber
417
Is the frame length < 64 bytes?
Yes
No
418
Update the Error_ Statistics Subarray of the entry for the Source Address in the Station List Array to reflect a runt
419
Is the frame byte-aligned?
No
Yes
420
Update the Error_ Statistics Subarray of the entry for the Source Address in the Station List Array to reflect an alignment error
421
Is the FCS of the frame correct?
No
Yes
422
Update the Error_ Statistics Subarray of the entry for the Source Address in the Station List Array to reflect an FCS error
End
423

FIG. 5

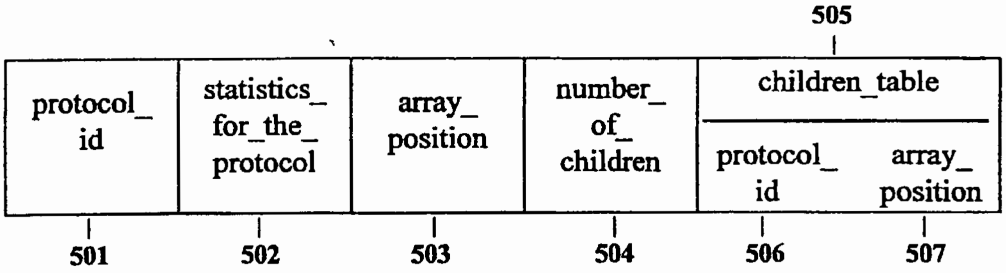

505
protocol_ id
statistics_ for_the_ protocol
array_ position
number_ of_ children
children_table
protocol_ id
array_ position
501
502
503
504
506
507

FIG. 6

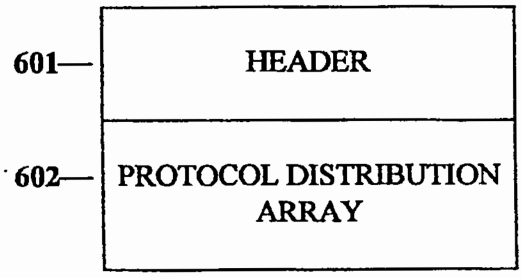

601
HEADER
602
PROTOCOL DISTRIBUTION ARRAY

FIG. 7

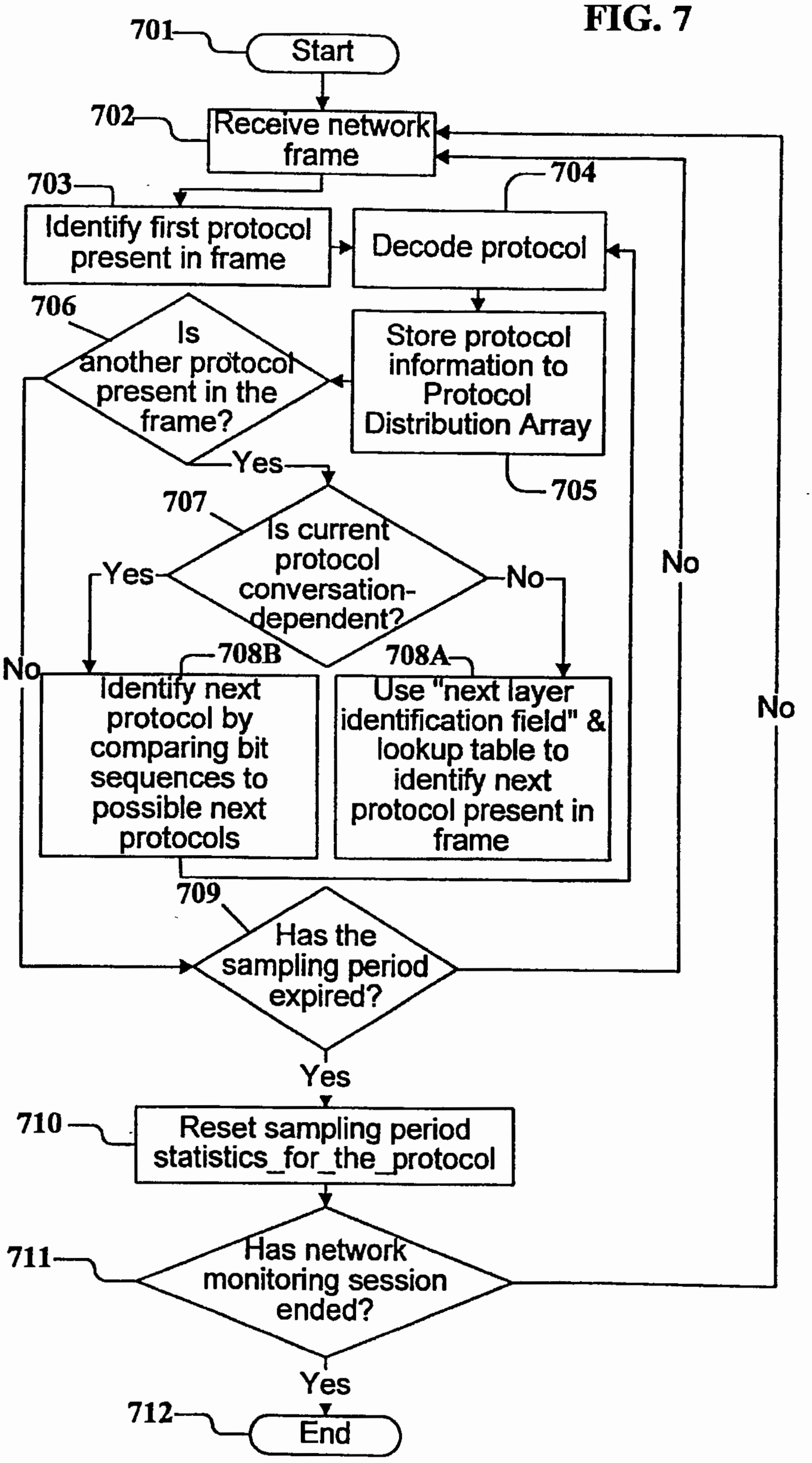

701
Start
702
Receive network frame
703
Identify first protocol present in frame
704
Decode protocol
705
Store protocol information to Protocol Distribution Array
706
Is another protocol present in the frame?
Yes
No
707
Is current protocol conversation-dependent?
Yes
No
708B
Identify next protocol by comparing bit sequences to possible next protocols
708A
Use "next layer identification field" & lookup table to identify next protocol present in frame
709
Has the sampling period expired?
No
Yes
710
Reset sampling period statistics_for_the_protocol
711
Has network monitoring session ended?
No
Yes
712
End

FIG. 8

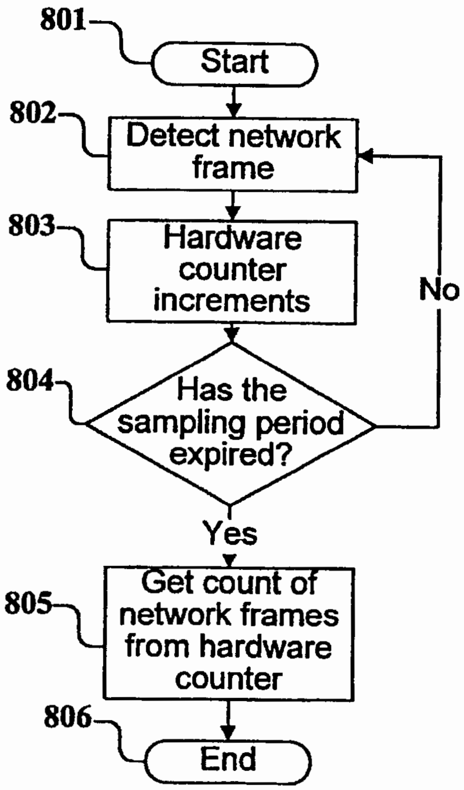

801
Start
802
Detect network frame
803
Hardware counter increments
804
Has the sampling period expired?
No
Yes
805
Get count of network frames from hardware counter
806
End

FIG. 9

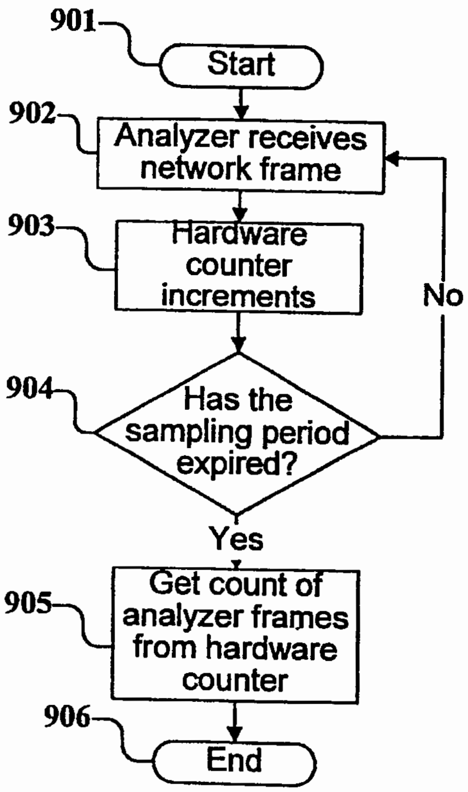

901
Start
902
Analyzer receives network frame
903
Hardware counter increments
904
Has the sampling period expired?
No
Yes
905
Get count of analyzer frames from hardware counter
906
End

FIG. 10

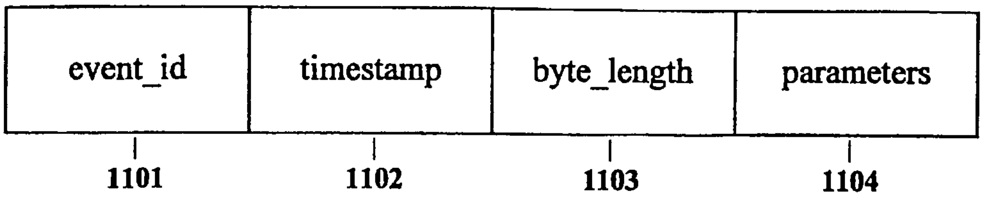


FIG. 11

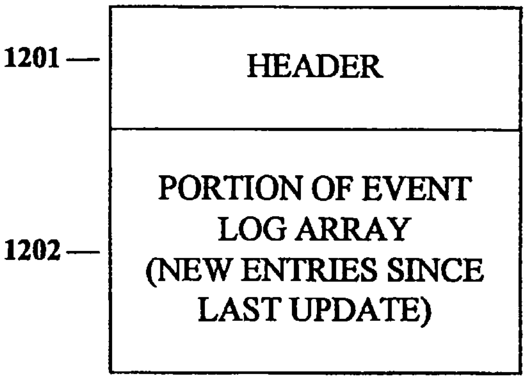


FIG. 12

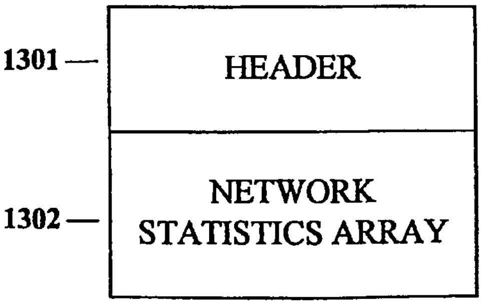


FIG. 13

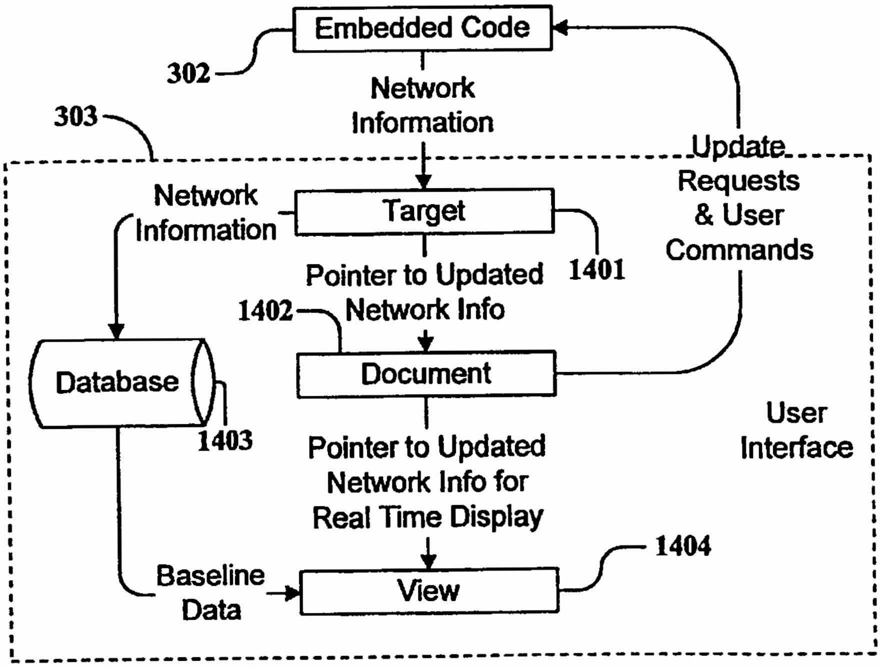

Embedded Code
302
Network Information
303
Update Requests & User Commands
Network Information
Target
1401
Pointer to Updated Network Info
1402
Database
1403
Document
User Interface
Pointer to Updated Network Info for Real Time Display
1404
Baseline Data
View

FIG. 14

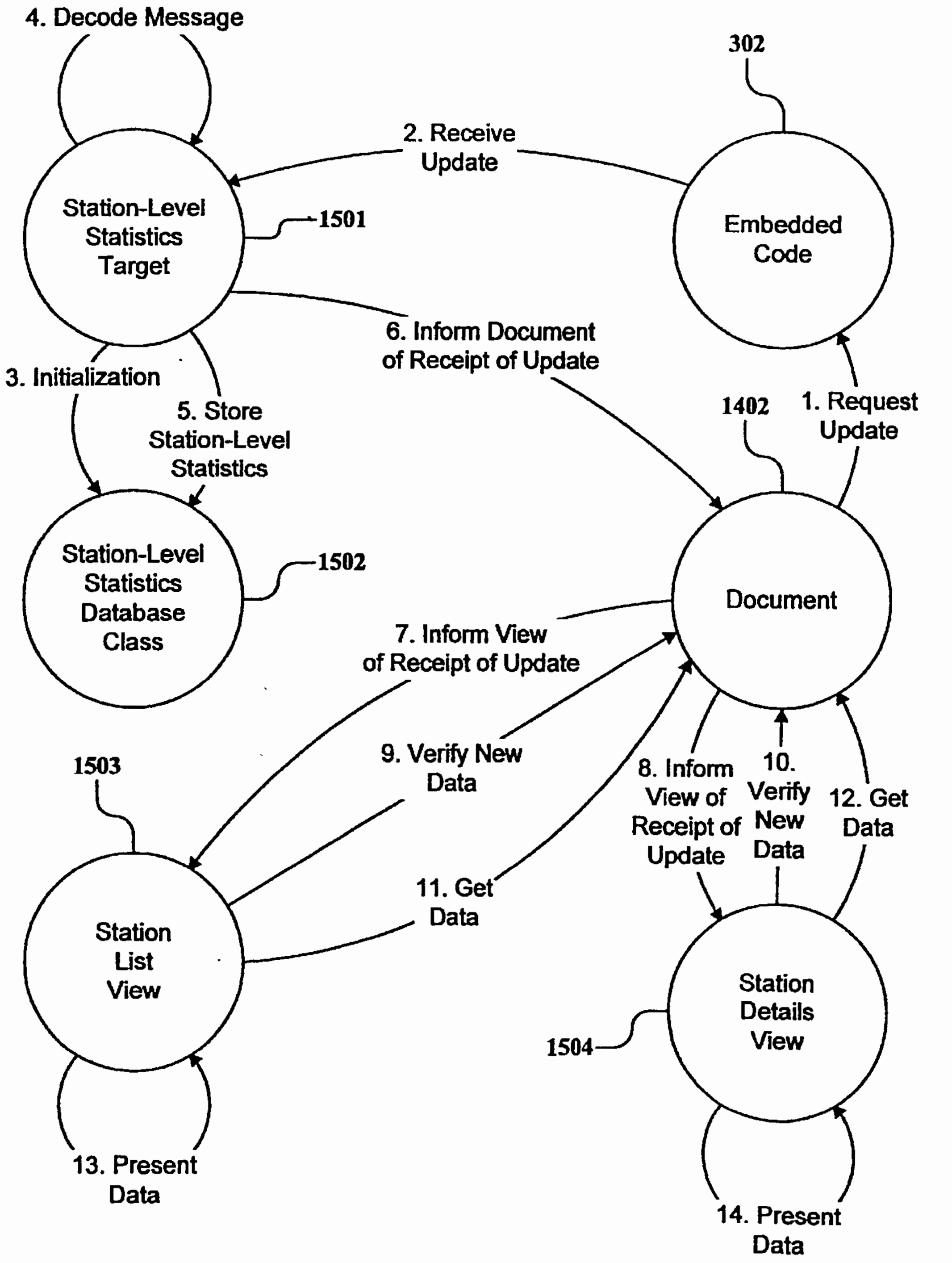

4. Decode Message
302
2. Receive Update
Station-Level Statistics Target
1501
Embedded Code
6. Inform Document of Receipt of Update
3. Initialization
5. Store Station-Level Statistics
1402
1. Request Update
Station-Level Statistics Database Class
1502
Document
7. Inform View of Receipt of Update
9. Verify New Data
1503
8. Inform View of Receipt of Update
10. Verify New Data
12. Get Data
11. Get Data
Station List View
Station Details View
1504
13. Present Data
14. Present Data

FIG. 15

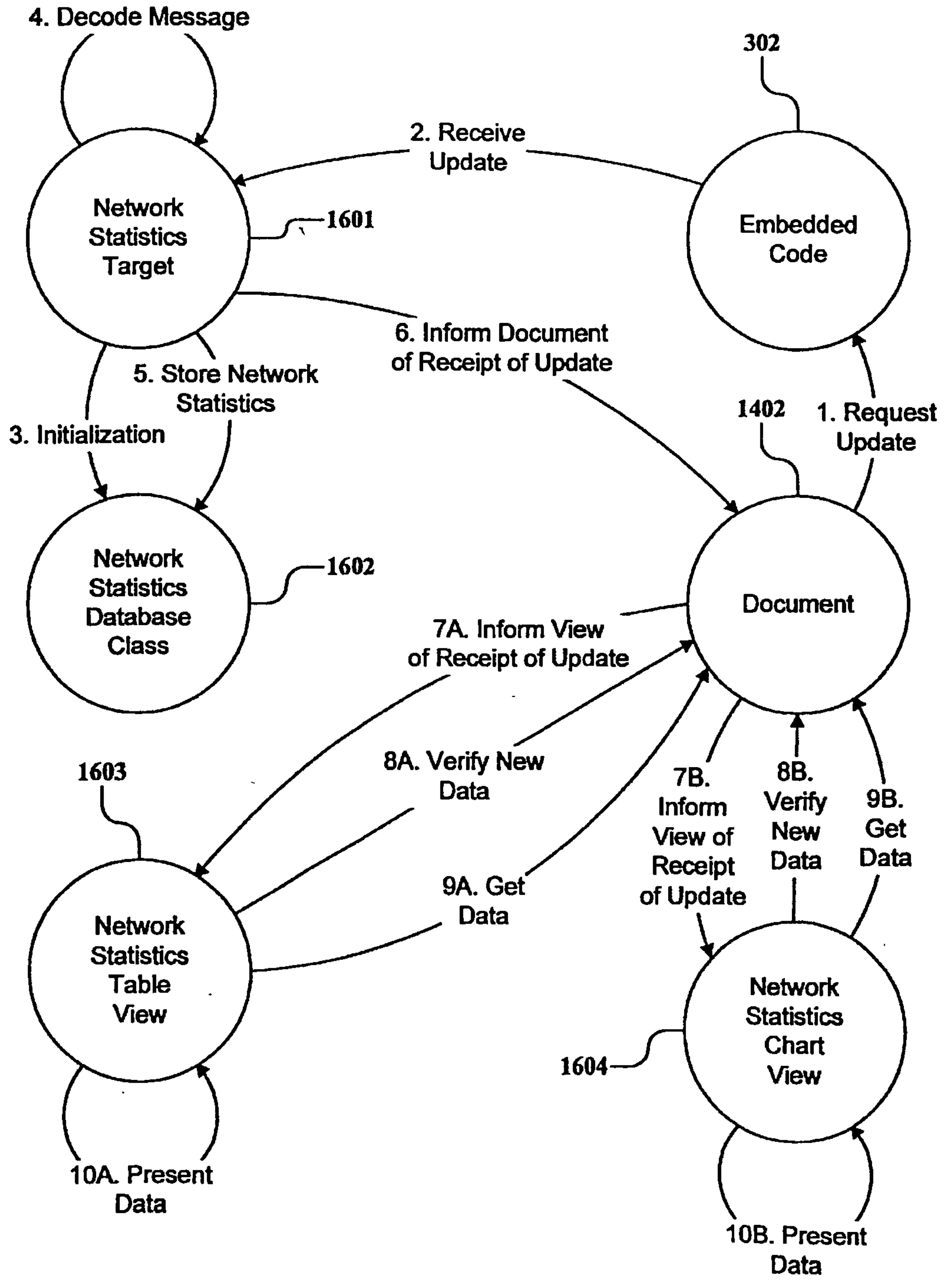

4. Decode Message
302
2. Receive Update
Network Statistics Target
1601
Embedded Code
6. Inform Document of Receipt of Update
5. Store Network Statistics
3. Initialization
1402
1. Request Update
Network Statistics Database Class
1602
Document
7A. Inform View of Receipt of Update
1603
8A. Verify New Data
7B. Inform View of Receipt of Update
8B. Verify New Data
9B. Get Data
9A. Get Data
Network Statistics Table View
Network Statistics Chart View
1604
10A. Present Data
10B. Present Data

FIG. 16

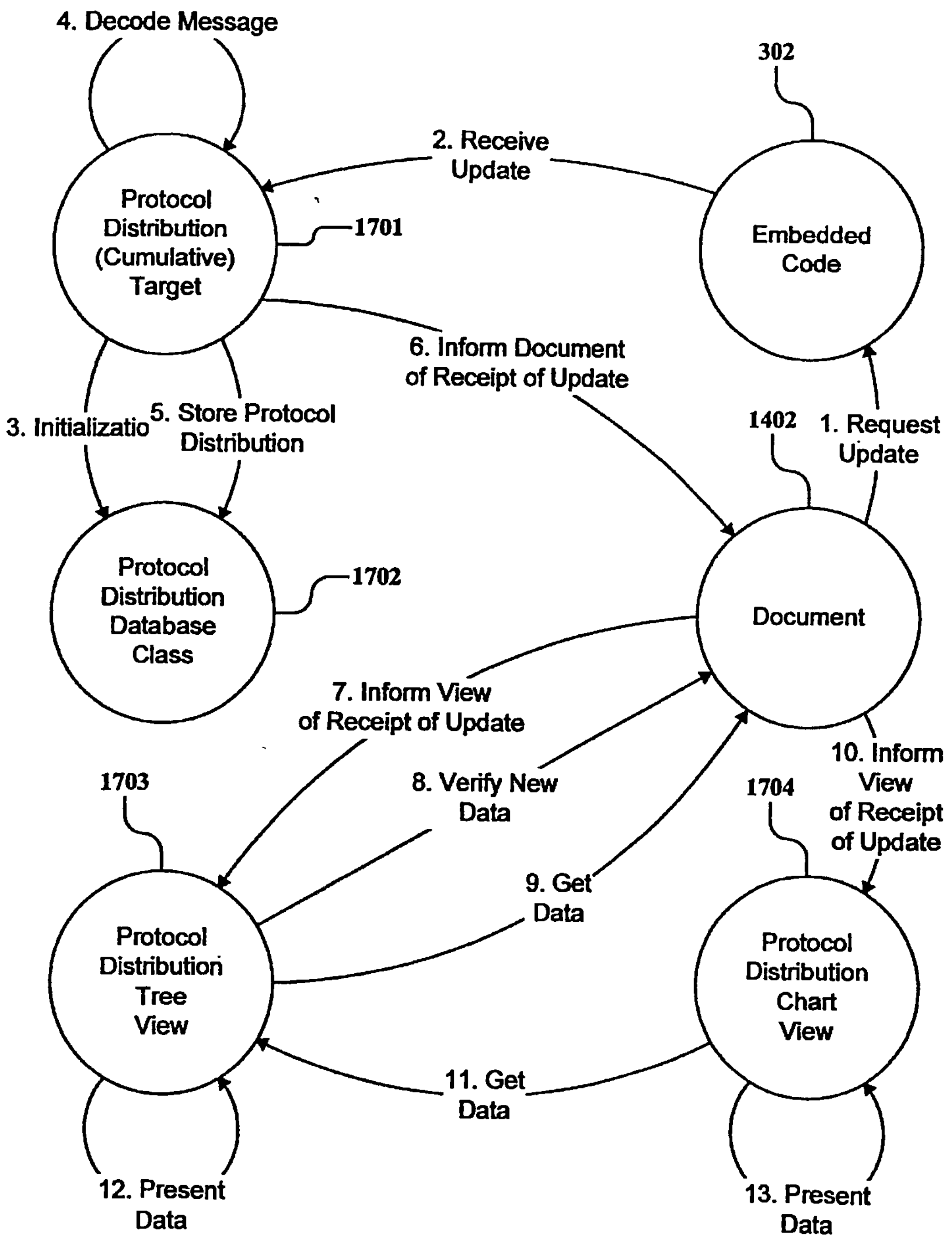
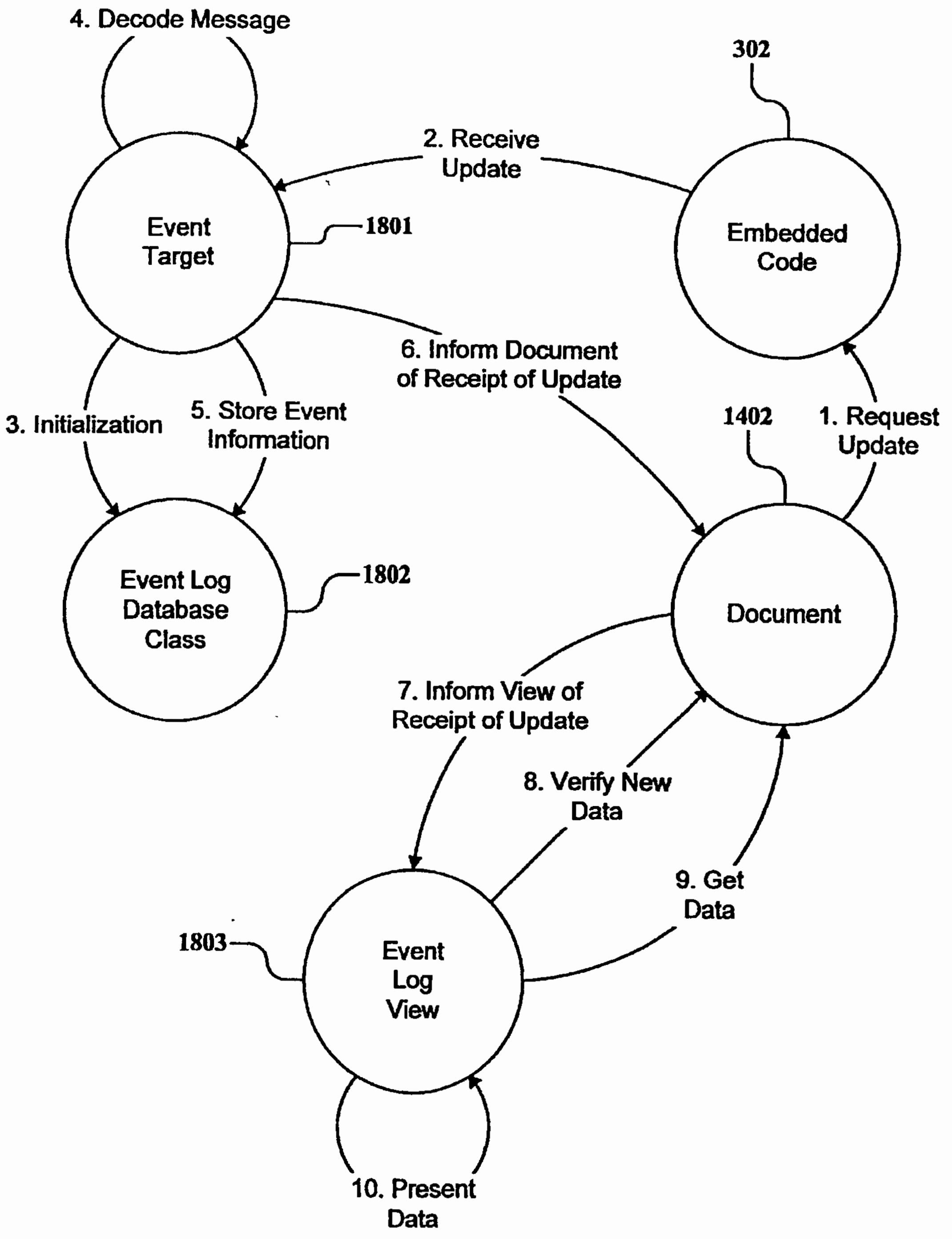
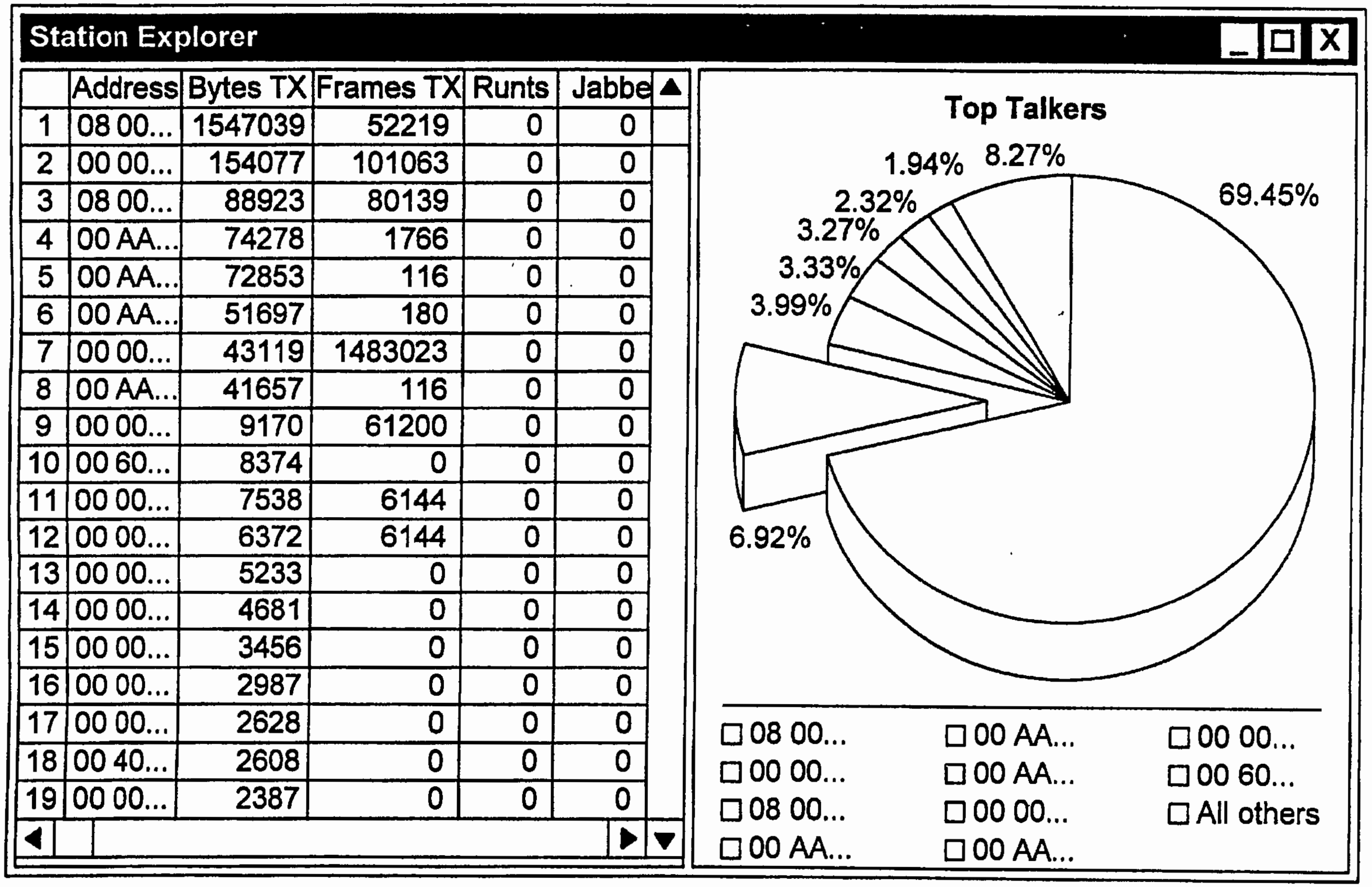

4. Decode Message
302
2. Receive Update
Protocol Distribution (Cumulative) Target
1701
Embedded Code
6. Inform Document of Receipt of Update
3. Initializatio
5. Store Protocol Distribution
1402
1. Request Update
Protocol Distribution Database Class
1702
Document
7. Inform View of Receipt of Update
1703
8. Verify New Data
1704
10. Inform View of Receipt of Update
9. Get Data
Protocol Distribution Tree View
Protocol Distribution Chart View
11. Get Data
12. Present Data
13. Present Data

FIG. 17


4. Decode Message
302
2. Receive Update
Event Target
1801
Embedded Code
6. Inform Document of Receipt of Update
3. Initialization
5. Store Event Information
1402
1. Request Update
Event Log Database Class
1802
Document
7. Inform View of Receipt of Update
8. Verify New Data
9. Get Data
1803
Event Log View
10. Present Data



Station Explorer

	Address	Bytes TX	Frames TX	Runts	Jabbe
1	08 00...	1547039	52219	0	0
2	00 00...	154077	101063	0	0
3	08 00...	88923	80139	0	0
4	00 AA...	74278	1766	0	0
5	00 AA...	72853	116	0	0
6	00 AA...	51697	180	0	0
7	00 00...	43119	1483023	0	0
8	00 AA...	41657	116	0	0
9	00 00...	9170	61200	0	0
10	00 60...	8374	0	0	0
11	00 00...	7538	6144	0	0
12	00 00...	6372	6144	0	0
13	00 00...	5233	0	0	0
14	00 00...	4681	0	0	0
15	00 00...	3456	0	0	0
16	00 00...	2987	0	0	0
17	00 00...	2628	0	0	0
18	00 40...	2608	0	0	0
19	00 00...	2387	0	0	0

FIG. 18

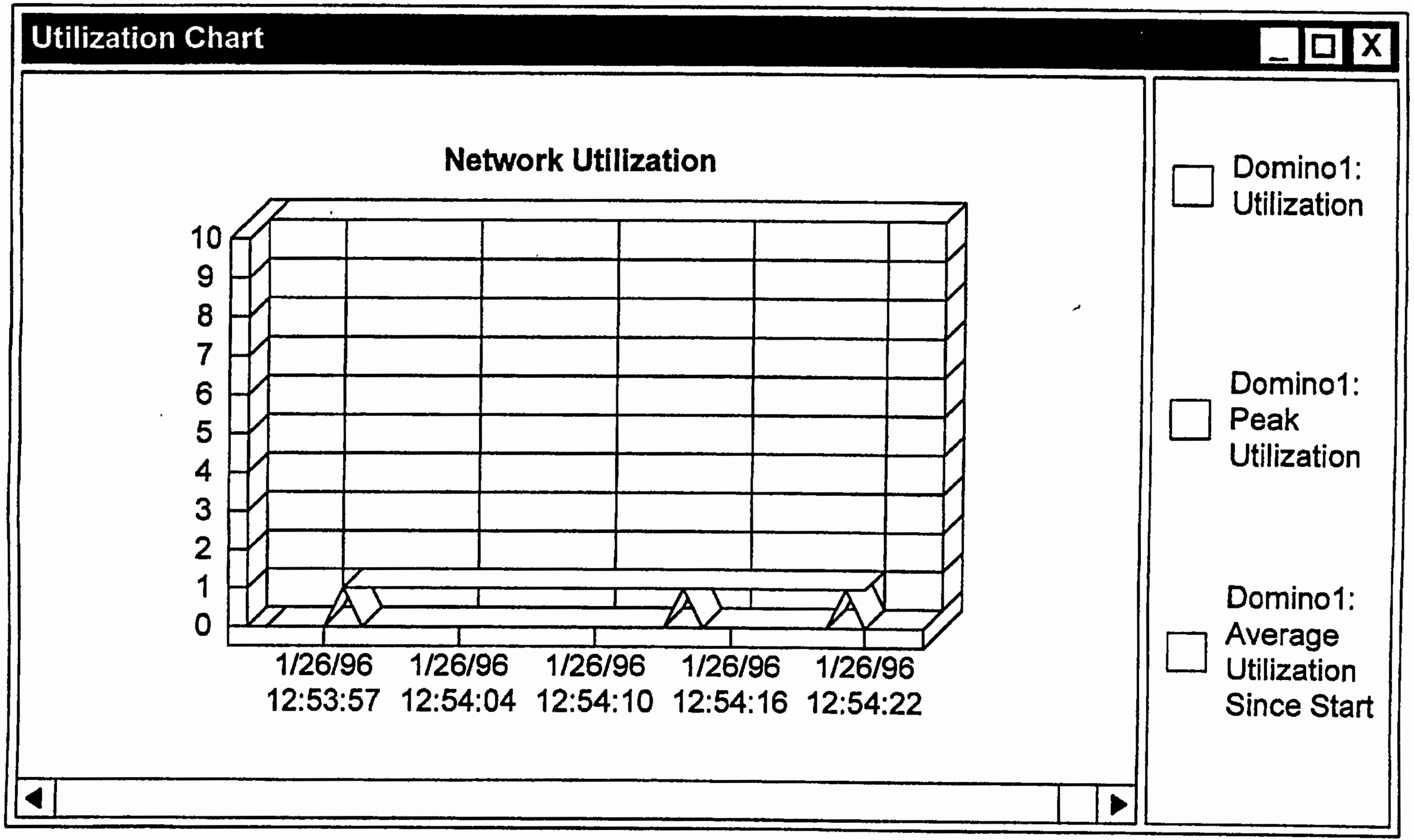

Utilization Chart
_
☐
X
Network Utilization
10
9
8
7
6
5
4
3
2
1
0
1/26/96 12:53:57
1/26/96 12:54:04
1/26/96 12:54:10
1/26/96 12:54:16
1/26/96 12:54:22
□ Domino1: Utilization
□ Domino1: Peak Utilization
□ Domino1: Average Utilization Since Start
◀
▶

FIG. 19A

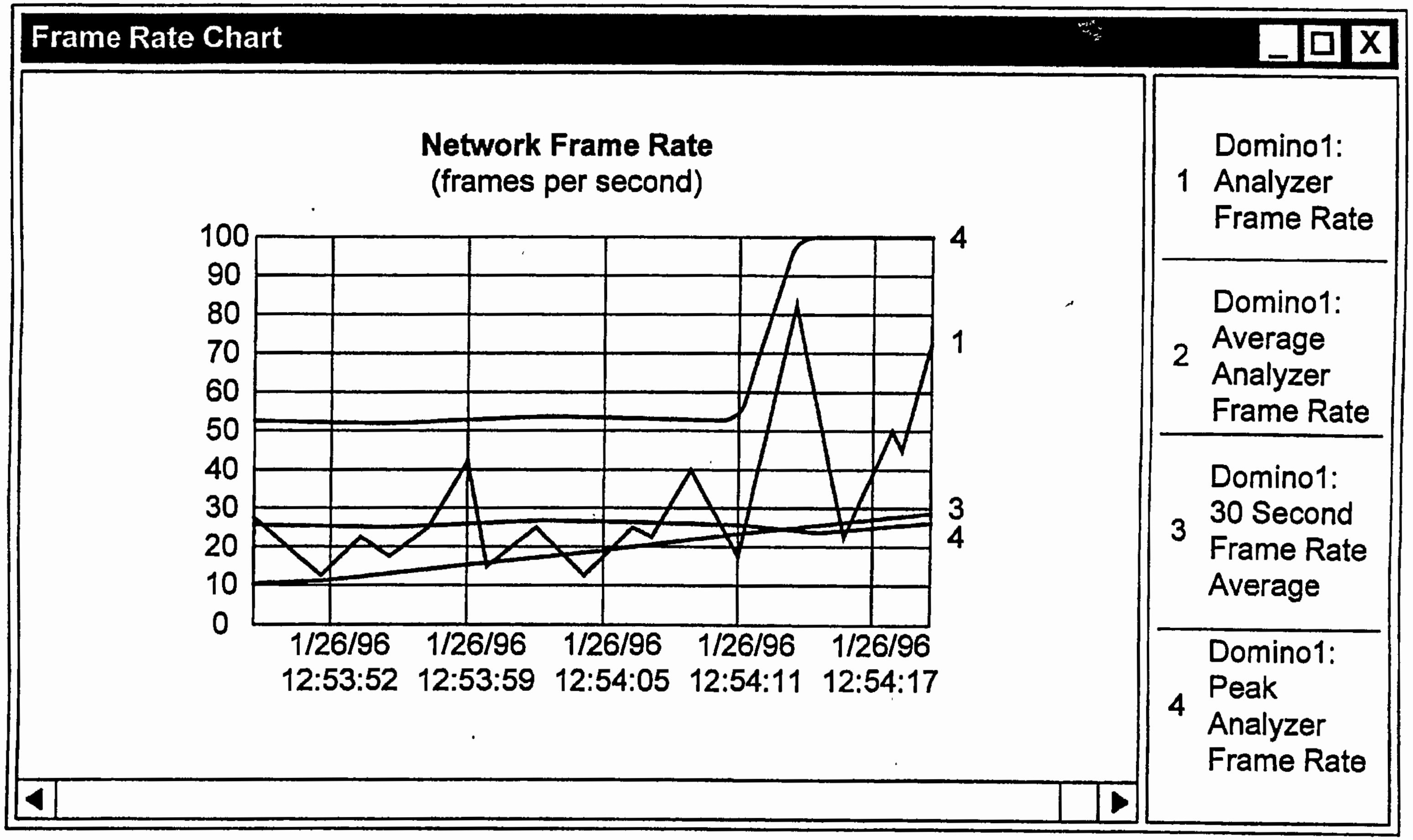

Frame Rate Chart
Network Frame Rate
(frames per second)
100
90
80
70
60
50
40
30
20
10
0
1/26/96 12:53:52
1/26/96 12:53:59
1/26/96 12:54:05
1/26/96 12:54:11
1/26/96 12:54:17
4
1
3
4
1 Domino1: Analyzer Frame Rate
2 Domino1: Average Analyzer Frame Rate
3 Domino1: 30 Second Frame Rate Average
4 Domino1: Peak Analyzer Frame Rate

FIG. 19B

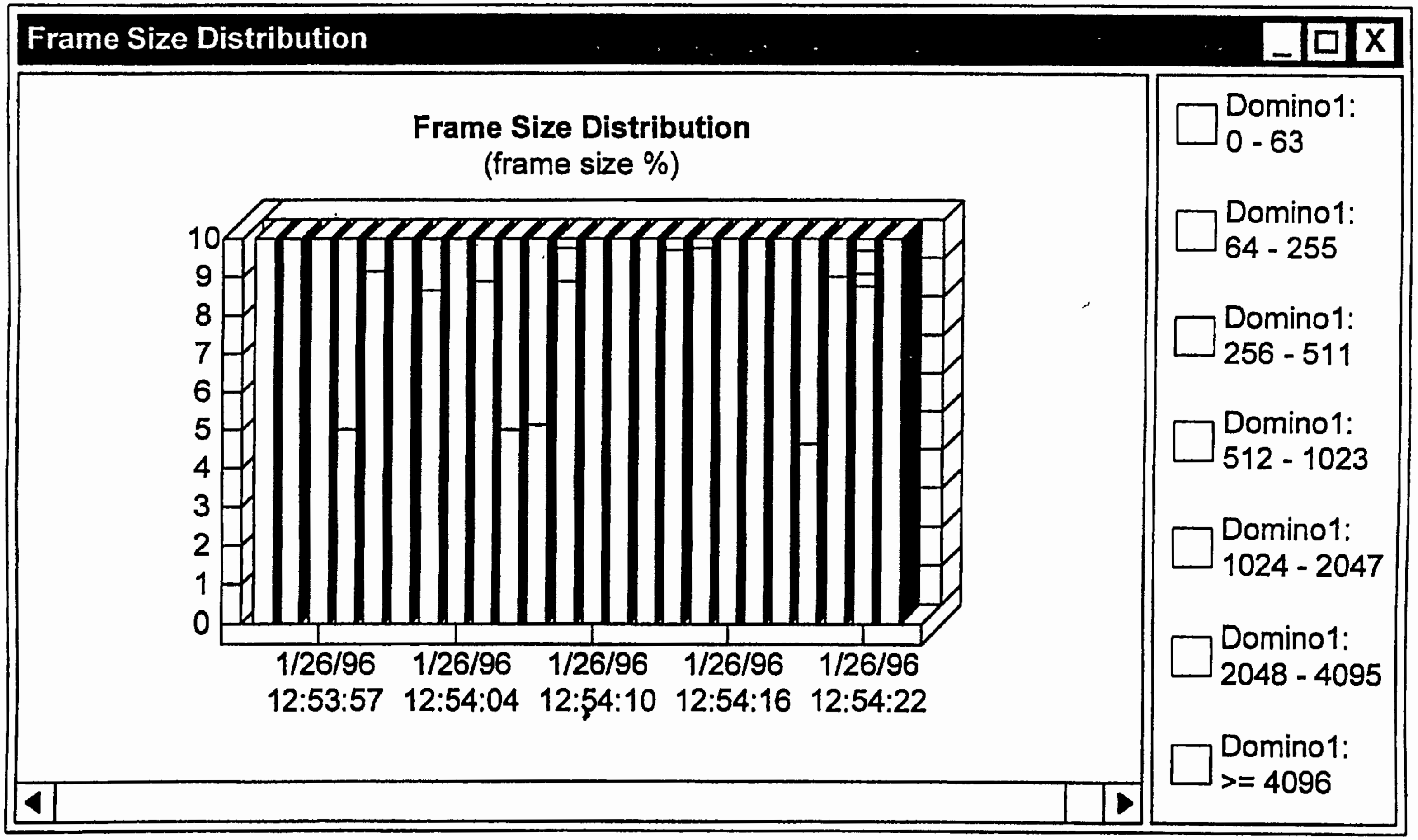
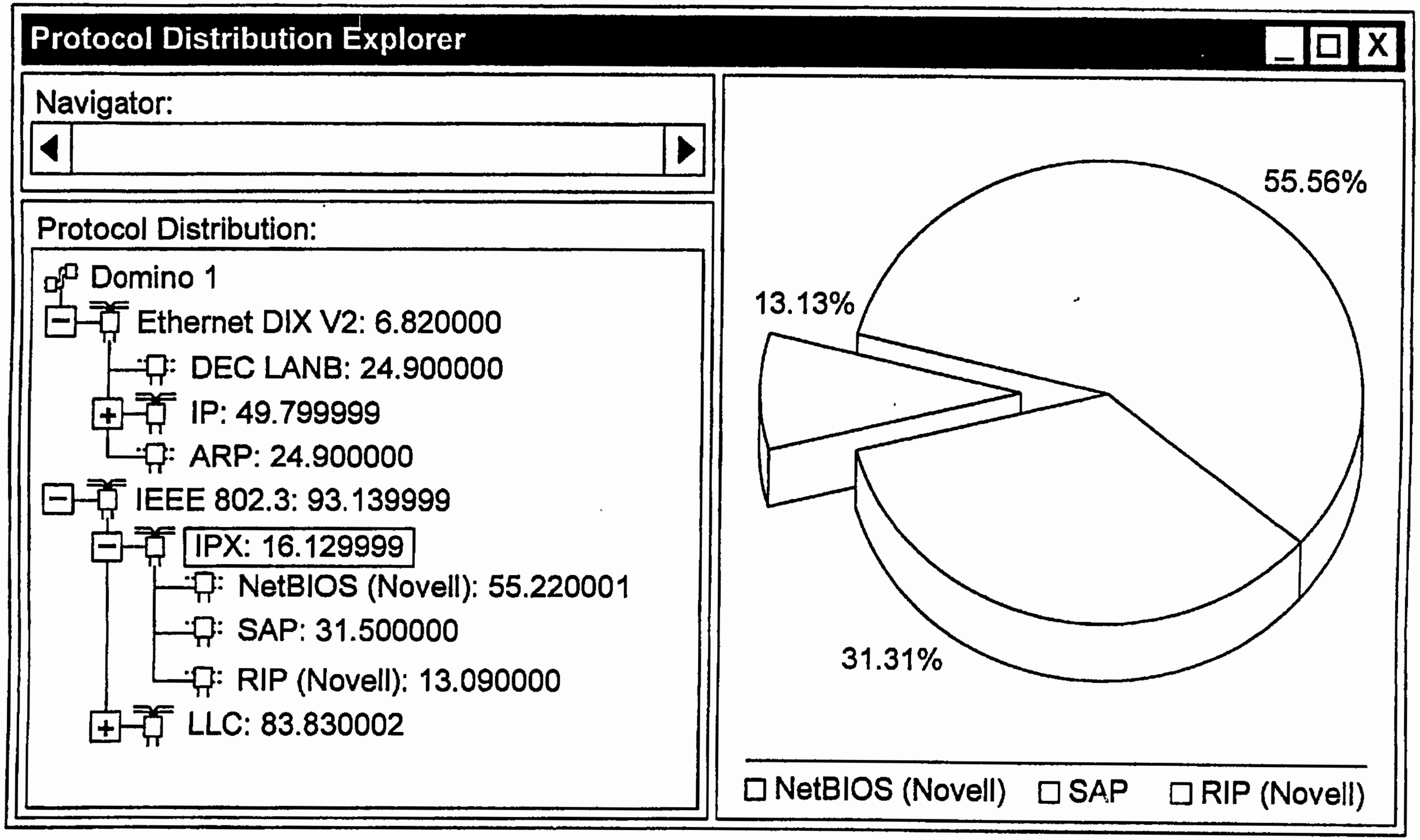

Frame Size Distribution
_ □ X
Frame Size Distribution
(frame size %)
10
9
8
7
6
5
4
3
2
1
0
1/26/96 12:53:57
1/26/96 12:54:04
1/26/96 12:54:10
1/26/96 12:54:16
1/26/96 12:54:22
□ Domino1: 0 - 63
□ Domino1: 64 - 255
□ Domino1: 256 - 511
□ Domino1: 512 - 1023
□ Domino1: 1024 - 2047
□ Domino1: 2048 - 4095
□ Domino1: >= 4096

FIG. 19C



Protocol Distribution Explorer

Navigator:

Protocol Distribution:

Domino 1

Ethernet DIX V2: 6.820000

DEC LANB: 24.900000

IP: 49.799999

ARP: 24.900000

IEEE 802.3: 93.139999

IPX: 16.129999

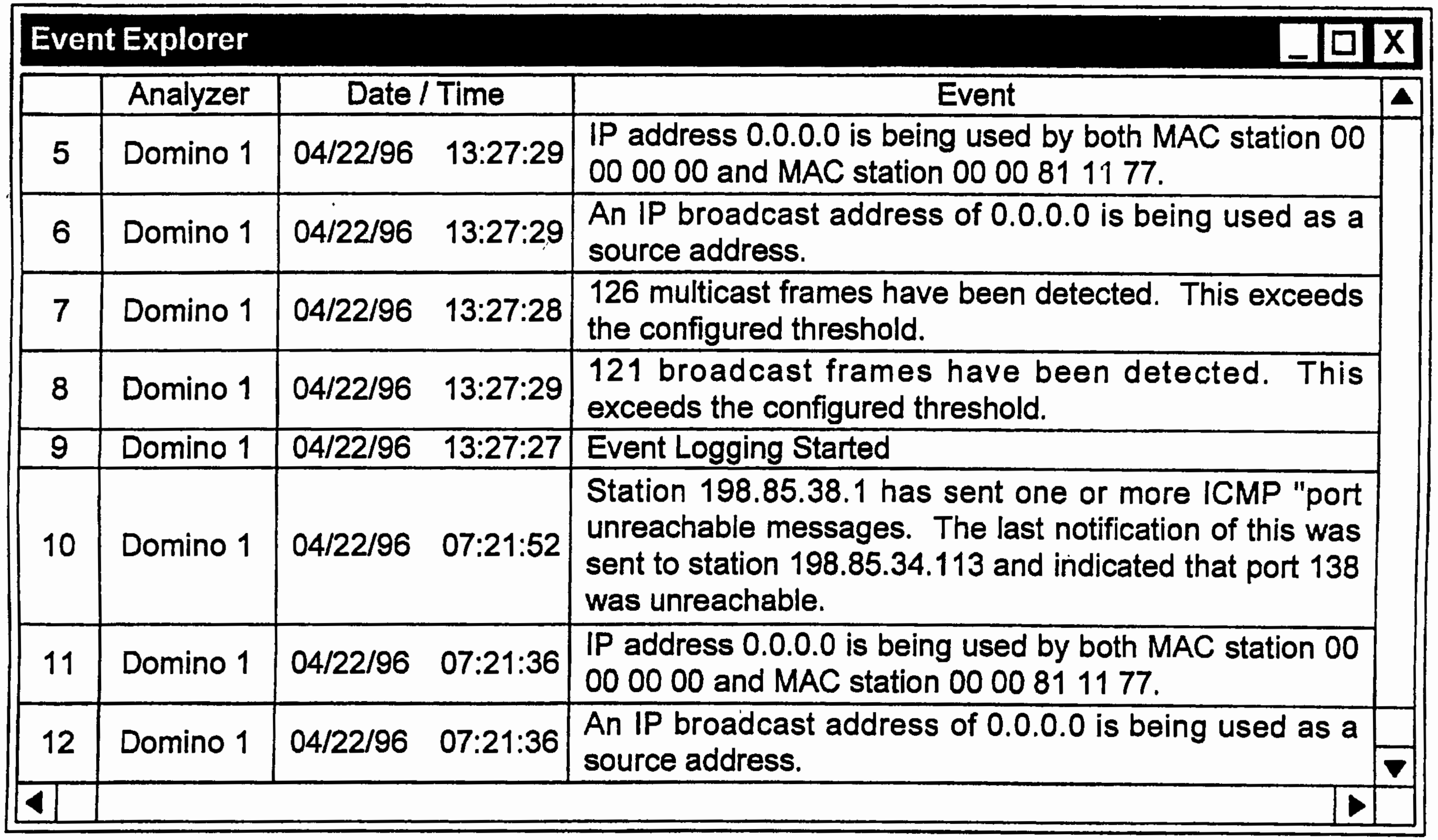
NetBIOS (Novell): 55.220001

SAP: 31.500000

RIP (Novell): 13.090000

LLC: 83.830002

FIG. 20



	Analyzer	Date / Time	Event
5	Domino 1	04/22/96 13:27:29	IP address 0.0.0.0 is being used by both MAC station 00 00 00 00 and MAC station 00 00 81 11 77.
6	Domino 1	04/22/96 13:27:29	An IP broadcast address of 0.0.0.0 is being used as a source address.
7	Domino 1	04/22/96 13:27:28	126 multicast frames have been detected. This exceeds the configured threshold.
8	Domino 1	04/22/96 13:27:29	121 broadcast frames have been detected. This exceeds the configured threshold.
9	Domino 1	04/22/96 13:27:27	Event Logging Started
10	Domino 1	04/22/96 07:21:52	Station 198.85.38.1 has sent one or more ICMP "port unreachable messages. The last notification of this was sent to station 198.85.34.113 and indicated that port 138 was unreachable.
11	Domino 1	04/22/96 07:21:36	IP address 0.0.0.0 is being used by both MAC station 00 00 00 00 and MAC station 00 00 81 11 77.
12	Domino 1	04/22/96 07:21:36	An IP broadcast address of 0.0.0.0 is being used as a source address.

FIG. 21

PROTOCOL ANALYZER FOR MONITORING DIGITAL TRANSMISSION NETWORKS

This application claims priority to provisional application number 60/023,459, filed Aug. 2, 1996.

FIELD OF THE INVENTION

The present invention relates generally to the field of computer and data communications networks and systems and more particularly to protocol analyzers for monitoring and analyzing digital transmission networks.

BACKGROUND OF THE INVENTION

Wide area computer networks ("WANs") first emerged in the 1970's to enable computers to communicate across broad geographic areas. Distributed computing resources, such as personal computers, workstations, servers and printers, have proliferated in recent years due to the declining cost and increasing performance of computer hardware. This has been a key factor in the growth of local area network technology. Local area networks ("LANs") allow increased productivity and utilization of distributed computers or stations through the sharing of resources, the transfer of information and the processing of data at the most efficient locations. As organizations have recognized the economic benefits of using LANs, network applications such as electronic mail, file transfer, host access and shared databases have been developed as means to increase user productivity. This increased sophistication, together with the growing number of distributed computing resources, has resulted in a rapid expansion in the number of installed LANs.

As the demand for LANs has grown, LAN technology has expanded and now includes many different physical connection configurations ("network topologies" or "networks"), such as Ethernet, a LAN that employs a bus topology where the computing resources are connected to a single cable; Token Ring, a LAN that employs a ring topology where the computing resources are connected to a single closed loop cable; and Fiber Distributed Data Interface ("FDDI"), a LAN that supports fiber optic cables where the computing resources are connected in a series of dual rings. These and the many other types of networks that have appeared typically have several different cabling systems, utilize different bandwidths and transmit data at different speeds. In addition, hardware and software systems for LANs usually have different sets of rules and standards ("protocols") which define the method of access to the network and communication among the resources on the network, such as Novell NetWare, IBM NetBIOS, DECNet, AppleTalk and Banyan Vines. More recently, large users of LANs have increasingly sought to integrate local area networks with WANs, and this trend is expected to intensify as inter-network technology advances so as to permit more rapid delivery of advanced multimedia communications utilizing Asynchronous Transfer Mode ("ATM"), an advanced high-speed switching protocol, and other broadband transmission technologies.

Digital data are usually transmitted over a network in frames (also referred to as "data frames" or "packets") which can be of fixed or variable length depending upon the number of bits in the data portion of the frame. Frames usually have headers (e.g., addresses) and footers on the two ends of the frame, with the conveyed data bits being in the middle. These headers and footers are also sometimes referred to as "protocols." The structure of a frame is discussed in more detail below in the section entitled Frame Analysis. The nature and content of the headers and footers are usually dictated by the type of network.

Transmissions from one network computer to another must be passed through a hierarchy of protocol layers. Each layer in one network computer carries on a conversation with the corresponding layer in another computer with which communication is taking place, in accordance with a protocol defining the rules of communication. In reality, information is transferred down from layer to layer in one computer, then through the communication channel medium and back up the successive layers in the other computer. To facilitate understanding, however, it is easier to consider each of the layers as communicating with its counterpart at the same level, in a horizontal direction.

The hierarchy of network layers is illustrated in FIG. 1. The highest network layer is the Application Layer 7. It is the level through which user applications access network services. The Presentation Layer 6 translates data from the Application Layer 7 into an intermediate format and provides data encryption and compression services. The Session Layer 5 allows two applications on different computers to communicate by establishing a dialog control between the two computers that regulates which side transmits, when each side transmits, and for how long. The Transport Layer 4 is responsible for error recognition and recovery, repackaging of long messages into small packages of information, and providing an acknowledgment of receipt. The Network Layer 3 addresses messages, determines the route along the network from the source to the destination computer, and manages traffic problems, such as switching, routing, and controlling the congestion of data transmissions. The Data Link Layer 2 packages raw bits into logical structured packets or frames. It then sends the frame from one computer to another. If the destination computer does not send an acknowledgment of receipt, the Data Link Layer 2 will resend the frame. The Physical Layer 1 is responsible for transmitting bits from one computer to another by regulating the transmission of a stream of bits over a physical medium. This layer defines how the cable is attached to the network interface card within the station computer and what transmission technique is used to send data over the cable. As a message is passed down through the layers, each layer may or may not add protocol information to the message.

As LANs and WANs have increased in number and complexity, networks have become more likely to develop problems which, in turn, have become more difficult to diagnose and solve. Network performance can suffer due to a variety of causes, such as the transmission of unnecessarily small frames of information, inefficient or incorrect routing of information, improper network configurations and superfluous network traffic. Specific network hardware and software systems may also contain design flaws which affect network performance or limit access by users to certain of the resources on the network. These problems are compounded by the fact that most local and wide area networks are continually changing and evolving due to growth, reconfiguration and the introduction of new network topologies, protocols, interconnection devices and software applications.

Increasing numbers of organizations use local and wide area networks, and the accurate and timely transmission and processing of information on LANs and WANs have become vital to the performance of many businesses. Mission-critical applications, such as telemarketing, order-entry, airline reservation systems and bank electronic funds transfer systems, now reside on LANs and WANs. The financial consequences of network problems that adversely affect these applications can be enormous. Without network analysis products which identify how and where data are moving on local and wide area networks, users of these networks have no means to effectively analyze and monitor performance or to isolate problems for prompt resolution.

Network analyzers monitor the digital traffic or bit stream so as to identify and examine principally the headers and footers of each frame in order to analyze the health of the digital network. Hence, they are often called network protocol analyzers. The period of time during which a network is being analyzed is referred to as a "network monitoring session." Typically, protocol analyzers are designed to identify, analyze and resolve interoperability and performance problems across the principal configurations of LAN and WAN topologies and protocols.

The protocol analyzers enable computer network users to perform a wide variety of network analysis tasks, such as counting errors, filtering frames, generating traffic and triggering alarms. There are many examples of digital network transmission protocol analyzer instruments. One such example is shown in U.S. Pat. No. 4,792,753, granted to Iwai on Dec. 20, 1988. Another digital network transmission protocol analyzer, directed particularly to Token Ring networks, collects several types of information about a network, including statistics, events, and network attributes by analyzing sequences of control frame transmissions and is shown in U.S. Pat. No. 5,097,469, granted to Douglas on Mar. 17, 1992. Many of the protocol analyzer instruments are combined with user interfaces having display and keyboard and/or other input capability. The generation and display of certain message traffic characteristics are addressed in U.S. Pat. No. 3,826,908, granted in July 1974 to Weathers et al. U.S. Pat. No. 4,775,973, granted to Toberlin, et al., on Oct. 4, 1988, discloses a method and apparatus for monitoring protocol portions of digital network transmissions and displaying a matrix of traffic from transmitting stations and to destination stations. U.S. Pat. No. 5,375,126 granted to Wallace on Dec. 20, 1994, discloses a system for testing digital data telecommunication networks, with display of fault analysis, comparative viewing of fault-free benchmark data and with provision to offer suggestions as to probable causes of faults. In the network communications monitor of U.S. Pat. No. 5,442,639, granted on Aug. 15, 1995, to Crowder et al., selected frames may be captured in a capture buffer, stored electronically, and/or displayed in real time. U.S. Pat. No. 5,487,073, granted to Urien on Jan. 23, 1996, discloses commanding a communications coupler to perform a set of network function tests. The network status results of the tests are sent to a data-processing unit for display.

Protocol analyzers are produced in two general types. One is larger, less portable and more comprehensive in the scope of tests which it can perform. This type is used primarily by developers and manufacturers of network systems. The other type is smaller, more portable, and often easier to operate and lower-priced, albeit often with some limitations as to the scope of its testing capability. This latter type of protocol analyzer is produced primarily for field service technicians who maintain computer network systems.

A protocol analyzer's monitoring, diagnostic and problem resolution activities are usually under software control. Such software control is exercised by a main central processing unit (CPU), which is usually one or more microprocessors contained within the protocol analyzer itself. The protocol analyzer may also utilize a separate computer controller, such as a "laptop," to facilitate human interface.

To some degree, the software which protocol analyzers use may be characterized as expert system software which facilitates isolation of problems on a network being analyzed. This expert system software may be contained in the protocol analyzer's internal memory or in the separate computer controller. The utility, efficiency, comprehensiveness, and ease of use of a protocol analyzer, particularly one designed for use by a field technician, is in large part directly proportional to the corresponding capabilities of the software in the protocol analyzer and even in its computer controller.

Current protocol analyzers for use by field technicians have numerous limitations. One such limitation is the inability to analyze and display comprehensive network transmission information in real-time (as the transmissions occur). When analysis of network transmissions must be done off-line, the likelihood that an important network occurrence or "event" will be missed is significantly increased.

In addition, current protocol analyzers do not present network transmission information in sufficiently meaningful or detailed ways, nor do they allow for on-line comparison of current network performance to prior network performance. For example, it would be useful if more meaningful displays of the numerous types of statistics related to the network as a whole or just a given station on the network were available to the user in juxtaposition with other information. Also, many users would like to see complicated information and detailed protocol distribution statistics displayed in a manner that is easier to use and easier visually to understand. Display to the user of more detailed information about anomalies or "events" that occur on the system would be useful to a user, especially if displayed in a more usable form and in "real time" and accumulated over a network monitoring session. Certainly, conveniently-displayed troubleshooting assistance would be helpful, as would visual reporting in "real time" and accumulated over an analysis session. Off-line analysis of selected frames captured during a network monitoring session could be more conveniently displayed to the user.

Finally, while protocol analyzers of the prior art provide reasonable diagnostic capability, they do not guide the field technician through event analysis and the appropriate solutions. In general, these limitations combine to prevent effective guidance to the field technician in actually analyzing and solving the network problem.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide a new and improved protocol analyzer capable of displaying station level statistics, displaying real time event detection, creating baseline network performance information and comparing said baseline information with real-time performance information and displaying to a user the results of that comparison, pre-programming monitoring sessions, generating reports in conjunction with analyzing digital transmission networks, all in real time.

In accordance with one embodiment of the present invention, the operation of a protocol analyzer includes one or more of the following: monitoring, in real time, the transmission of data packets having protocol portions and data portions; identifying the protocol portions of said packets in real time; analyzing, in real time, the protocol portions of said packets to ascertain relevant information; storing said information to a database in real time; sorting said information in real time, according to station level parameters; statistically analyzing said sorted information in real time to obtain statistical information; displaying said statistical information in real-time reports, displaying said statistical information in report formats selected by an operator; displaying real time performance of the network simultaneously with baseline network performance; simultaneously displaying statistical information gathered from a plurality of protocol analyzer instruments; pre-programming the monitoring of the transmission of data packets wherein the operator may select the duration of the network monitoring session; monitoring in real time one or more selected and assorted network parameters and comparing the results of said analysis with arbitrary threshold values for said parameters to determine if the transmission on the network is exceeding said threshold so as to constitute an event; analyzing in real time said sorted information to calculate the probabilities of the possible causes of said ascertained events; and displaying in real time the one, or more possible causes of said event.

It is another object of the present invention to analyze and meaningfully display the statistics of the occurrence and distribution of protocols encapsulated within the several levels of the several data frames analyzed by a protocol analyzer instrument.

In accordance with another embodiment of the present invention, the operation of a protocol analyzer includes one or more of the following: monitoring, in real time, the transmission of data packets having protocol portions and data portions; identifying the protocol portions of said packets in real time; analyzing, in real time, the protocol portions of said packets to ascertain relevant information; storing said information to a database in real time; sorting said information according to protocol distribution criteria; statistically analyzing said sorted information; and displaying said statistical information.

In accordance with yet another embodiment of the present invention, the operation of a protocol analyzer includes one or more of the following: monitoring, in real time, the transmission of data packets having protocol portions and data portions; identifying the protocol portions of said packets in real time; analyzing, in real time, the protocol portions of said packets to ascertain relevant information; storing said information to a database in real time; sorting said information according to ISO layer; sorting said information according to protocol sub-families; statistically analyzing said sorted information; and displaying said statistical information in a protocol-tree format.

It is yet another object of the present invention to analyze, store the analysis results and display the analysis results, in real time, of digital data transmission comprising data frames having protocol portions and data portions, without the need to wait for the later analysis of protocol portions of frames, which protocol portions were stored in real time. This and other objects of the present invention are achieved by use of a RISC (Reduced Instruction Set Computer) processor to analyze the protocol portion of each frame, in real time, followed by contemporaneous statistical analysis of the RISC (Reduced Instruction Set Computer) processor analysis of the protocols of several successive frames, followed by substantially simultaneous storage and display of the statistical analysis results.

It is still another object of the present invention to store the results of the analysis of digital data transmission in a database capable of storing and retrieving the analysis results to permit display in real time. This and other objects of the present invention are achieved by use of an object oriented database and object oriented application programming to access said object oriented database.

It is still yet another object of the present invention to log, store, and display digital data transmission events in real time. This and other objects of the present invention are achieved by recognizing, in a protocol analyzer instrument, the occurrence of an event, periodically polling the protocol analyzer instrument for, among other information, a record of events that occurred since the last polling, transmitting, to a user interface, a message containing information about the events that occurred since the last polling, receiving the new event information in an event target ("target" is a term used to identify a software device to which data can be sent for storage, forwarding, or processing), storing the new event information in an event log object in an event log database class, informing a document of receipt of new event information, the document informing an event log view of receipt of new information, obtaining confirmation of new event log information and a pointer thereto in the database, and incorporating the new event information into a display of event log information.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be more fully understood by reference to the following detailed description when considered in conjunction with the following drawings wherein like reference numbers denote the same or similar portions or processes shown throughout the several Figures in which:

FIG. 1 is an illustration of the hierarchy of network protocol layers;

FIG. 2 is a diagram describing the structure of an Ethernet data frame;

FIG. 3 is a diagram which illustrates the flow of data, analysis, and control in accordance with the present invention;

FIG. 4 is a flowchart illustrating the process by which statistics for individual stations on a network (station-level statistics) are calculated;

FIG. 5 is a diagram illustrating the structure of an entry in the protocol distribution array within a digital memory;

FIG. 6 is a diagram illustrating the structure of the message for a protocol distribution update;

FIG. 7 is a flowchart illustrating the method by which protocol distribution is calculated;

FIG. 8 is a flowchart illustrating the method by which the Network Statistic "Network Frames Received" is calculated;

FIG. 9 is a flowchart illustrating the method by which Network Statistic "Analyzer Frames Received" is calculated;

FIG. 10 illustrates the structure of an entry in the Event Log Array;

FIG. 11 illustrates the structure of an Event Update Message;

FIG. 12 illustrates the structure of a Network Statistics Update Message;

FIG. 13 illustrates the structure of the user interface;

FIG. 14 is a scenario diagram for a station-level statistics update;

FIG. 15 is a scenario diagram for a network statistics update;

FIG. 16 is a scenario diagram for a protocol distribution update;

FIG. 17 is a scenario diagram for an event update;

FIG. 18 illustrates the visual appearance of an example of a split-screen display of station-level statistics;

FIGS. 19A, 19B, and 19C illustrate three examples of the appearance of chart display formats for network statistics;

FIG. 20 illustrates an example of the appearance of a split-screen display of protocol distribution; and

FIG. 21 illustrates the appearance of a display screen listing of events.

DETAILED DESCRIPTION OF THE INVENTION

I. Introduction

The following detailed description is divided into sections which have section titles to indicate the general nature of the information that follows. The sections and their titles are intended solely to assist in organization of the description and to aid the reader. They are not intended to indicate that information suggested by any one section title is not contained in any other section.

Where the description of the design and operation of the present invention is illustrated by use of an example which is specific to a particular network topology, it may be presumed unless stated otherwise that the network topology is Ethernet. Limiting examples to Ethernet networks is intended only to provide consistency in order to facilitate understanding and is not meant to indicate a limitation of the suitability of the present invention for analyzing other network topologies such as token ring, FDDI, frame relay, etc.

II. Overview of the Implementation of the Invention

The preferred embodiment of the present invention comprises a hardware implementation and a software implementation.

A. Software Implementation

The software implementation of the present invention performs two functions. The first is to perform meaningful statistical calculations on the protocol information retrieved from the network. The portion of the software implementation responsible for performing these calculations will be referred to hereinafter as the "embedded code."

The second function performed by the software implementation of the invention is to provide the software with means for interaction between the protocol analyzer and the operator. Such interaction includes the displaying of the data calculated by the embedded code as well as responding to operator commands. The portion of the software implementation which performs this function will hereinafter be referred to as the "user interface" or "UI." The user interface is preferably coded in the Microsoft Visual C++ programming language produced by Microsoft Corporation at One Microsoft Way, Redmond, Wash. 98052-9953, and operates in the Microsoft Windows 3.1 and Microsoft Windows 95 operating systems, also produced by Microsoft Corporation.

B. Hardware Implementation

The hardware implementation of the present invention likewise performs two functions. First, it provides a physical platform for execution of the embedded code and for interfacing with the network being monitored. This portion of the hardware implementation of the invention will hereinafter be referred to as the "protocol analyzer instrument." The present invention may comprise a plurality of protocol analyzer instruments (see FIG. 3), each having at least one RISC processor and each monitoring a different network or segment of a network or monitoring the same network or segment but at a different port or station on the network.

The second function performed by the hardware implementation is to provide the physical means for the operation of the user interface. Such means include input devices (such as a keyboard, a mouse, a trackball, etc.) and a display device (such as a cathode ray tube monitor or a liquid crystal display). In the preferred embodiment of the invention, this second function is performed by a laptop personal computer (PC) containing an Intel 80486 or Pentium processor operating at 25 MHz or faster, preferably eight megabytes or more of random access memory, a hard disk drive with at least about forty-five megabytes of free disk space, a 3.5 inch floppy disk drive, a bi-directional parallel communication port, a keyboard, and a pointing device such as a trackball, joystick, or mouse. This second portion of the hardware implementation of the invention will be referred to hereinafter as the "PC."

In the preferred embodiment, the PC is connected to one or more protocol analyzer instruments through the PC's parallel communication port. The software implementation (both the embedded code and the user interface) of the invention is stored in a storage device (such as a hard disk drive, a magnetic tape drive, or other similar medium) on the PC. When the operator activates the protocol analyzer instrument, an initialization process takes place in which the embedded code is downloaded from the PC's storage device through the parallel communication port to the protocol analyzer instrument(s).

C. Protocol Analyzer Instrument

Except for the features described herein, the protocol analyzer instrument can be similar to the hardware implementations of conventional protocol analyzers. See U.S. Pat. No. 4,792,753 (mentioned above). In the preferred embodiment of the present invention, the protocol analyzer instrument is a DominoLAN DA-320 Internetwork Analyzer manufactured by Wandel & Goltermann Technologies, Inc. at 1030 Swabia Court, Research Triangle Park, N.C. 27709-3585. (DominoLAN is a trademark of Wandel & Goltermann Technologies, Inc.) The Domino Getting Started Guide, the Domino Operating Guide, the DominoLAN Toolbox Applications, and the Release Notes for the relevant release, all of which are included with the DA-320 analyzer, are hereby incorporated by reference as if fully set forth herein.

The protocol analyzer instrument preferably comprises two hardware modules, a network interface (NI) module and a protocol analysis (PA) module which preferably occupy the same convenient physical cabinet. Each module is controlled by its own INMOS T425 transputer processor operating at 25 MHz and using a 32-bit word RISC (Reduced Instruction Set Computer) architecture ("RISC processor"), manufactured by SGS Thompson Corporation, INMOS, Ltd., 1000 Aztec West, Alnondsbury, Bristol, BS12 4SQ, UK.

These RISC processors are responsible for execution of the embedded code when the protocol analyzer instrument is in use. The use of a processor with a limited instruction set, such as a RISC processor, results in increased processing speed. This increased processing speed allows both the NI and PA modules of the protocol analyzer instrument to perform real time analysis of the network transmissions. While the preferred embodiment utilizes RISC processors to achieve the desired processing speed, alternatives such as the Intel 960 processor manufactured by Intel Corporation, 2200 Mission College Blvd., Santa Clara, Calif. 95052; and the PowerPC processor manufactured by Motorola, Inc., P.O. Box 20912, Phoenix Ariz. 85036, will be readily apparent to a person having ordinary skill in the art. (PowerPC is a registered trademark of International Business Machines Corporation.) The scope of the invention, therefore, should not be limited to the description of the preferred hardware implementation contained herein.

The NI module is preferably equipped with 512 kilobytes of static random access memory (SRAM), while the PA module preferably has four megabytes of dynamic random access memory (DRAM) and is expandable to sixteen megabytes. The preferred protocol analyzer instrument also contains a LAN card printed circuit pack (art number 82C581), which comprises two LAN chips (part number 82C585), purchased from 3Com Corp., and a plurality of hardware counters used to count the number of frames and bytes detected on the network.

D. Data Flow Overview

FIG. 3 illustrates an overview of the flow of information about the operation of a network 301. Data-bearing frames (see FIG. 2) are transmitted over the Network 301 and are received and analyzed by Embedded Code 302 executed by a Protocol Analyzer Instrument 304 using its one or more RISC processors 314 and hard-wired analyzer circuits within the Protocol Analyzer Instrument. The results of that protocol analysis are then available to be sent, as commanded by the user, to a software-based User Interface 303 running on the PC 305 for storage and presentation to the user. The User Interface then presents the analysis results to the user via the PC's display device 318. The User Interface 303 also passes the user's commands (e.g., network parameters to be monitored, sampling rate, etc.) to the Embedded Code 302.

FIG. 3 also illustrates that the PC 305 contains a mass memory device 317, sometimes referred to as a direct access storage device. This is the hard disc drive of the preferred embodiment of the PC 305 that is used, inter alia, to implement the preferred embodiment of the present invention. The User Interface 303 works with a POET object-oriented database program 310 selectively to store, on the mass memory or hard drive 317, the results of the analyses that are performed by the Protocol Analyzer Instrument 304 and which are then periodically uploaded to the PC 305.

III. Frame Analysis

The format of a data frame varies slightly depending on the network type (i.e. token ring network, ethernet, etc.) but the analysis of the frame is basically the same. For example, the format of an ethernet network data frame is illustrated in FIG. 2. The frame begins with an eight-byte Preamble field 208 which is used for synchronization. This is followed by the Destination address 209 (the address of the station which is to receive the data frame). Next is the Source address field 210 (the address of the station which sent the data frame). The address fields contain the Medium Access Control ("MAC") addresses of the source and destination stations. The MAC address is a unique address hard wired into the station's network interface card (NIC). Each address field is six bytes in length. The Type field 211, which follows the address information, is a two-byte field that specifies the higher layer protocol used in the Data field. The Data field 212, which is the only variable-length field, is next and ranges from 46 to 1500 bytes. It contains the higher level protocols currently in use as well as the data being transmitted. Last is a four-byte Frame Check Sequence ("FCS") field 213, which is used for error detection. The term "frame length" refers to the total number of bytes contained in the frame less the number of bytes in the Preamble 208. The contents of each field are identified by the embedded code executed on the protocol analyzer instrument. The exact method by which relevant information is extracted from a

frame would be readily apparent to a person having ordinary skill in the art of programming software for protocol analyzers.

In the present invention, frame analysis is performed by the embedded code executed by the protocol analyzer instrument. By performing frame analysis on the protocol analyzer instrument, which preferably contains two RISC processors, the analysis of the frames can be accomplished in real-time. The contents of each frame received during a network monitoring session are temporarily stored in the memory located on the protocol analyzer instrument. The portion of this memory which stores the contents of received frames will be referred to hereinafter as the "capture buffer." The frames stored in the capture buffer will be referred to hereinafter as "captured frames." The contents of the capture buffer are continuously updated. When the buffer is filled, the oldest captured frames are discarded and replaced with newly captured frames.

IV. Filtering

Filtering is the process by which the user can select certain types of frames to be analyzed. The user can specify certain parameters that frames must meet before the frames are sent to the NI module. Filtering is preferably accomplished using a hardware filter, contained within the NI module, such as the filter disclosed in a copending U.S. patent application Ser. No. 08/384,855, filed on Feb. 7, 1995, in the name of Bradley Anderson, which is incorporated by reference to the same extent as if fully reproduced herein now issued as U.S. Pat. No. 5,590,159.

The hardware filter compares incoming bit sequences to the bit sequence which corresponds to the user-defined parameters and only frames containing bit sequences which match the bit sequence corresponding to the user defined parameters are sent to the NI module of the protocol analyzer instrument. These frames are the only frames which are analyzed. As an arbitrary example of the filtering operation, only frames addresses to station A are to be analyzed and/or only frames transmitted by station B are to be analyzed.

V. Embedded Code

A. Station-Level Statistics

Statistics for each station on a network will be referred to hereinafter as "station-level statistics." Station-level statistics such as, but not limited to, number of bytes transmitted, number of frames transmitted, number of bytes received, number of frames received, and total number of errors generated by that station are all calculated by the embedded code running on the protocol analyzer instrument.

As station-level statistics for each station operating on the network are calculated, they are stored in an array called the "station list array" in the memory of the protocol analyzer instrument. An array is a data structure used to store data. Many other data structures, which can also be used to store data, are well known to persons having ordinary skill in the art of programming. The use of the term "array" throughout this specification is not meant to be limited strictly to arrays; because, many of these other data structures would also suffice.

The station list array contains: the station address, traffic statistics (bytes received, bytes transmitted, frames received, and frames transmitted, etc.), and error statistics for each station which is or has been active on the network during the network monitoring session. The type of error statistics calculated will vary depending on the type of network.

FIG. 4 illustrates the process by which station-level statistics are calculated for an ethernet network. Station-level statistics which are unique to other network topologies, such as FDDI, Token Ring, frame relay, etc., are calculated in a manner which is analogous to the process described below.

After the start 401 of the station-level statistics calculation, receipt of a frame is recognized at programming step 402 by the protocol analyzer instrument. Next, the address information of each frame is used preferably to identify the destination address at programming step 403 and to identify the source address at programming step 404 for each data frame sent over the network. The destination address portion 209 (see FIG. 2) of the frame is identified, and the bytes contained in that portion of the frame are examined in order to ascertain the destination station to which the frame has been addressed. The source address portion 210 of the frame is identified, and the bytes contained in that portion of the frame are examined in order to ascertain the source station from which the frame was sent.

Preferably, the next step in the calculation of station-level statistics is programming step 405 in which the value of the Frame Check Sequence ("FCS") field 213 is identified in programming step 405. In an ethernet frame, the FCS is contained in the last four bytes of the frame. The FCS is a four-byte cyclic redundancy check ("CRC") or checksum which is calculated by the source station. The source station calculates the FCS by performing a well-known mathematical function on the bits in the Destination 209, Source 210, Type 211, and Data 212 fields of the frame. The FCS is used for purposes of error detection.

The length of the frame is preferably determined in programming step 406 by summing the total number of bytes in the Destination 209, Source 210, Type 211, Data 212 and FCS 213 fields.

The next step 407 is to determine whether there is an entry corresponding to the destination address of the frame in the station list array in the memory of the protocol analyzer instrument. If that particular destination station has previously received or sent any frames during the network monitoring session, there will be an entry corresponding to that destination station's address in the station list array. If that destination station has not yet received or sent any frames during the network monitoring session, there is no entry for that destination address in the station list array. If there is no entry corresponding to that particular destination address in the station list array, an entry corresponding to that destination address is created by programming step 408.

The frames_received array variable of the station list array entry corresponding to the destination address is incremented by one at the programming step 409.

Where used, an underscore within a term denotes a variable name as opposed to a value or definition. Also, an array is a memory structure.

Similarly, the bytes_received array variable of the station list array entry corresponding to the destination address is incremented in step 410 by the frame length of the current frame.

The step 411 determines whether there is an entry corresponding to the source address of the frame in the station list array. If the source station has previously received or sent any frames during the network monitoring session, there will be an entry corresponding to the source station's address in the station list array. If the source station has not yet received or sent any frames, there is no entry for the source address in the station list array. If there is no entry corresponding to the source address in the station list array, an entry corresponding to the source address is created by step 412.

The frames_transmitted array variable of the station list array entry corresponding to the source address is incremented by one at programming step 413. Similarly, the bytes_transmitted array variable of the station list array entry corresponding to the source address is incremented by the frame length of the current frame in step 414.

The next steps involve updating the error_statistics array variable of the entry in the station list array corresponding to the source address. The error_statistics array variable is actually a subarray whose length depends upon the number of types of errors detected for the particular network topology. It contains the error_id and the number_of errors for each type of error detected for the corresponding station. The error_id is an arbitrary, predefined code which represents one of several types of errors that the protocol analyzer instrument is equipped to recognize. For an ethernet network, these errors include but are not limited to "jabbers," "runts," "alignment errors," and "FCS errors." Each of these errors is discussed in detail below. Number_of_errors represents the number of occurrences of the particular error type attributable to the corresponding station.

The first step 415 in updating the error_statistics subarray of the entry in the station list array corresponding to the source address of the current frame is to determine whether the length of the current frame is greater than the maximum 1518 bytes permitted in an Ethernet frame. Such overly-long frames are commonly referred to as jabbers. Jabbers originate from a source station that will not stop transmitting. If a frame's length is greater than 1518 bytes, it is likely that the source station is defective. If the current frame is a jabber, the error_statistics subarray of the entry in the station list array corresponding to the source address is updated accordingly at programming step 416.

Step 417 determines whether the frame length of the current frame is less than 64 bytes. Such frames are commonly referred to as runts. Runts are short frames which may also indicate that the source station is defective. If the current frame is a runt, the error_statistics subarray of the entry in the station list array corresponding to the source address is updated accordingly in step 418.

Programming step 419 determines whether the current frame is byte-aligned. A frame is said to be byte aligned if the frame size in bits is evenly divisible by eight. For this purpose, the frame size is said to be equal to frame length (expressed in bits) plus the length of the preamble (also expressed in bits). That is, the existance of an arithmatic remainder from the division by eight of the number of bits in the frame plus preamble indicates that the frame does not contain a whole number of bytes.

The determination of whether a frame is byte-aligned is done by the LAN chip on the protocol analyzer instrument (mentioned above under Protocol Analyzer Instrument). If the current frame is not byte aligned, an alignment error has occurred. The error_statistics subarray, of the entry in the station list array corresponding to the source address, is updated accordingly by programming step 420.

As discussed above, the source station calculates the FCS by performing a mathematical function on the bits in the Destination, Source, Type, and Data fields of the data frame. The LAN chip, also discussed above, performs the same mathematical function and compares the results to the contents of the FCS. If the two do not match, an FCS error has occurred, indicating that the frame is corrupt. Step 421 examines the results of that calculation and comparison by the LAN chip in order to determine whether one or more bits of the current frame may have been corrupted in transmission.

If the current frame contains an FCS error and is thus corrupt, the error_statistics subarray of the entry in the station list array corresponding to the source address is updated accordingly in step 422.

After all of the above station-level statistics and any other desired station-level statistics have been calculated, the final step involves transmitting the information stored in the station list array to the PC for use by the user interface. This transmission is facilitated through the use of a "message," which is the preferred method of communication among the various hardware and software components of the preferred embodiment of the present invention.

The message for station-level statistics consists of a header and the contents of the station list array. The header identifies the destination for the message and the type of message, in much the same format as the messages illustrated in FIGS. 6, 11, and 12. In this case, the message would be a station-level statistics update message. The contents of the station list array are placed in the message in the order in which they were initially placed in the station list array.

When the User Interface (UI) software in the PC requests information on station-level statistics, the message is sent from the protocol analyzer instrument to the PC for use by the user interface. A person having ordinary skill in the art of digital transmission protocol analyzers will know that station-level statistics peculiar to other network topologies can be calculated in a similar manner.

B. Network Statistics

Statistics based upon network performance as a whole will be referred to hereinafter as "network statistics." Network statistics are calculated by the embedded code running on the protocol analyzer instrument. Network statistics may be cumulative (calculated over the entire network monitoring session, which might typically be a twenty-four-hour period) or per sampling period (calculated over a sampling period specified by the user, which might typically be a one-second period).

Network statistics may also vary somewhat between the various network topologies. For example, network statistics calculated for an ethernet network include number of Network Frames Received, Network Frame Rate, number of Analyzer Frames Received, Analyzer Frame Rate, Peak Analyzer Frame Rate, Peak Frame Rate Timestamp, Average Frame Rate, Average 32-Second Frame Rate, Utilization, Average Utilization, Peak Utilization, Peak Utilization Timestamp, number of Broadcast Frames Received, number of Multicast Frames Received, Frame Size Distribution-Cumulative, Frame Size Distribution-Sample, number of Network Collisions, number of Alignment Errors and the numbers of Jabber, Runt, and FCS Errors. Each of these statistics is discussed below.

Referring now to FIG. 8, the number of Network Frames Received is calculated by a hardware counter on the protocol analyzer instrument. From the Start step 801, the protocol analyzer instrument detects receipt of a network frame in programming step 802. The hardware counter is connected directly to the network line and is able to count every frame traveling over the network, i.e. Network Frames Received. For each frame detected, the hardware counter is incremented in step 803. This process is repeated until the sampling period has expired, step 804. When the network monitoring session has ended, it ends at some point after or to coincide with the expiration of a sampling period. The number of Network Frames Received is then obtained from the hardware counter by the Embedded Code software in the protocol analyzer instrument, step 805, for later uploading to the PC.

Similarly, the Network Frame Rate is the number of Network Frames Received, preferably over a one second interval, as monitored by a hardware counter, and is calculated every second.

As the NI module of the protocol analyzer instrument receives frames, these frames are transmitted to the PA module of the protocol analyzer instrument, where the frames are processed. However, if the Network Frame Rate is extremely high, some of the frames may not be sent to the PA module for processing. In this situation, some of the frames will still be counted as having been received by the NI module but may never be processed or analyzed by the PA module, see discussion above regarding Frame Analysis for a discussion of some of the types of frame analyses that may be performed by the PA module.

In addition, the user has the option of choosing to monitor only certain types of frames, see discussion above under Filtering. In this case, all of the frames that are received by the NI module will still be counted as having been received, in order to arrive at the number of Network Frames Received. However, only frames which meet the user-defined parameters are passed to the PA module. Therefore, in these situations when only selected types of frames are passed to the PA module for analysis, the number of frames actually sent to the PA module is different from the value of Network Frames Received.

The number of frames actually sent to the PA module is referred to as the Analyzer Frames Received. The Analyzer Frames Received are calculated by the program shown in FIG. 9. From the Start step 901, the PA module of the protocol analyzer instrument receives a frame from the NI module in step 902. Another hardware counter is incremented accordingly in step 903. The process is repeated until the sampling period has expired, step 904. The final count is then obtained by the Embedded Code from the counter in step 905 for later uploading by the PA module to the PC.

Similarly, Analyzer Frame Rate is the number of Analyzer Frames Received, preferably over a one second interval and is calculated every second. The highest frame rate (in frames per second) detected during the network monitoring session by the analyzer for any single sampling period and the time at which it occurred represents Peak Analyzer Frame Rate and Peak Frame Rate Timestamp. For each sampling period, the current Analyzer Frame Rate is compared to the Peak Analyzer Frame Rate. If the current Analyzer Frame Rate is greater than the Peak Analyzer Frame Rate, the Peak Analyzer Frame Rate is replaced with the current Analyzer Frame Rate. The Peak Frame Rate Timestamp is then replaced with the current time.

The Average Frame Rate is calculated in a manner similar to the Analyzer Frame Rate except that the Average Frame Rate is averaged over the time elapsed during the Network Monitoring Session rather than over one second. Similarly, the Average 32-Second Frame Rate represents the Average Frame Rate over the past 32 seconds as opposed to the entire Network Monitoring Session. Therefore, for the first thirty-one seconds of a monitoring session, there will be no value for Average 32-Second Frame Rate.

After the first thirty-two seconds, the Average 32-Second Frame Rate is recalculated every second on a rolling basis (the frame rate is averaged over the most recent thirty-two second time span).

The embedded code on the protocol analyzer instrument is responsible for calculating the Network Utilization statistic. Network Utilization represents the percentage of the theoretical network bandwidth that is currently being used. For an ethernet network, the theoretical network bandwidth is ten million bits per second (ten megabaud). This is equivalent to 1,250,000 bytes per second (one byte=eight bits).

The embedded code calculates Network Utilization every second in the following manner. First, the total number of bytes represented by the Preamble **208**, Destination **209**, Source **210**, Type **211**, Data **212** and FCS **213** fields of each frame (see FIG. 2) received during the second are summed. To this is added an additional twelve bytes for each frame received during a sampling period to represent quiet time. Quiet time is a 9.6 microsecond interval that follows each frame, during which no data are sent over the line. At a ten megabaud transmission rate, that 9.6 microseconds is equivalent to ninety-six bits or twelve, eight-bit bytes. Therefore, quiet time is the equivalent of twelve byte times.

The embedded code then divides this value by 12,500 (1,250,000 bytes/sec×100%). The resulting percentage statistic is referred to as Network Utilization. Average Utilization is an average of all Utilization values over the duration of the network monitoring session.

Peak Utilization represents the highest percentage of network capacity used during the current session and Peak Utilization Timestamp represents the time at which the peak utilization was detected. For each sampling period, the current Utilization is compared to the Peak Utilization. If the current Utilization is greater than the Peak Utilization, the Peak Utilization is replaced with the current Utilization. The Peak Utilization Timestamp is then replaced with the current time.

Frame Size Distribution is the network statistic that represents the number of frames, classified by size range, that were received by the protocol analyzer instrument since the analyzer was started for the monitoring session (Frame Size Distribution-Cumulative) or during a specified sampling period (Frame Size Distribution-Sample). Frame Size Distribution is calculated through the use of two memory arrays which store information on frame size distribution. One array stores frame size distribution on a cumulative basis and the other array stores frame size distribution on a sampling period basis. There are positions in both arrays corresponding to arbitrary size ranges (e.g. the value for the number of frames detected with lengths between 167 and 255 bytes is stored in position 2 of each array).

The above process is summarized as follows: The frame length of each frame (see Frame Analysis above) is examined. Next, the appropriate memory-array position of the cumulative array is incremented by one to reflect the occurrence of a frame in that particular size range. If the frame was detected during the sampling period, the appropriate array position of the sampling period array is also incremented by one.

As discussed above, under Protocol Analyzer Instrument, the protocol analyzer instrument includes a commercially-obtained LAN chip. The LAN chip is responsible for calculating Broadcast Frames Received, Multicast Frames Received and Network Collisions.

A broadcast frame is a frame sent from one station to all other stations on the network. A broadcast frame's destination address contains an address (referred to as a "broadcast address") that all other stations recognize as being addressed to them. Similarly, a multicast frame is a frame sent to a selected group of stations on a LAN. A multicast frame contains an address (referred to as a "multicast address") that the selected group of stations recognize as being addressed to them. By examining the Destination address field **209** (FIG. 2) of an incoming network frame, the LAN chip can recognize a broadcast address or a multicast address. If the Destination address field **209** contains an address which represents a broadcast or multicast address, the respective counter corresponding to either broadcast frames or multicast frames on the NI module is incremented by one. These counts represent the Broadcast Frames Received and Multicast Frames Received statistics.

Collisions occur when two stations on an Ethernet network stations attempt to transmit frames at the same time, resulting in their transmissions "colliding." Access to an Ethernet network is regulated by a Carrier Sense Multiple Access/Collision Detection (CSMA/CD) contention-based algorithm which is well known to a person having ordinary skill in the art. An Ethernet station listens to the network to determine whether any traffic is present. When the network is clear, it transmits and then listens again to see if the data collides with traffic from any other station. If all is clear, the transmission is complete. If a collision occurs, the station waits a short, random amount of time and retransmits. The LAN chip detects collisions by performing the standard CSMA/CD algorithm when a frame is received. If a collision is detected, a collision counter is incremented by one. This counter is part of the LAN chip.

The present invention also calculates network-wide statistics for errors such as Alignment Errors, Jabbers, Runts, and FCS Errors (see Station Level Statistics above for detailed definitions of these errors). These errors are detected by the LAN chip on the protocol analyzer instrument in a manner similar to that described above.

The network statistics which are unique to other network topologies, such as token ring, FDDI, frame relay, etc., are calculated in a manner which is analogous to the above process.

Information on all network statistics are stored into an array ("network statistics array"). The information stored for each statistic varies depending on the type of statistic, but basically, the value of each statistic is stored into an array along with a timestamp.

When the UI requests information on network statistics, the information stored in the network statistics array is sent to the PC for use by the user interface. This transmission is facilitated through the use of a message. The structure of a network statistics update message is shown in FIG. 12.

The message for network statistics consists of a header **1301** and the contents of the network statistics array **1302**. The header identifies the destination for the message and the type of message (network statistics update message in this case). The contents of the network statistics array are placed in the message in the order in which they were initially placed in the network statistics array.

It will be evident to a person having ordinary skill in the art that network statistics for other network topologies can readily be calculated in a similar manner.

C. Protocol Distribution

The distribution and percentage distribution of the various protocols present in data frames are hereinafter referred to as "protocol distribution". The calculation of protocol distribution is performed by the embedded code executed by the protocol analyzer instrument.

Referring now to FIG. 7, after the start step **701**, the protocol distribution calculation begins with programming step **702** in which the network frame is received. Step **703** next determines the first protocol present in the frame received by the protocol analyzer instrument. For an ethernet frame, this is done by looking at the Type field 211 (see FIG. 2) of the frame. The Type field 211 of an ethernet frame designates the first protocol present in the Data field 212 of the frame. If the Type field 211 contains a value greater than hexadecimal 500, the first protocol present is the Ethernet Version 2 (EthernetV2) protocol, otherwise the first protocol present is the IEEE 802.3 protocol. This first protocol (either the EthernetV2 protocol or IEEE 802.3 protocol) is found in the data portion 212 of the frame.

Next, that protocol (and subsequently all other protocols contained in the frame) is decoded in step 704. The protocol being decoded at any particular time is referred to as the current protocol.

The next step 705 involves storing information for the current protocol. Information on the current protocol is stored in a memory array. An entry in this array is shown in FIG. 5 ("protocol distribution array entry") and contains: the protocol_id 501, statistics_for_the_protocol 502, array_position 503, number_of_children 504, and a children_table 505. Each of these array variables is discussed below. Array information for a protocol is updated whenever that particular protocol is detected in a received frame.

If the current protocol has not previously been detected, a new array entry is created for that protocol. Additionally, if the current protocol is encapsulated within the frame differently than prior occurrences of that protocol, a new array entry is created. The protocol distribution array is stored in the memory of the protocol analyzer instrument and is maintained for the duration of the network monitoring session.

The protocol_id 501 array variable is a programmer defined, arbitrary number used by the embedded code to identify the current protocol. The protocol_id 501 is used to help identify the protocol in the protocol distribution array and has no relationship to the "next layer protocol identification field" defined above. The "next layer protocol identification field" contains a value which is used to identify the protocol directly encapsulated within the current protocol. When this encapsulated protocol is placed into the array, it will be assigned a protocol_id 501 distinct from the value which was in the "next layer protocol identification field" (the value used to initially identify the encapsulated protocol).

The statistics_for_the_protocol 502 includes four entries: (1) the number of frames received (on a cumulative basis for the network monitoring session) which contained the current protocol; (2) the total number of bytes (cumulative basis for the network monitoring session) within those frames (cumulative number of frames) containing the current protocol; (3) the number of frames received (per sampling period, i.e. the sampling time period specified by the user) which contained the current protocol; (4) the total number of bytes (per sampling period) within those frames (sampling period frames). Statistics which are calculated per sampling period are reset at the expiration of the sampling period.

The array_position 503 indicates the position, in the protocol distribution array, where the information on the current protocol is stored.

The number_of children 504 for a particular protocol (the "parent") represents the total number of unique children detected for the parent protocol since the network monitoring session began. A "child" of a parent protocol is any protocol which is encapsulated directly (immediately follows) within the parent.

The children_table 505 is a subarray containing information for all of the children of the parent. This information includes the protocol_id 506 of the child and the array_position 507 of the child.

The next step (step 706) in calculating protocol distribution is to determine if there is another protocol (the "next protocol") encapsulated within the current protocol. If there is a next protocol encapsulated within the data portion 212 of the current protocol or frame, there are two methods used to identify it.

Most protocols contain a "next layer protocol identification field" which is much like the Type field 211 of the Ethernet frame and contains a numerical identification code corresponding to the next protocol present in the frame (i.e., the protocol encapsulated directly within the first protocol). The exact location and contents of the "next layer protocol identification field" within a protocol can vary depending on the standards for that type of protocol. For example, the IEEE 802.3 protocol is defined by the Institute of Electronics and Electrical Engineers' standard 802.3.

Some protocols, however, either do not contain a "next layer protocol identification field" or their "next layer protocol identification field" contains insufficient information for any station other than the ones that are communicating to identify the next protocol. These protocols, including Transport Control Protocol ("TCP") and User Datagram Protocol ("UDP"), are referred to as "conversation-dependent" protocols. Step 707 determines whether the current protocol is conversation-dependent.

If the current protocol is not conversation dependent, it's "next layer protocol identification field" is preferably used in conjunction with a lookup table in step 708A to identify the next protocol present in the frame. This lookup table is stored in the memory of the protocol analyzer instrument. The lookup table maps the value found in the "next layer protocol identification field" to the corresponding protocol which now identifies the protocol encapsulated directly within the data portion 212 of the current protocol.

If, however, it is determined in Step 707 that the current protocol is a conversation-dependent protocol, the unknown next protocol encapsulated within the current protocol is detected in step 708B by comparing bit sequences of the unknown next protocol to known bit patterns from the protocols which can be encapsulated within the current protocol. These known bit patterns can be obtained by referencing the standard defining the protocol. For instance, the UDP protocol is defined by Request for Comments (RFC) number 768, promulgated by the Institute of Electronics and Electrical Engineers. Similarly, the TCP protocol is defined by Request for Comments (RFC) number 793, promulgated by the Institute of Electronics and Electrical Engineers. The known patterns are preferably contained in a lookup table which is used to map known bit patterns to corresponding protocols.

As an example, if the current protocol is UDP, the bits in the unknown next protocol would be compared to bit patterns known to exist in the DHCP (Dynamic Host Configuration Protocol), BootP (Bootstrap Protocol), NetBIOS DGM (Datagram Protocol), RIP (Routing Information Protocol), RWHO (Remote Unix WHO Protocol), TACACS , SNMP (Simple Network Management Protocol) Version 2, and NTP (Network Time Protocol) protocols, all of which can be encapsulated within the UDP protocol.

If a bit sequence in the unknown next protocol sufficiently resembles a bit pattern known to exist in any of these protocols, the protocol corresponding to the known pattern is deemed to be the unknown next protocol and is detected as such.

The above processes are performed iteratively (due to Step 706) for each protocol present in the frame until all protocols present in the frame have been decoded. The entire process is repeated for all frames detected during the sampling period (step 709) or detected during the network monitoring session (step 711), as specified by the user. Upon the expiration of a sampling period, the statistics_for_the_ protocol which are calculated per sampling period are reset in step 710.

After the protocol distribution has been determined by the protocol analyzer instrument and before the step 710 reset operation, the information stored in the protocol distribution array is transmitted to the PC for use by the user interface. This transmission is facilitated through the use of a message. The structure of a protocol distribution update message is shown in FIG. 6.

The message for protocol distribution consists of a header 601 and the contents of the protocol distribution array 602. The header identifies the destination for the message and the type of message (protocol distribution update message in this case). The contents of the protocol distribution array are placed in the message in the order in which they were initially placed in the protocol distribution array.

When the UI requests information on protocol distribution, the message is sent from the PA module of the protocol analyzer instrument to the PC for use by the user interface.

D. Event Information

The embedded code is also capable of detecting and logging "events" in real-time during network monitoring sessions. An "event" occurs when a parameter being monitored on the network exceeds a predefined or user-defined threshold. That user-defined threshold can even be a number of occurrences on the network of some specific phenomenon, during a sampling period. The threshold specifies a value (e.g. number of occurrences, in the case of parameters such as runts, jabbers, etc., or percentage, in the case of a parameter such as Network Utilization) per specified time period and the number of consecutive time periods for which the value must be exceeded to constitute an event. For example, a user can set a threshold, for the parameter runts, of five runts per ten minute sampling period for two consecutive sampling periods. If more than five runts are detected in each of two consecutive sampling periods, the occurrence would be logged as an event.

Ethernet events detected include: High Utilization, High Frame Rate, High Broadcast Rate, High Multicast Rate, Network Collisions, Alignment Errors, FCS Errors, Runts, Jabbers, and Illegal Source Address. Events which are unique to other network topologies, such as FDDI, Token Ring, frame relay, etc., are detected in a manner which is analogous to the process for detecting ethernet events described below.

Utilization, Frame Rate, Broadcast Rate, Multicast Rate, Network Collisions, Alignment Errors, FCS Errors, Runts, and Jabbers are all calculated by the LAN chip on the protocol analyzer instrument. For a discussion of how these rates and errors are calculated or detected, see Station-Level Statistics and Network Statistics above. These rates and errors are flagged as events when they exceed defined thresholds.

An Illegal Source Address error occurs when a frame containing an illegal MAC source address is received. An illegal MAC source address field might contain all binary ones. Such illegal MAC addresses can be caused by a malfunctioning network interface card ("NIC") or NIC driver, they can be artificially produced by some type of traffic generator, or they might be the result of a collision. An Illegal Source Address event occurs when any Illegal Source Address error is detected by the protocol analyzer instrument (i.e., the threshold for this event is usually zero).

Internet Protocol ("IP") events include: Duplicate IP Address, Illegal Source IP Address, and Zeros Broadcast Address. The errors which are the bases of these events only occur if the IP "protocol" is currently in use. The presence, if any, of the IP "protocol" in a frame is detected during the protocol decode process described in detail above in Protocol Distribution. The IP "protocol" contains a field identifying the IP Source Address. This field will be referred to as the "IP Source Address field." The IP "protocol" also contains a field identifying the IP Destination Address. This field will be referred to as the "IP Destination Address field."

A Duplicate IP Address error occurs when two stations try to use the same network IP address. This error is detected by analyzing the IP Source Address field of the IP "protocol." A memory array or other data structure ("IP station list") is used to store information on all detected IP addresses. An array or other data structure ("MAC station list") is used to store information on MAC addresses. These two station lists are cross referenced with each other through the use of linkages and pointers to determine the relationship between every MAC address and every IP address. In other words, every IP address is associated with a MAC address. One MAC address can have several IP addresses but each IP address can correspond to only one MAC address. If two MAC stations in the MAC station list are using the same IP address, a Duplicate IP Address error has occurred. A Duplicate IP Address event occurs when any Duplicate IP Address error is detected by the protocol analyzer instrument (i.e., the threshold for this event is also usually zero).

An Illegal Source IP Address error occurs when the IP Source Address field of the IP "protocol" contains invalid data such as all zeros, a broadcast address, or a multicast address. This error is detected by analyzing the IP Source Address field of the IP "protocol." An Illegal Source IP Address event occurs whenever an Illegal Source IP Address error has been detected by the protocol analyzer instrument (i.e., the threshold for this event is also zero).

A Zeros Broadcast Address error occurs when a sending station has used all zeroes to represent a broadcast address in the portion (IP Destination Address) of the IP "protocol" containing the IP destination address. This error is detected by analyzing IP Destination Address field of the IP "protocol." The IP Destination Address should be all ones when used to designate a broadcast address. A Zeros Broadcast Address event occurs whenever the number of Zeros Broadcast Address errors detected by the protocol analyzer instrument exceeds the defined threshold defined for this event.

ICMP (Internet Control Message Protocol) events include: Host Unreachable, ICMP Redirect, ICMP Parameter Error, Network Unreachable, Port Unreachable, Source Quench, and Time-to-Live Exceeded. The messages which are the bases of these events only occur if the ICMP "protocol" is currently in use. The presence, if any, of the ICMP "protocol" in a frame is detected during the protocol decode process described in detail above in Protocol Distribution.

A Host Unreachable message (a network message not related to a "message" sent by the PA to the UI) is sent by a router to notify the sender of a frame that the router cannot forward that frame to the appropriate destination. A router is a software or hardware connection between two or more networks that enables traffic to be routed from one network to another based upon the intended destinations of the traffic. The appropriate field of the ICMP "protocol" is analyzed to determine whether a Host Unreachable message has been received. A Host Unreachable event occurs when the number of Host Unreachable messages received by the protocol analyzer instrument exceeds the defined threshold for this event.

An ICMP Parameter Error is a message (another network message) indicating that a frame has been discarded due to a problem in its header portion. The appropriate field of the ICMP "protocol" is analyzed to determine whether an ICMP Parameter Error message has been received. An ICMP Parameter Error event occurs when the number of ICMP Parameter Error messages received by the protocol analyzer instrument exceeds the defined threshold for this event.

An ICMP Redirect message (also another network message) occurs when a sending station addresses a frame to a default router because it does not know any other route for that particular destination. If the default router sees that it must transmit the frame out of the same port on which it was received, the router sends the host an ICMP Redirect message advising the sending station of a better router for that destination. The appropriate field of the ICMP "protocol" is analyzed to determine whether an ICMP Redirect message has been received. An ICMP Redirect event occurs when the number of ICMP Redirect messages received by the protocol analyzer instrument exceeds the defined threshold for this event.

A Network Unreachable message (another network message) is sent from a router to the sender of a frame when the router does not have a route or a default route to which to forward the data frame. The appropriate field of the ICMP "protocol" is analyzed to determine whether a Network Unreachable message has been received. A Network Unreachable event occurs when the number of Network Unreachable messages received by the protocol analyzer instrument exceeds the defined threshold for this event.

A Port Unreachable message (another network message) is sent by a destination station to inform the source station that the port indicated by the source station is not currently in use by any process. The appropriate field of the ICMP "protocol" is analyzed to determine whether a Port Unreachable message has been received. A Port Unreachable event occurs when the number of Port Unreachable messages received by the protocol analyzer instrument exceeds the defined threshold for this event.

A Source Quench message (another network message) is sent, by a router or a host, stating that it is receiving so many data frames that its buffers are overflowing. The message is sent back to the source of the excess data frames instructing that source station to slow the flow of data. The appropriate field of the ICMP "protocol" is analyzed to determine whether a Source Quench message has been received. A Source Quench event occurs when the number of Source Quench messages received by the protocol analyzer instrument exceeds the defined threshold for this event.

A Time-to-Live Exceeded message is generated by a router which has received and discarded a transmission which has exceeded its allowable lifetime. Sometimes routing loops form between routers that cause a frame to be forwarded endlessly through the same set of routers over and over. In the IP "protocol," there is a field, called the time-to-live (TTL) field, that limits the lifetime of a frame containing the IP "protocol." The TTL field prevents such an occurrence. When a host generates an IP message (another network message), it gives the TTL field a number value between one and 255. The value is basically equal to the number of routers that can forward the IP message. Each time a router forwards the IP message it reduces the TTL number by one. If a router receives an IP message with a TTL value of one, it decrements the TTL number to zero and discards the message.

The router transmits a Time-to-Live Expired message back to the source to notify the source about the discard. The appropriate field of the ICMP "protocol" is analyzed to determine whether a Time-to-Live Expired message has been received. A Time-to-Live Expired event occurs when the number of Time-to-Live Expired messages received by the protocol analyzer instrument exceeds the defined threshold for this event.

As events are detected, information on the events including event_id, timestamp, byte_length, and parameters are stored in a circular array (event log array). The use of two pointers (one to denote the memory location of the recordation of the last event detected and the other to denote the memory location of the last event sent to the UI) and a circular array allows event updates to be sent to the PC when requested by the UI.

The structure of an entry in this circular array is shown in FIG. 10. "Event_id" **1101** is the variable name used to refer to the programmer-defined id code of the event. Timestamp **1102** is the date and time at which the specific event occurred. Byte_length **1103** is the total number of bytes in the parameter **1104** portion of the array entry. Parameter **1104** contains information on each event. This parameter information is used by the UI portion to construct a detailed event message for display or reporting of the event to the user. For example, if the error was Duplicate IP Address Detected, there would be three parameters, namely the MAC addresses of the two stations using the same IP address as well as the IP address itself.

After events have been detected, information about the events is transmitted to the PC for use by the user interface portion. This transmission is facilitated through the use of a message. The structure of an event update message is shown in FIG. 11. The event update message contains a header **1201** and a portion of the event log array **1202**. The information sent from the event log array is the event_id **1101**, timestamp **1102**, byte_length **1103** and parameters **1104** for the entries in the event log array **1202** since the last update sent to the UI.

VI. User Interface

A. Overview

As discussed above, the user interface is the portion of the software implementation of the present invention that is executed by the PC. At the beginning of a network monitoring session, the user selects which network parameters are to be monitored. Each of these parameters, including station-level statistics, network statistics, event information and protocol distribution is discussed in detail above. The User Interface (UI) is capable of displaying any station-level statistic, network statistic, event information, and protocol distribution (discussed above) which the user requests to see and which the protocol analyzer instrument can capture and report to the UI.

FIG. 13 illustrates the general structure of the user interface (UI). The UI sends request messages to the Embedded Code **302** seeking update information about the various network parameters ("network information").

A message is a preferred method of communication among the various hardware and software components of the present invention. The message contains a header portion which identifies the destination for the message and the type of message (e.g., Network Statistics Request Message, Network Statistics Update Message, etc.). The message also contains the data being transmitted (e.g., the updated network statistics themselves).

The user can select how often network information is updated, i.e. how often the UI requests updates from the embedded code on these parameters. The operation of the UI is largely software controlled using custom software (the design of which is disclosed herein) and also uses off-the-shelf software tools. The custom software is preferably designed using a technique known as "object-oriented programming" which is described in a text entitled *Object Oriented Design with Applications*, by Grady Booch, copyright 1991, from Benjamin Cummings Publishing Co., Inc., Redwood City, Calif., which is incorporated by reference as though fully reproduced herein. Many of the terms used herein, e.g., object, class, scenario diagram, etc., are taken from the Booch text and are well known to programmers familiar with object oriented programming. Another text by Grady Booch, entitled *Object-Oriented Analysis and Design with Applications*, second edition, copyright 1994, is similarly incorporated herein by reference.

The portion of the UI software that is responsible for sending update request messages is referred to as the "Document" **1402** (FIG. **13**). The Document **1402** is the portion of the software that is responsible for managing the flow of data for the UI. After a Request Message is sent to the Embedded Code **302**, the Embedded Code **302** sends the updated network information to an appropriate software "Target" **1401** via an Update Message. The word "target" refers to a software device that is used to accept data for storage, forwarding, or processing.

There is a software Target **1401** for every network parameter that is monitored. In other words, Network Statistics Update Messages are routed to the Network Statistics Target and Station-level Statistics Update Messages are routed to the Station-Level Statistics Target and so on. The Target **1401** is responsible for receiving updated network information, storing the information in a Database **1403** (discussed in detail below) located on the PC's storage device, and providing the Document **1402** with a pointer to the memory location containing the updated network information.

Views **1404** are the portions of the UI software that are responsible for presenting network information, in the form of charts, tables, tree formats, etc., to the user via the PC's display device, e.g., a color cathode ray tube. There is a View **1404** for each network parameter (i.e. network statistics, protocol distribution, etc.) and each type of presentation method (i.e., charts, tables, tree formats etc.). For example, there is a view entitled Network Statistics Chart View, which presents network statistics in a graphical or chart format. A plurality of views can be used at the same time to present network information to the user in several formats simultaneously.

If the user is viewing network information in real-time (i.e., as the information is being uploaded from the protocol analyzer instrument), the Document **1402** informs the appropriate View **1404** of the receipt of some update from the embedded code **302**. The View **1404** then gets from the Document **1402** the pointer to the memory location (in the PC's RAM or random access memory) that contains the updated network information. The View **1404** then presents this information in the appropriate format to the user via the PC's display device.

When the term "real-time" is used in relation to the presentation of network information, the presentation of such information is actually done as updates are received from the embedded code rather than simultaneously with the calculations.

If the user is not viewing "real time" network information but is viewing network information from a database containing network information gathered during a previous network monitoring session (i.e., "baseline data"), the View **1404** gathers relevant information from the Database **1403** and presents the information in the appropriate format to the user via the PC's display device.

Simultaneous display of a plurality of network parameters, either all real time, or all from the database, or mixed is accomplished through the use of well-known features and capability inherent in the Microsoft Windows operating system. Therefore, information on a plurality of network parameters can be displayed simultaneously. Also, nothing has been incorporated into the present invention that limits or disables these well known features and capabilities.

The various Views are programmed to present the network information to the user in the forms of charts, graphs, tables and trees as mentioned above. Off-the-shelf products are used to present the network information to the user.

The product ChartFx (Version 3.0), marketed by Software FX, Inc. at 7100 West Camino Real, Boca Raton, Fla. 33060, is used to display network information in the form of charts and graphs. Network information on station-level statistics, network statistics and protocol distribution in preferably displayed in the form of charts and/or graphs. The User's Manual for ChartFx is hereby incorporated by reference as if fully reproduced herein.

The product SpreadVBX++ (Ver. 2.0), marketed by FarPoint Technologies, Inc. at 133 South Center Court, Suite 1000, Morrisville, N.C. 27560, is used to display network information in the form of tables and spreadsheets. Network information relating to station-level statistics, network statistics and event information is preferably displayed in the form of tables and/or spreadsheets. The User's Manual for SpreadVBX++ is hereby incorporated by reference as if fully reproduced herein.

The product TreeControl (Version 1), marketed by Premia Creative Controls Corp. at 1075 N.W. Murray Blvd., Suite 268, Portland, Oreg. 97229, is used to display protocol distribution in a tree format. The User's Manual for TreeControl is hereby incorporated by reference as if fully reproduced herein.

In creating the User Interface of the present invention, use was made of the Microsoft Foundation Class (MFC) Library, made by Microsoft Corp., i.e., many terms, including "document" and "view," were taken from the literature relating to that library. The User's Manual for the MFC Library is hereby incorporated by reference as if fully reproduced herein.

B. Database

The preferred embodiment of the present invention utilizes an object-oriented (OO) database application. In the preferred embodiment of the invention, the database application used is the POET database product (Ver. 3.0), marketed by Poet Software Corp. at 999 Baker Way, Suite 100, San Mateo, Calif. 94404. The Reference Manual for POET 3.0 is hereby incorporated by reference as if fully reproduced herein.

POET 3.0 is an OO database application which uses a C++ programming language Application Program Interface

(API). Other database applications which use a C++ API would also be appropriate for use in the present invention. The present invention could also utilize ODBC (Open Database Connectivity) database applications if they are used in conjunction with an SQL (Structured Query Language) API.

The primary reason why an object-oriented database as opposed to a standard relational database was selected to implement the present invention is the increased access speed attainable by using an object-oriented database. An object-oriented database such as POET stores C++ objects in a database and allows the programmer to retrieve them using the database operations. The objects read from the database look and act just like the objects stored because an object-oriented database knows how to read C++ class declarations and therefore, manage C++ objects.

In the preferred embodiment of the invention, the database may be saved to a storage device for use as a "baseline" against which future network monitoring sessions may be compared.

C. Station-Level Statistics User Interface

FIG. 14 is a "scenario diagram" depicting the process by which station-level statistics are displayed to the user in real time. First, the Document **1402** requests an update of station-level statistics from the Embedded Code **302**. Second, the Station-Level Statistics Target **1501** receives the updated station-level statistics (i.e. the station-level statistics update message discussed above under Station-Level Statistics).

In step three, the Station-Level Statistics class **1502** is initialized. By "initialized" is meant that a new instance is created of that POET object of the station-level-statistic type, for storing the new data. The Station-Level Statistics class is a class in the POET database which contains information on station-level statistics.

In step four, the Station-Level Statistics Target **1501** decodes the Station-Level Statistics Update Message. The Station-Level Statistics Target **1501** begins this decoding process by reading the message header and the station list array from the update message. Next, it determines which type of station-level statistics are contained in the update message (i.e., ethernet statistics, token ring statistics, FDDI statistics, frame relay statistics, etc.). Finally, the Station-Level Statistics Target **1501** places each of the station-level statistics obtained from the decoded update message in a POET data object for storage and later access.

If the user has selected to store information on station-level statistics to a database on the PC's storage device for later use as a baseline, this information is stored in the appropriate location in the POET database in step five.

In step six, the Station-Level Statistics Target **1501** informs the Document **1402** that the Target **1501** has received some kind of an update. A pointer to the POET data objects containing the updated station-level statistics is sent from the Target **1501** to the Document **1402**. A pointer is an address which identifies or "points to" the memory location in RAM that contains the data.

In steps seven and eight, the Document **1402** informs the Station List View **1503** and the Station Details View **1504** that the views may have to be updated. That is, the Document **1402** informs the Views **1503** and **1504** that some new data has been received but not the exact type and content of the new data. The Station List View **1503** controls the display of a listing of MAC addresses and other information about activity at those MAC address stations. The Station Details View **1504** controls the display of sortings and other detailing of the stations to highlight such factors as which stations are transmitting the most frames, which are receiving the most frames, which are involved with the most error messages, etc. The scope and nature of the details displayed is arbitrary to the user.

In steps nine and ten, the Station List View **1503** and the Station Details View **1504** request verification from the Document **1402** that there is new data which should, in fact, be added to the Station List View **1503** and Details View **1504**. This step is useful because the user has initially selected how often information on station-level statistics was to updated, and it is possible that there was no new station-level statistic information between the last update and the present update. In this case, there are no new data to be added to the views.

If the Document **1402** responds that there are indeed new data, i.e. there is new information in the station-level statistics array, then these new data are obtained by the views in steps eleven and twelve. The Station List View **1503** and the Station Details View **1504** receive a pointer or address to the location in the random-access memory (RAM) of the PC that contains the new data from the Document **1402**. The views then obtain the object containing the new data or information.

Steps thirteen and fourteen involve presenting all of the updated station-level statistics to the user in the form of tables and charts. At this point, the views use the pointers passed to them to gather the new data from its memory location for presentation in the appropriate format. The Station List View **1503** is responsible for displaying the data in the form of a table. This is accomplished through the off-the-shelf product SpreadVBX++, discussed above under User Interface—Overview. The Station List View **1503** is capable of presenting station-level statistics to the user in a sorted order based upon the value of any of the individual statistics. The Station Details View **1504** is responsible for displaying station-level statistics in the form of piecharts indicating top transmitting, receiving, and error producing stations. This is accomplished through use of the off-the-shelf product ChartFx, discussed above under User Interface—Overview.

FIG. **18** illustrates an example of a display screen arrangement for displaying station statistics to the user. The list can show "top talkers" and "top listeners" as well as a host of other catagories of information, the desirability and usefulness of which will be readily evident to a person having ordinary skill in the art of digital network transmission analyzers. A split-screen display is also available with Microsoft Windows to show that the desired statistics can be shown in any number of formats, including the pie chart illustrated in FIG. **18**.

D. Network Statistics User Interface

FIG. **15** is a scenario diagram depicting the process by which information on network statistics is displayed to the user in real time. First the Document **1402** requests an update of network statistics from the Embedded Code **302**. Second, the Network Statistics Target **1601** receives the updated network statistics (i.e. the Network Statistics Update Message discussed above under Network Statistics)..

In step three, the Network Statistics class **1602** is initialized. The Network Statistics class is a class in the POET database which contains information on network statistics. One instance of the Network Statistics class is Ethernet Network Statistics which contains network statistic information particular to an Ethernet network.

In step four, the Network Statistics Target **1601** decodes the Network Statistic Update Message. The Network Statistics Target 1601 begins this decoding process by reading the Header 1301 and the Network Statistics Array 1302 from the update message. Next, the Network Statistics Target 1601 determines which type of network statistics are contained in the update message (i.e., ethernet statistics, token ring statistics, FDDI statistics, frame relay statistics, etc.). Finally, the Network Statistics Target 1601 places each of the network statistics obtained from the decoded update message in a POET data object for storage and later access.

If the user has selected to store information on network statistics to a database on the PC's storage device for later use as a baseline, this information is stored in the appropriate location in the POET database in step five. The appropriate storage location in the POET database is based upon the relevant class (i.e., Ethernet Network Statistics, Token Ring Network Statistics, etc.).

In step six, the Network Statistics Target 1601 informs the Document 1402 that the Target 1601 has received an update. A pointer to the data objects containing the updated network statistics is sent from the Target 1601 to the Document 1402.

In step seven, the Document 1402 informs the Network Statistics Table View 1603 and/or the Network Statistics Chart View 1604 (depending on which view(s) the user is using) that the views may have to be updated.

In step eight, the Network Statistics Table View 1603 and/or Network Statistics Chart View 1604 request verification from the Document 1402 that there is, in fact, new data which should be added to the chart and/or table views. This step is useful because the user has selected how often information on network statistics was to be updated, and it is possible that there was no new network statistic information between the last update and the present update. In this case, there are no new data to be added to the charts and/or tables.

If the Document 1402 responds that there are indeed new data, i.e. there is new information in the network statistics array, then these new data are obtained in step nine. The Network Statistics Table View 1603 and/or the Network Statistics Chart View 1604 receive a pointer to the new data from the Document 1402.

Step ten involves presenting all of the updated network statistics to the user in the form of tables and charts. At this point, the views use the pointers passed to them to gather the new data from their RAM memory location for presentation in the appropriate format. The Network Statistics Table View 1603 is responsible for displaying the data in the form of tables, and this is accomplished through the off-the-shelf product SpreadVBX++, discussed above under User Interface—Overview. The Network Statistics Chart View 1604 is responsible for displaying network statistics in the form of charts and graphs, and this is accomplished through use of the off-the-shelf product ChartFx, discussed above under User Interface—Overview.

FIGS. 19A, 19B and 19C illustrate three examples of display screen display formats useful for showing network statistics to the user. FIG. 19A illustrates how a network utilization chart might look. FIG. 19B illustrates how a network frame rate chart might look, and FIG. 19C illustrates how a frame size distribution chart might look. While charts are shown, a person having ordinary skill in the programming art, and a person having ordinary skill in the digital network transmission art will be the aware that may other other formats of display can readily be substituted for charts.

E. Protocol Distribution User Interface

FIG. 16 is a scenario diagram depicting the process by which cumulative protocol distribution information is displayed to the user in real time. The process by which protocol distribution that is calculated per sampling period is displayed to the user is analogous to this process.

First the Document 1402 requests an update of protocol distribution from the Embedded Code 302. Second, the Protocol Distribution (Cumulative) Target 1701 receives the updated protocol distribution information (i.e. the Protocol Distribution Update Message discussed above under Protocol Distribution).

In step three, the Protocol Distribution class is initialized. The Protocol Distribution class 1702 is a class in the POET database which contains information on protocol distribution. Instances of the Protocol Distribution class include Protocol Distribution (Cumulative), which contains protocol distribution information on a cumulative basis, i.e. since the network monitoring session began, and Protocol Distribution (Sample), which contains information on protocol distribution for the user-defined sampling period.

In step four, the Protocol Distribution (Cumulative) Target 1701 decodes the Protocol Distribution Update Message. The Protocol Distribution (Cumulative) Target 1701 begins this decoding process by reading the Header 601 (FIG. 6) of the "message" and Protocol Distribution Array information or data array or portion 602 of the message, that was taken from the Protocol Distribution Array of the memory of the protocol analyzer instrument.

As discussed above under Protocol Distribution, each entry in the Protocol Distribution Array data 602 portion of the message contains the protocol_id, statistics_for_the_protocol, number_of_children, and a children_table. By iteratively examining the contents of each entry in the Protocol Distribution Array 602 portion of the message and cross-referencing the entry with prior entries, the Protocol Distribution (Cumulative) Target 1701 builds a hierarchical protocol distribution structure (tree structure). If the user has chosen to view protocol distribution in a percentage format, the appropriate statistics_for_the_protocol are converted to a percentage (i.e., the total number of bytes received for the protocol is divided by the total number of bytes received and then multiplied by one hundred). Finally, the Protocol Distribution (Cumulative) Target 1701 places each element of the newly created hierarchical protocol distribution structure in a POET data object for later storage and access.

If the user has selected to store protocol distribution information in a database on the PC's storage device for later use as a baseline, this information is stored in the appropriate location in the POET database (i.e., Protocol Distribution (Cumulative)) in step five.

In step six, the Protocol Distribution (Cumulative) Target 1701 informs the Document 1402 that the Target 1701 has received an update. A pointer to the location, in the PC's RAM memory, of the POET data objects that contain the updated hierarchical protocol distribution structure is sent from the Target 1701 to the Document 1402.

In step seven, the Document 1402 informs the Protocol Distribution Tree View 1703 that the view should perhaps be updated.

In step eight, the Protocol Distribution Tree View 1703 requests verification from the Document 1402 that there is new data which should be added to the tree-type display of protocol distribution. This step is used because the user selected how often information on protocol distribution was updated. It is possible that, while there were some new data received, there may have been no new protocol distribution information contained in the new data received from the time of the last update and the present update. In this case, there are no new data to be added to the tree.

If the Document **1402** responds that there are indeed new protocol distribution data, i.e. there is new information in the protocol distribution array, then these new data are obtained in step nine. The Protocol Distribution Tree View **1703** receives a pointer to the new data (in RAM) from the Document **1402**.

In step ten, the Document **1402** informs the Protocol Distribution Chart View **1704** that the view should be updated. In step eleven, the Protocol Distribution Chart View **1704** receives a pointer to the new data from the Protocol Distribution Tree View **1703**.

Steps twelve and thirteen involve presenting the data to the user in a tree format and a chart format. At this point, the views use the pointers passed to them to gather the new data from its memory location in the RAM of the PC for presentation in the appropriate format. The Protocol Distribution Tree View **1703** is responsible for displaying the data in a tree format. It builds a tree structure based upon the hierarchical protocol distribution structure. An off-the-shelf product entitled TreeControl (discussed above under User Interface—Overview) is used to display the tree structure. The Protocol Distribution Chart View **1704** is responsible for displaying protocol distribution in a pie-chart format. This is accomplished through use of the off-the-shelf product ChartFx, discussed above under User Interface—Overview.

FIG. **20** illustrates how a split-screen display can be used to highlight one ISO protocol layer, instantly revealing usage by the protocols detected on the network.

F. Event Information User Interface

FIG. **17** is a scenario diagram depicting the process by which event information is displayed to the user in real time. First the Document **1402** requests an update of event information from the Embedded Code **302**. Second, the Event Target **1801** receives the updated event information (i.e. the event update message discussed above under Event Information).

In step three, the Event Log database class **1802** is initialized. The Event Log class is a class in the POET database which contains event information.

In step four, the Event Target **1801** decodes the Event Update Message. The Event Target **1801** begins this decoding process by reading the message header and the portion of the event log array. It then places information relating to each event, as contained in the portion of the event log array of the memory of the protocol analyzer instrument into a POET data object in RAM storage of the PC for later storage and access.

If the user has selected to store event information in a database on the PC's storage device for later use as a baseline, the information is stored in the appropriate location in the POET database in step five.

In step six, the Event Target **1801** informs the Document **1402** that the Target **1801** has received an update. A pointer to the POET data objects containing the updated event information is sent from the Target **1801** to the Document **1402**.

In step seven, the Document **1402** informs the Event Log View **1803** that the view should perhaps be updated. In step eight, the Event Log View **1803** requests verification from the Document **1402** that there is, in fact, new data which should be added to the Event Log View. This step is useful because the user had selected how often event information was updated, and it is possible that there was no new event information from the time of the last update to the time of the present update. In this case, there are no new data to be added to the view.

If the Document **1402** responds that there is indeed new data, i.e. there is new information in the event log array in the memory of the PC, then these new data are obtained by the Event Log View **1803** in step nine. The Event Log View **1803** receives from the Document **1402**, a pointer to the new data now stored in the PC's RAM.

Step ten involves presenting all of the updated event information to the user in a table format. At this point, the event log view uses the pointer passed to it to gather the new event information from its memory location in the PC's RAM for presentation in the appropriate format. For each event, the Event Log View **1803** presents the name of the event (derived from the event_id), the time of the event, and a brief description of the event (derived from the event parameters) in the form of a table. Presentation of event information in a table format is accomplished through the off-the-shelf product SpreadVBX++, discussed above under User Interface—Overview.

FIG. **21** illustrates a preferred example of how detected events can be sorted and displayed with timestamps, on the PC's display device so as to enhance the user's ability to find information quickly.

G. Hypertext Troubleshooting Information

The user interface is also capable of displaying detailed information about a particular event and the possible causes of the event in a hypertext format. Hypertext in conjunction with the present invention allows a user to access detailed explanations through use of the PC's pointing device. The user can obtain detailed definitions of statistics and events as well as possible causes of each type of event by double-clicking the PC's pointing device on the event or statistic displayed by the user interface. A "window" is opened on a display containing a detailed definition of the event or statistic as well as a brief discussion of the possible causes and ramifications of the event. This information is contained in a standard Microsoft Windows context-sensitive help file format.

In addition to specific information relating to events and statistics, the user interface is also capable of displaying step-by-step troubleshooting information in a hypertext format which assists the user in solving problems on a network by posing increasingly specific queries until a solution is reached. This information is likewise contained in a standard Microsoft Windows help file format. The exact method by which the above data are displayed would be readily apparent to a person having ordinary skill in the art of software programming and in the art of network analyzing. The text of the troubleshooting information can be created and written specifically for the UI by a person having ordinary skill in the digital data transmission art; or troubleshooting information for Ethernet and Token Ring networks can be examined in a textbook entitled *Ethernet and Token Ring Optimization*, by Daniel J. Nassar, Copyright 1996, Henry Holt & Co., Inc., New York, N.Y., which is hereby incorporated by reference as though fully reproduced herein.

H. Reports

As discussed above, transmission information concerning Protocol Distribution, Station-Level Statistics, Network Statistics and Events are displayed for the user in the form of graphs and charts. The transmission information can also be used to create customized presentation-quality reports. These customized reports provide the transmission information in a presentation quality format. The invention also allows the user to specify the time span which the report will cover.

Reports may be printed on a standard printer connected to the PC or displayed on the PC's display device. Reports also may be previewed and modified on-line prior to printing. The reporting feature is implemented using an off-the-shelf reporting application entitled ReportFX (Ver. 1.0), marketed by Software FX, Inc. at 7100 West Camino Real, Boca Raton, Fla. 33060. The User's Manual for ReportFx is hereby incorporated by reference as if fully reproduced herein.

I. Analysis of Captured Frames

The present invention is also capable of saving the contents of the capture buffer to a capture file on a storage device (see Frame Analysis above for discussion of capture buffer). The present invention can then display information about specific frames stored in the capture file. The user interface allows the user to examine a captured frame, search the capture file for filter criteria, view the protocols present in a frame, specific frames, view only those frames which meet specific and print the contents of the frame on a printer attached to the PC. Analysis of captured frames is accomplished by use of a software application entitled Examine which is marketed by Wandel & Goltermann Technologies, Inc. at 1030 Swabia Court, Research Triangle Park, N.C. 27709-3585.

J. Use of Analysis

If an event is noted, depending upon the nature of the event noted, the data portion of the message conveying that event information to the PC will include any MAC addresses that were involved in that event. The user can request reporting or displaying of any combination of further information about that MAC address that might be pertinent to that event.

For example, if a high level of network utilization is noted, the transmitting stations and receiving stations can be displayed as sorted according to the number of messages transmitted or received. This will immediately disclose which stations are transmitting the most (top talkers)and which stations are receiving the most (top listeners).

The station statistics for the top talker or top listener can then be queried by protocol used. If a large number of the frames for a station carry the IP (internet protocol), it could mean that the employee using that station is either gathering a lot of needed project information from the internet or that the employee is "surfing" the internet on company time. Therefore, some supervisory involvement may be in order to ascertain if that employee should be switched to a more lightly-loaded network or should be admonished about wasting company time.

VII. Conclusion

While the protocol analyzer herein described constitutes the preferred embodiment of the present invention, it is to be understood that the invention is not limited to this precise form of apparatus and that changes may be made therein without departing from the scope of the invention which is defined in the appended claims.

We claim:

1. A protocol analyzer for calculating and displaying protocol distribution in real-time in connection with monitoring data frames carried on a digital transmission network, comprising:

means for monitoring the transmission, over the digital transmission network, of frames containing data and protocols;

means for identifying the protocols within the frames and for identifying the encapsulation of the protocols within the frames:

means for storing the identity and encapsulation of the protocols within the frames:

means for calculating the protocol distribution of the frames; and

means for displaying in real-time the protocol distribution of the frames in a hierarchical tree format based upon the encapsulation of the protocols within the frames, said means comprising:

a first processing instrumentality, comprising means for periodically requesting a protocol distribution update message containing encoded updated protocol distribution information from said means for calculating the protocol distribution of the frames;

a second processing instrumentality, having:

a. means for receiving and decoding the protocol distribution update message to obtain the updated protocol distribution information, and

b. means for providing said first processing instrumentality with a pointer to the updated protocol distribution information;

a display device for displaying the updated protocol distribution information; and

a third processing instrumentality, having:

a. means for obtaining a pointer to the updated protocol distribution information from said first processing instrumentality, and

b. means for sending the updated protocol distribution information to said display device in a hierarchical tree format based upon the encapsulation of the protocols within the frames.

* * * * *

#6
11/25/03

Our Ref./Docket No: APPT-001-3

Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz, *et al.*

Application No.: 09/608126

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.

TRANSMITTAL: RESPONSE TO OFFICE ACTION

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

Transmitted herewith is a response to an office action for the above referenced application. Included with the response are:

_____ drawing(s);

This application has:

_____ a small entity status. If a claim for such status has not earlier been made, consider this as a claim for small entity status.

_____ No additional fee is required.

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this correspondence is being facsimile transmitted to the U.S. Patent and Trademark Office at 703-872-9306 addressed to Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: 3 Nov 03

Signed:

Name: Dov Rosenfeld, Reg. No. 38687

Application No.: 09/608126 Page 2

____ Applicant(s) believe(s) that no Extension of Time is required. However, this conditional petition is being made to provide for the possibility that applicant has inadvertently overlooked the need for a petition for an extension of time.

_X_ Applicant(s) hereby petition(s) for an Extension of Time under 37 CFR 1.136(a) of:

_X_ one months ($110) ____ two months ($410)

____ three months ($930) ____ four months ($1450)

If an additional extension of time is required, please consider this as a petition therefor.

_X_ A credit card payment form for the required fee(s) is attached.

_X_ The Commissioner is hereby authorized to charge payment of the following fees associated with this communication or credit any overpayment to Deposit Account No. 50-0292 (A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED):

_X_ Any missing filing fees required under 37 CFR 1.16 for presentation of additional claims.

_X_ Any missing extension or petition fees required under 37 CFR 1.17.

Respectfully Submitted,

3 Nov 03
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

Our Ref./Docket No: APPT-001-3 Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz, *et al.*

Application No.: 09/608126

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.

TRANSMITTAL: RESPONSE TO OFFICE ACTION

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

Transmitted herewith is a response to an office action for the above referenced application. Included with the response are:

_____ drawing(s);

This application has:

_____ a small entity status. If a claim for such status has not earlier been made, consider this as a claim for small entity status.

_____ No additional fee is required.

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this correspondence is being facsimile transmitted to the U.S. Patent and Trademark Office at 703-872-9306 addressed to Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: 3 Nov 03

Signed:

Name: Dov Rosenfeld, Reg. No. 38687

Application No.: 09/608126 Page 2

_____ Applicant(s) believe(s) that no Extension of Time is required. However, this conditional petition is being made to provide for the possibility that applicant has inadvertently overlooked the need for a petition for an extension of time.

_X_ Applicant(s) hereby petition(s) for an Extension of Time under 37 CFR 1.136(a) of:

_X_ one months ($110) _____ two months ($410)

_____ three months ($930) _____ four months ($1450)

If an additional extension of time is required, please consider this as a petition therefor.

_X_ A credit card payment form for the required fee(s) is attached.

_X_ The Commissioner is hereby authorized to charge payment of the following fees associated with this communication or credit any overpayment to Deposit Account No. 50-0292 (A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED):

_X_ Any missing filing fees required under 37 CFR 1.16 for presentation of additional claims.

_X_ Any missing extension or petition fees required under 37 CFR 1.17.

Respectfully Submitted,

3 Nov 2003
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

INVENTEK Fax

Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618, USA
Phone: (510)547-3378; Fax: (510)653-7992
dov@inventek.com

Patent Application Ser. No.: 09/608126	Ref./Docket No: APPT-001-3
Applicant(s): Dietz, *et al.*	***Examiner.:*** Vu, Thong H.
Filing Date: June 30, 2000	***Art Unit:*** 2142

FAX COVER PAGE

TO: Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

United States Patent and Trademark Office
(Examiner Vu, Thong H., Art Unit 2142)

Fax No.: 703-872-9306

DATE: November 03, 2003

FROM: Dov Rosenfeld, Reg. No. 38687

RE: Response to Office Action

Number of pages including cover: 20

OFFICIAL COMMUNICATION

PLEASE URGENTLY DELIVER A COPY OF THIS RESPONSE TO EXAMINER VU, THONG H., ART UNIT 2142

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: Nov 3, 2003

Signed:
Name: Dov Rosenfeld, Reg. No. 38687

TRANSMITTAL FORM

(to be used for all correspondence after initial filing)

Application Number	09/608126
Filing Date	30 Jun 2000
First Named Inventor	Dietz, Russell S.
Group Art Unit	2142
Examiner Name	Vu, Thong H.
Attorney Docket Number	APPT-001-3

ENCLOSURES *(check all that apply)*

☐ Fee Transmittal Form

☒ Fee Attached (extension of time)

☒ Amendment / Response

☐ ☐ After Final

☐ ☐ Affidavits/declaration(s)

☒ Extension of Time Request

☐ Express Abandonment Request

☐ Information Disclosure Statement

☐ Certified Copy of Priority Document(s)

☐ Response to Missing Parts/ Incomplete Application

☐ ☐ Response to Missing Parts under 37 CFR 1.52 or 1.53

☐ Assignment Papers *(for an Application)*

☐ Drawing(s)

☐ Licensing-related Papers

☐ Petition Routing Slip (PTO/SB/69) and Accompanying Petition

☐ To Convert a Provisional Application

☐ Power of Attorney, Revocation Change of Correspondence Address

☐ Terminal Disclaimer

☐ Small Entity Statement

☐ Request of Refund

Remarks

☐ After Allowance Communication to Group

☐ Appeal Communication to Board of Appeals and Interferences

☐ Appeal Communication to Group *(Appeal Notice, Brief, Reply Brief)*

☐ Proprietary Information

☐ Status Letter

☐ Additional Enclosure(s) *(please identify below):*

☐ Return Postcard

☐

☐

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT/ CORRESPONDENCE ADDRESS

Firm *or* Individual name	Dov Rosenfeld, Reg. No. 38687
Signature	
Date	November 3, 2003

ADDRESS FOR CORRESPONDENCE

Firm *or* Individual name	Dov Rosenfeld 5507 College Avenue, Suite 2 Oakland, CA 94618, Tel: +1-510-547-3378

CERTIFICATE OF FACSIMILE TRANSMISSION

I hereby certify that this correspondence is being facsimile transmitted with the United States Patent and Trademark Office at Telephone number 703-746-7239 addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on this date: November 3, 2003

Type or printed name	Dov Rosenfeld, Reg. No. 38687		
Signature		Date	November 3, 2003

#7
B. Jon
11/5/03

Our Ref./Docket No: APPT-001-3

Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz, *et al.*	Group Art Unit: 2142
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	

RESPONSE TO OFFICE ACTION UNDER 37 CFR 1.111

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

This is a response to the Office Action of July 10, 2003.

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***.

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this correspondence is being facsimile transmitted to the U.S. Patent and Trademark Office at 703-872-9306 addressed to Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: 3 Nov 2003

Signed:

Name: Dov Rosenfeld, Reg. No. 38687

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.

A1 Sub B1

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 ~~comprising none~~ that may contain one or more flow-entries for previously encountered conversational flows, the looking up to determine if the received packet is of an existing flow;

~~(d)~~(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

~~(e)~~(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 though the connection point is received by the packet acquisition device.

2. (Original) A method according to claim 1, further including:

extracting identifying portions from the packet,

wherein the looking up uses a function of the identifying portions.

A1

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.

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. (Currently amended) A method according to claim 1, wherein ~~step (d)~~ 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 (e)~~ 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.

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;

A1

(b) a memory for storing a database ~~comprising none~~ that may contain one or more flow-entries for previously encountered conversational flows to which a received packet may belong; 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.

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.

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.

REMARKS

Status of the Application:

Claims 1–21 are the claims of record of the application. Claims 1–21 have been rejected.

Amendment to the Claims:

Applicants have amended the claims to overcome misnumbering and other informality pointed out by the examiner, and to further bring out the inventive aspects over the cited prior art.

Claim Objections

In paragraph 2 of the office action, paragraphs in claims 1 and 10 were objected to as being misnumbered.

This amendment corrects the misnumbering. Applicants thank the examiner for pointing this typographical error out. Examiner was correct in the interpretation.

In paragraph 3 of the office action, the expression "none or more" was objected to.

This amendment changed "none or more" to "one or more" and added that the database may contain one or more entries. The use of "none or more" is common in the art, and regularly appears in specifications, e.g., those put out by the IEEE and W3 consortium. In this case, the use of "none or more" covers the case, for example, that there are not yet any entries in the database. However, because the examiner objected, alternate language was found.

Claim Rejections -35 USC § 102

In paragraph 4 of the office action, claims 1–21 have been rejected under 35 USC 102(e) as being anticipated by U.S. patent 5,850,388 to Anderson et al., hereinafter "Anderson."

Why Anderson does not anticipate Applicant's invention

While aspects of the present invention, like Anderson, provides statistics, the present invention differs from Anderson in several ways. Applicant will argue that Anderson does not anticipate the present invention as follows:

1) The present invention analyzes and compiles statistics about conversational flows; Anderson does not distinguish flows, but rather gathers station-level statistics, and network protocol statistics.

2) The present invention includes looking up each and every packet to see if it belongs to a previously encountered flow; Anderson only provides for looking up a database after analysis as a separate process that looks at statistics gathered: the station-level statistics, or the protocol statistics.

3) An aspect of the present invention includes, for any packet ascertained to belong to an existing flow by looking up the database, identifying the state of the flow, and carrying out any state operations defined that that state; Anderson has no concept of state of the flow, or even of a conversational flow, so that no such state operations are therefore carried out.

4) Anderson provides for filtering the packets prior to analysis; the present invention analyzes each and every packet.

There are many other aspects of the present claims that are not anticipated by Anderson.

Description of Anderson

Anderson describes a protocol analyzer that is capable of displaying station level statistics, network statistics, real-time event information, and protocol distribution.

The operation of Anderson is summarized By FIGS. 3, 4, and 7. FIG. 3 is a diagram illustrating the flow of data, analysis, and control in Anderson, FIG. 4 is a flowchart illustrating the process by which statistics for individual stations on a network (station-level statistics) are calculated, and FIG. 7 is a flowchart illustrating the method by which protocol distribution is calculated.

FIG. 3 is a diagram illustrating the flow of data, analysis, and control. As shown in FIG. 3, Anderson includes a protocol analyzer instrument 304 that carries out station level analysis (see FIG. 4) and protocol analysis (see FIGS. 5, 6, and 7). FIG. 4 is a flowchart illustrating the process by which statistics for individual stations on a network (station-level statistics) are calculated, while FIG. 7 is a flowchart illustrating the method by which protocol distribution is calculated and stored in the data structure shown in FIG. 5 for each protocol recognized. The station level analysis of FIG. 4 is described starting col. 10, line 43, while protocol distribution analysis of FIG. 7 is described starting col. 16, line 64.

The analysis is carried out for a sampling time, e.g., the network monitoring session.

The results of the statistical analysis are sent to a user interface in a PC operated by the user. FIG. 6 illustrates the data structure used to send the protocol distribution to the user interface in the PC.

The user interface includes looking up a database 310. It is the results of the statistical analysis that are looked up to produce various reports for the user.

Description of the present invention

FIG. 3 shows the operation of the analyzer of the present invention. This is also described in more detail in related and incorporated by reference U.S. Patent application 09/608237 for *METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK*, to inventors Dietz, et al, Attorney/Agent Docket APPT-001-1. The present invention adds statistical analysis to U.S. Patent application 09/608237.

FIG. 3 describes a network monitor that includes carrying out protocol specific operations on *every* individual packet that passes through a connection point on a network. The operations include extracting information from header fields in the packet to use for building a signature for identifying the ***conversational flow*** of the packet and for recognizing future packets as belonging to a ***previously encountered flow***. A parser subsystem includes a parser for recognizing different patterns in the packet that identify the protocols used. For each protocol recognized, an extractor extracts important packet elements from the packet. These form a ***signature (i.e., key)*** for the packet. The extractor also preferably generates a hash for rapidly identifying a flow that may have this signature from a database of known flows.

For ***each packet,*** the flow signature of each packet, the hash and at least some of the payload are passed to an analyzer subsystem.

The analyzer subsystem receives parts of each packet from the parser subsystem, and, ***for each packet,*** looks up a database of flow records for previously encountered conversational flows to determine whether a signature is from an existing flow. The analyzer further ***identifies the state of the existing flow***, and performs any ***state processing operations*** specified for the state. In the case of a newly encountered flow, the analyzer includes a flow insertion and deletion engine for inserting new flows into the database of flows. Any state operations that are specified for the new flow are also carried out.

Statistics are maintained for each conversational flow.

The present invention analyzes and compiles statistics about conversational flows; Anderson does not distinguish flows, but rather gathers station-level statistics, and network protocol statistics.

The present invention includes a process that recognizes ***a conversational flow*** and then generates statistics for the conversational flow. Anderson does not recognize a conversational flow, but instead compiles statistics for particular stations, and/or for particular network protocols used. A conversational flow is not identified simply by the stations that are involved in a communication, but rather by the nature of the communication, e.g., the application program being invoked. Thus, even for the same two stations, the present invention identifies different conversational flows between two stations and compiles statistics on ***each*** conversational flow.

It is important to be able to distinguish between the term "***connection flow***" commonly used to describe all the packets involved with a single connection, and a ***conersational flow*** as used in the present invention. A conversational flow is the sequence of packets that are exchanged in any direction as a result of an activity—for instance, the running of an application on a server as requested by a client. Unlike Anderson, ***the present invention is able to identify and classify conversational flows rather than only connection flows,*** including gathering statistics on the flows. The reason for this is that some conversational flows involve more than one connection, and some even involve more than one exchange of packets between a client and server. Thus, there may be different ***states*** to a flow. This is particularly true when using client/server protocols such as RPC, DCOMP, and SAP, which enable a service to be set up or defined prior to any use of that service.

An example of such a case is the SAP (Service Advertising Protocol), a NetWare (Novell Systems, Provo, Utah) protocol used to identify the services and addresses of servers attached to a network. In the initial exchange, a client might send a SAP request to a server for print service. The server would then send a SAP reply that identifies a particular address—for example, SAP#5—as the print service on that server. Such responses might be used to update a table in a router, for instance, known as a Server Information Table. A client who has inadvertently seen this reply or who has access to the table (via the router that has the Service Information Table) would know that SAP#5 for this particular server is a print service. Therefore, in order to print data on the server, such a client would not need to make a request for a print service, but would simply send data to be printed specifying SAP#5. Like the previous exchange, the transmission of data to be printed also involves an exchange between a client and a server, but requires a second connection and is therefore independent of the initial exchange. In order to eliminate the possibility of disjointed conversational exchanges, it is desirable for a network packet monitor to be able to "virtually concatenate"—that is, to link—the first exchange with the second. If the clients were the same, the two packet exchanges would then be correctly identified as being part of the same conversational flow.

Other protocols that may lead to disjointed flows, include RPC (Remote Procedure Call); DCOM (Distributed Component Object Model), formerly called Network OLE (Microsoft Corporation, Redmond, Washington); and CORBA (Common Object Request Broker Architecture). RPC is a programming interface from Sun Microsystems (Palo Alto, California) that allows one program to use the services of another program in a remote machine. DCOM, Microsoft's counterpart to CORBA, defines the remote procedure call that allows those objects—objects are self-contained software modules—to be run remotely over the network. And CORBA, a standard from the Object Management Group (OMG) for communicating between distributed objects, provides a way to execute programs (objects) written in different programming languages running on different platforms regardless of where they reside in a network.

The present invention includes looking up each and every packet to see if it belongs to a previously encountered flow; Anderson only provides for looking up a database after analysis as a separate process that looks at statistics gathered: the station-level statistics, or the protocol statistics.

Each and every packet has its signature extracted. A database that includes any previously encountered flow is looked up to ascertain if the present packet belongs to an existing flow. Anderson does have a database, but it is a database in the PC (304) for use by the user interface. The only information passed to the PC by Anderson's analysis program is station level statistics or protocol distributions obtained during sampling periods. Individual packets or parts thereof are not passed on to the user interface, so each and every packet is not looked up.

An aspect of the present invention includes, for any packet ascertained to belong to an existing flow by looking up the database, identifying the state of the flow, and carrying out any state operations defined that that state;

Anderson has no concept of state of the flow, or even of a conversational flow, so that no such state operations are therefore carried out.

As described above, each conversational flow may have several states before reaching a "steady" state. At any state in the flow, according to an aspect of the invention, there may be some state-specific operations that need to be carried out to continue the identification process. Anderson does not include carrying out state-specific processing, or even the concept of conversational flows.

There is no way the method described by Anderson can carry out the "virtually concatenation" described above. Anderson does not include the concept of the state of a flow.

Anderson provides for filtering the packets prior to analysis; the present invention analyzes each and every packet.

Anderson includes a filter that provides for optionally selects only certain type of frames for analysis. See the paragraph starting col. 10, line 20. Therefore, not only does Anderson not look up its database (Anderson's 310) for each and every packet because only statistics rather than individual packets are passed on to the database, but also not even every packet is subject to the analysis of analyzer 304.

Note that the analysis carried out by Applicant's analyzer (Applicant's FIG. 3) identifies flows, and compiles statistics on the recognized flows. Therefore no would be required. Filters have disadvantages as described in incorporated by reference U.S. Patent application 09/608237.

Examiner's 102(e) rejection of claim 1 over Anderson.

The amendments

In order to further bring out the difference between Anderson and Johnson, the Applicants have amended claim 1 to explicitly state that the packet acquisition device is coupled to the connection point, and that the lookup is carried out for every packet received from the packet acquisition device.

Anderson's database 1403 vs. the flow entry database. Anderson does not look up that database 1403 for each packet

In paragraph 5 of the office action, the Examiner asserts that, in respect to claim 1, Applicant's flow entry database is Anderson's database 1403. This database is part of Anderson's user interface described in Section IV of Anderson starting col. 22, line 48 through to the end of the description (col. 31). As stated in col. 9, lines 36-40, the database is used ***selectively*** to store the results of the analysis that are performed by the protocol analyzer instrument.

Step (b) of claim 1, as amended, include for each received packet, looking up a flow-entry database that may contain one or more flow-entries for previously encountered conversational flows, the looking up to determine if the received packet is of an existing flow. Anderson does not do any looking up of database 1403 for each received packet. In fact, as shown in FIG. 3, Anderson's database is part of the under interface 303 which is quite removed from Anderson's protocol analyzer 304. Anderson's database received statistical reports, either station level reports or protocol distribution reports, so cannot look up information for each received packet.

Anderson's "previous session" is not a previously encountered conversational flow.

Furthermore, Applicant's lookup is to determine if a packet is part of an existing conversational flow. In paragraph 5, the examiner erroneously asserts that the conversational flow is Anderson's "previous network monitoring session" as in col. 24, lines 6-13. This part of Anderson states:

> *If the user is not viewing "real time" network information but is viewing network information from a database containing network information gathered during a previous network monitoring session (i.e., "baseline data"), the View 1404 gathers relevant information from the Database 1403 and presents the information in the appropriate format to the user via the PC's display device.*

Anderson's session as used here is a ***network monitoring session*** during which data is collected, and not a conversational flow. Anderson clearly defines the network monitoring session as "the period of time during which a network is being analyzed."

Anderson's "prior entries" are not the same as previously encountered conversational flows.

The examiner also asserts that the "prior entries" mentioned in Anderson col. 28, lines 26–43 are the same as Applicants previously encountered conversational flows in the flow-entry database.

It has already been stated that the flow entry database is not the same as Anderson's database 1403.

The prior entries are in the protocol distribution array 602 part of the message shown in FIG. 6 that is used to send the results of protocol analysis from Anderson's analyzer to the PC by use by the user interface.

FIGS. 5, 6, and 7 described the protocol analysis part of Anderson's analyzer. See, for example, Section C. starting col. 16, line 59 for a description of how the protocol analyzer works. Anderson's protocol analysis method tries to recognize every protocol in a frame, one frame at a time. A memory array is set of up for each protocol that is encountered during a sampling period, e.g., during the network monitoring session. Such an array is shown in FIG. 5. Array information is updated each time a protocol is encountered in a received frame. The array keeps track of statistics for protocol during the session.

As states in col. 19, starting line 1, the steps of the analysis are performed iteratively for each protocol present in a received frame until all protocols in the frame have been decoded. The entire process is repeated for all frames detected during the network monitoring session, after which the statistics are reset.

The results of the analysis of all protocols encountered is then sent to the PC for use by the user interface using a protocol distribution update message, as shown in FIG. 6.

Thus, the "prior entries" mentioned in Anderson col. 28, lines 26–43 are prior entries of previously previous distribution update messages that have been collected. Again, this is very different from Applicant's looking up the flow database for previously encountered conversational flows.

There is in fact no concept of a conversational flow in Anderson.

The examiner further asserts that "every packet passing through the connection point is received by the packet acquisition device" is described by Anderson in col. 8, line 26–col. 9, line 13. As already discussed, Anderson provides a filter, so teaches away from even dealing with each packet.

In summary, the examiner has failed to show that claim 1 (as amended) is anticipated by Anderson. Claim 1, as amended, is allowable and action to that end is respectfully requested.

Rejection of dependent claims 2–16 over Anderson.

Claim 1 is now allowable. Therefore, 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, including claims 2–16, are allowable.

Rejection of independent claim 10 over Anderson.

In paragraph 15 of the office action, the Examiner asserts that Anderson described 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. For this assertion, the examiner cites "between the last update and the present update" in col. 26, lines 6-40. Col. 26 describes actions that occur in the user interface, not in the analyzer, and described how previous collected data is analyzed. This is not carried out for each packet. There is now concept of conversational flow or state of the flow in Anderson, and Anderson cannot carry out state operations on the packet.

Thus, even if the examiner remains unconvinced by the arguments with respect to the independent claim 1, the examiner has failed to show that claim 10 is anticipated by Anderson, and claim 10 would still be allowable.

Rejection of dependent claims 11–16 over Anderson.

These claims are all dependent on claim 10. Thus, even if the examiner remains unconvinced by the arguments with respect to the independent claim 1, because claim 10 is allowable, the examiner's arguments with respect to claims 11–16 are moot. The Applicants are not making any admission to such arguments being correct, only that they are moot.

Rejection of independent claim 17 over Anderson.

In paragraph 6 of the office action, the Examiner asserts in that claim 17 contains the similar limitations set forth of method claim 1, and that therefore claim 17 is rejected for the similar rationale set forth in claim 1.

As argued above, the examiner has failed to show that the features of claim 1 (as amended) are anticipated by Anderson.

Claim 17 has been amended to include that the looking up is for every received packet. As argued above, claim 17 would be allowable because Anderson does not include several of its features.

Claim 17 is therefore allowable, and action to that end is respectfully requested.

Rejection of the dependent claims 18–21 over Anderson.

Claim 17 is now allowable. Therefore, 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, including claims 18–21, are 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.

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,

3 Nov 2003
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

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

7590 12/23/2003

Dov Rosenfeld
Suite 2
5507 College Avenue
Oakland, CA 94618

EXAMINER	
VU, THONG H	
ART UNIT	PAPER NUMBER
2142	8

DATE MAILED: 12/23/2003

Please find below and/or attached an Office communication concerning this application or proceeding.

PTO-90C (Rev. 10/03)

Office Action Summary	**Application No.** 09/608,126	**Applicant(s)** DIETZ ET AL.	
	Examiner Thong H Vu	**Art Unit** 2142	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
- Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *03 November 2003*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-21* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-21* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. §§ 119 and 120

12)☐ 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:

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)).

* See the attached detailed Office action for a list of the certified copies not received.

13)☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application) since a specific reference was included in the first sentence of the specification or in an Application Data Sheet. 37 CFR 1.78.

a) ☐ The translation of the foreign language provisional application has been received.

14)☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121 since a specific reference was included in the first sentence of the specification or in an Application Data Sheet. 37 CFR 1.78.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☐ Information Disclosure Statement(s) (PTO-1449) Paper No(s) _____ .

4) ☐ Interview Summary (PTO-413) Paper No(s). _____ .

5) ☐ Notice of Informal Patent Application (PTO-152)

6) ☐ Other: .

1. Claims 1-21 are pending.

2. Applicant is required to update the copending applications on pages 1-2.

3. Claims 1 and 17 are amended. Therefore, the Final rejection is appropriate.

Response to Arguments

4. In response to applicant's argument that the references fail to show certain features of applicant's invention, it is noted that the features upon which applicant relies (i.e., re-use information from data transaction) are not recited in the rejected claim(s). Although the claims are interpreted in light of the specification, limitations from the specification are not read into the claims. See *In re Van Geuns*, 988 F.2d 1181, 26 USPQ2d 1057 (Fed. Cir. 1993).

5. Applicant's arguments fail to comply with 37 CFR 1.111(b) because they amount to a general allegation that the claims define a patentable invention without specifically pointing out how the language of the claims patentably distinguishes them from the references.

6. Applicant's arguments do not comply with 37 CFR 1.111(c) because they do not clearly point out the patentable novelty which he or she thinks the claims present in view of the state of the art disclosed by the references cited or the objections made. Further, they do not show how the amendments avoid such references or objections.

7. Applicant's arguments, see pages 6-12, filed 11/03/03, with respect to the rejection(s)of claim(s) 1 and 17 under Anderson reference have been fully considered but they are not persuasive. Applicant argues the prior art does not teach:

1. analyzing a conversional flows;

2. looking up each and every packet to see if it belongs to a previously flow;

3. identifying the flow, carrying an operation which defines the state;

4. analyzing each and every packet.

Examiner notes the prior art taught :

(1) analyzing a conversional flows [Anderson, a protocol analyzer monitoring real time event information over the Ethernet which was well-known in the art as full duplex (i.e.: two way communication network), col 16 lines 10-25];

(2) looking up each and every packet [Anderson, each frame, col 11 lines 5-17,col 13 lines 52-67] to see if it belongs to a previously flow [Anderson, determines whether the entry corresponding to the source address of the frame in the station list array, col 11 lines 57-67];

(3) identifying the flow (i.e.: event ID), carrying a operation which defines the state [Anderson, construct a detailed event message for reporting of the event to the user, col 22 lines 25-36];

(4) analyzing each and every packet [Anderson, determines whether the entry corresponding to the source address of the frame in the station list array, col 11 lines 57-67].

Thus, the rejection is sustained.

Claim Rejections - 35 USC § 112

8. 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).

Double Patenting

9. The nonstatutory double patenting rejection is based on a judicially created doctrine grounded in public policy (a policy reflected in the statute) so as to prevent the unjustified or improper timewise extension of the "right to exclude" granted by a patent and to prevent possible harassment by multiple assignees. See *In re Goodman*, 11 F.3d 1046, 29 USPQ2d 2010 (Fed. Cir. 1993); *In re Longi*, 759 F.2d 887, 225 USPQ 645 (Fed. Cir. 1985); *In re Van Ornum*, 686 F.2d 937, 214 USPQ 761 (CCPA 1982); *In re Vogel*, 422 F.2d 438, 164 USPQ 619 (CCPA 1970);and, *In re Thorington,* 418 F.2d 528, 163 USPQ 644 (CCPA 1969).

A timely filed terminal disclaimer in compliance with 37 CFR 1.321(c) may be used to overcome an actual or provisional rejection based on a nonstatutory double patenting ground provided the conflicting application or patent is shown to be commonly owned with this application. See 37 CFR 1.130(b).

Effective January 1, 1994, a registered attorney or agent of record may sign a terminal disclaimer. A terminal disclaimer signed by the assignee must fully comply with 37 CFR 3.73(b).

Claims 1-21 are rejected under the judicially created doctrine of double patenting over claims 1-10 of U. S. Patent No. 6,651,099 B1 since the claims, if allowed, would improperly extend the "right to exclude" already granted in the patent.

The subject matter claimed in the instant application is fully disclosed in the patent and is covered by the patent since the patent and the application are claiming common subject matter, as follows:

(Application): A method of analyzing a flow of packets passing through a connection point (protocol analyzer) on a computer network, *(Patent '099):A packet monitor for examining packets passing through a connection point on a computer network in real-time, the packets provided to the packet monitor via a packet acquisition device connected to the connection point:*

(a) receiving a packet from a packet acquisition device coupled to the connection point;

(a) a packet-buffer memory configured to accept a packet from the packet acquisition device;

(b) for each received packet, looking up a flow-entry database that may contain one or more flow-entries for previously encountered conversational flows, the looking up to determine if the received packet is of an existing flow;

(b) a parsing/extraction operations memory configured to store a database of parsing/extraction operations that includes information describing how to determine at least one of the protocols used in a packet from data in the packet; (c) a parser subsystem coupled to the packet buffer and to the pattern/extraction operations memory, the parser subsystem configured to examine the packet accepted by the buffer, extract selected portions of the accepted packet, and form a function of the selected portions sufficient to identify that the accepted packet is part of a conversational flow-sequence; (d) a memory storing a flow-entry database including a plurality of flow-entries for conversational flows encountered by the monitor;
(e) a lookup engine connected to the parser subsystem and to the flow-entry database, and configured to determine using at least some of the selected portions of the accepted packet if there is an entry in the flow-entry database for the conversational flow sequence of the accepted packet;

(c) if the packet is of an existing flow, identifying the last encountered state of the flow, performing any state operation 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 *(h) a state processor coupled to the flow-entry database, the protocol/state identification engine, and to the state patterns/operations memory, the state processor, configured to carry out any state operations specified in the state patterns/operations memory for the protocol and state of the flow of the packet, the carrying out of the state operations furthering the process of identifying which application program is associated with the conversational flow-sequence of the packet, the state processor progressing through a series of states and state operations until there are no more state operations to perform for the accepted packet, in which case the state processor updates the flow-entry, or until a final state is reached that indicates that no more analysis of the flow is required, in which case the result of the analysis is announce the protocol and state of the conversational flow of the packet;*

(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 though the connection point is received by the packet acquisition device.

(f) a state patterns/operations memory configured to store a set of predefined state transition patterns and state operations such that traversing a particular transition pattern as a result of a particular conversational flow-sequence of packets (i.e.: new flow entry) indicates that the particular conversational flow-sequence is associated with the operation of a particular application program, visiting each state in a traversal including carrying out none or more predefined state operations;
(g) a protocol/state identification mechanism coupled to the state patterns/operations memory and to the lookup engine, the protocol/state identification engine configured to determine the protocol and state of the conversational flow of the packet;

Furthermore, there is no apparent reason why applicant was prevented from presenting claims corresponding to those of the instant application during prosecution of the application which matured into a patent. See *In re Schneller*, 397 F.2d 350, 158 USPQ 210 (CCPA 1968). See also MPEP § 804.

Claim Rejections - 35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

10. Claims 1-21 are rejected under 35 U.S.C. § 102(e) as being anticipated by Anderson et al [Anderson 5,850,388].

11. As per claim 1, Anderson discloses a method of analyzing a flow of packets (or frames) passing through a connection point (protocol analyzer) on a computer network [col 4 line 49-col 6 line 19], the method comprising:

(a) receiving a packet from a packet acquisition device [protocol analyzer, col 8 line 26-col 9 line 13];

(b) looking up a flow-entry database [database, col 5 lines 24-46, col 9 lines 30-40, col 23 lines 35-45, col 24 lines 6-20,57-col 25 line 50; lookup table, col 18 lines 29-37] comprising one or more flow-entries for previously encountered conversational flows, the looking up to determine if the received packet is of an existing flow [previous session, col 24 lines 6-13; prior entries, col 28 lines 26-43];

(c) if the packet is of an existing flow, updating the flow-entry of the existing flow including storing one or more statistical measures kept in the flow-entry [col 17 lines 15-23, col 25 lines 22-47, col 27 lines 24-34, col 28 lines 49-67]; and

(d) if the packet is of a new flow, storing a new flow-entry for the new flow in the flow-entry database [update new information, col 27 lines 10-53], including storing one or more statistical measures kept in the flow-entry [statistics, col 27 lines 10-34], wherein every packet passing though the connection point is received by the packet acquisition device [protocol analyzer col 8 line 26-col 9 line 13].

12. Claim 17 contains the similar limitations set forth of method claim 1. Therefore, claim 17 is rejected for the similar rationale set forth in claim 1.

13. As per claim 2, Anderson discloses extracting identifying portions from the packet, wherein the looking up uses a function of the identifying portions [information is extracted from a frame, col 9 line 42-col 10 line 18].

14. As per claim 3, Anderson discloses the steps are carried out in real time on each packet passing through the connection point [col 4 line 58-col 5 line 46].

15. As per claim 4, Anderson discloses the one or more statistical measure includes selected from the set of consisting of the total packet count for the flow, the time and a differential time from the last entered time to the present time [col 28 lines 58-67].

16. As per claim 5, Anderson discloses including one or more metrics (parameters) related to the flow of a flow entry from one or more of the statistical measure in the flow entry [col 10 lines 20-40, col 19 lines 30-45,col 22 lines 16-65].

17. As per claim 6, Anderson discloses the metrics include one or more quality of service (QOS) metrics (id, time, length, col 22 lines 16-23].

18. As per claim 7, Anderson discloses 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 [Anderson, the last updated, col 29 lines 60-67].

19. As per claim 8, Anderson discloses calculating one or more quality of service (QOS) metrics from the base metrics [col 14 lines 39-60, col15 lines 32-46,col 17 lines 45-57].

20. As per claim 9, Anderson discloses 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 [col 28 lines 58-67].

21. As per claim 10, Anderson discloses

(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 [between the last update and the present update, col 26 lines 6-40];

(d) includes if the packet is of a new flow, performing any state operations required for the initial state of the new flow [new data and user initial select how often information on station statistics was to update, col 26 lines 6-15].

22. As per claim 11, Anderson discloses 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 [col 30 line 58-col 31 line 10].

23. As per claim 12, Anderson discloses 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 [col 30 line 58-col 31 line 10].

24. As per claim 13, Anderson discloses reporting is part of the state operations for the state of the flow [col 30 line 58-col 31 line 10].

25. As per claim 14, Anderson discloses updating the flow-entry, including storing identifying information for future packets to be identified with the flow-entry [col 16 lines 47-54, col19 lines 17-24, col 22 line 66-col 23 line 6] .

26. As per claim 15, Anderson discloses receiving further packets, wherein the state processing of each received packet of a flow furthers the identifying of the application program of the flow as inherent of new data received [col 28 lines 58-67].

27. As per claim 16, Anderson discloses 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 as inherent feature of parameters [col 22 lines 16-65].

28. As per claim 20, Anderson discloses 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 [software performs statistical calculations ,col 7 lines 33-53].

29. As per claim 21, Anderson discloses the statistical processor determines and reports the one or more metrics from time to time [col 30 line 58-col 31 line 10].

Claim Rejections - 35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

30. Claims 1-21 are rejected under 35 U.S.C. § 102(e) as being anticipated by Chapman et al [Chapman 6,330,226 B1].

40. As per claim 1, Chapman discloses a method of analyzing a flow of packets passing through a connection point on a computer network [Chapman, a method of monitoring traffic flows through a traffic admission control apparatus, see abstract, col 3 lines 5-27, Fig 1]

(a) receiving a packet from a packet acquisition device coupled to the connection point [Chapman, the admission control detects the packets, col 3 lines 28-44];

(b) for each received packet, looking up a flow-entry database (i.e.: history table) that may contain one or more flow-entries for previously encountered conversational flows (i.e.: two-way traffic or interactive environment), the looking up to determine if the received packet is of an existing flow [Chapman, matching to a predefined pattern, col 3 lines 28-44];

(c) if the packet is of an existing flow, identifying the last encountered state of the flow (i.e.: most recent update flow), performing any state operation 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 [Chapman, flow ID is compared and updated, col 3 lines 50-60, col 5 lines 1-7]; and

(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 though the connection point is received by the packet acquisition device [Chapman, new entry is made or stored into database, col 3 lines 50-60, col 4 lines 40-55].

41. Claim 17 contains the similar limitations set forth of method claim 1. Therefore, claim 17 is rejected for the similar rationale set forth in claim 1.

42. As per claim 2, Chapman discloses extracting identifying portions from the packet, wherein the looking up uses a function of the identifying portions [Chapman, detecting the packets, col 3 lines 28-44].

43. As per claim 3, Chapman discloses the steps are carried out in real time on each packet passing through the connection point [Chapman, interactive user, col col 5 lines 20-30].

44. As per claim 4, Chapman discloses the one or more statistical measure includes selected from the set of consisting of the total packet count for the flow, the time and a differential time from the last entered time to the present time as inherent feature of history table record or database [Chapman, col 6 lines 35-45].

45. As per claim 5, Chapman discloses including one or more metrics related to the flow of a flow entry from one or more of the statistical measure in the flow entry [Chapman, flow characteristic, col 4 lines 40-55].

46. As per claim 6, Chapman discloses the metrics include one or more quality of service (QOS) metrics [Chapman, flow characteristic, col 4 lines 40-55].

47. As per claim 7, Chapman discloses 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 [Chapman, regular time intervals, col 4 lines 27-37].

48. As per claim 8, Chapman discloses calculating one or more quality of service (QOS) metrics from the base metrics [Chapman, computing the packet loss characteristic, col 5 lines 33-57].

49. As per claim 9, Chapman discloses 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 [Chapman, adjust its windows to fit the bandwidth, col 4 lines 15-25].

50. As per claim 10, Chapman discloses (c) 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 [Chapman, detect problem condition, col 4 lines 25-37];

(d) if the packet is of a new flow, performing any state operations required for the initial state of the new flow [Chapman, a new entry, col 3 lines 50-60].

51. As per claim 11, Chapman discloses 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 [Chapman, database, col 4 lines 40-55].

52. As per claim 12, Chapman discloses 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 [Chapman, the most recent update, col 5 lines 1-7].

53. As per claim 13, Chapman discloses reporting is part of the state operations for the state of the flow [Chapman, a sample history, col 4 lines 25-30].

54. As per claim 14, Chapman discloses updating the flow-entry, including storing identifying information for future packets to be identified with the flow-entry [Chapman, the history is updated, col 3 lines 50-60].

55. As per claim 15, Chapman discloses receiving further packets, wherein the state processing of each received packet of a flow furthers the identifying of the application program of the flow as inherent of new entry received.

56. As per claim 16, Chapman discloses 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 [Chapman, detect problem condition, col 4 lines 25-37].

57. As per claim 20, Chapman discloses 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 [Chapman, database, col 4 lines 40-55].

58. As per claim 21, Chapman discloses the statistical processor determines and reports the one or more metrics from time to time [Chapman, database, col 4 lines 40-55].

Claim Rejections - 35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

59. Claims 1-21 are rejected under 35 U.S.C. § 102(e) as being anticipated by Bullard [6,625,657 B1].

60. As per claim 1, Bullard discloses a method of analyzing a flow of packets passing through a connection point on a computer network [Bullard, a method for tracking network record, abstract]

(a) receiving a packet from a packet acquisition device coupled to the connection point [Bullard, monitoring each packet of a network flow, col 28 line 45-col 29 line 5];

(b) for each received packet, looking up a flow-entry database [Bullard, database col 10 lines 7-17; flow description, col 12 lines 52-62; flow descriptors, col 14 lines 17-58] that may contain one or more flow-entries for previously encountered conversational flows (i.e.: bi-directional flow) [col 7 lines 18-25] the looking up to determine if the received packet is of an existing flow [Bullard, matching to older record, col 16 lines 16-36];

(c) if the packet is of an existing flow, identifying the last encountered state of the flow, performing any state operation specified for the state of the flow [Bullard, flow status descriptors, col 14 lines 17-58], and updating the flow-entry of the existing flow [Bullard, updating the record, col 8 lines 20-67] including storing one or more statistical measures kept in the flow-entry [Bullard, statistical phenomenon, col 31 lines 7-40]; and

(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 though the connection point is received by the packet acquisition device [Bullard, new NAR, col 16 lines 16-36; new IP packet, col 26 lines 28-46].

61. Claim 17 contains the similar limitations set forth of method claim 1. Therefore, claim 17 is rejected for the similar rationale set forth in claim 1.

62. As per claim 2, Bullard discloses extracting identifying portions from the packet, wherein the looking up uses a function of the identifying portions [Bullard, retrieving identified data, col 34 lines 45-63].

63. As per claim 3, Bullard discloses the steps are carried out in real time on each packet passing through the connection point as inherent feature of Internet provider.

64. As per claim 4, Bullard discloses the one or more statistical measure includes selected from the set of consisting of the total packet count for the flow, the time and a differential time from the last entered time to the present time [Bullard, time periods T1,T2, col 19 line 42-col 20 line 24].

65. As per claim 5, Bullard discloses including one or more metrics related to the flow of a flow entry from one or more of the statistical measure in the flow entry [Bullard, quality of service identifiers, col 14 lines 45-50].

66. As per claim 6, Bullard discloses the metrics include one or more quality of service (QOS) metrics [Bullard, quality of service identifiers, col 14 lines 45-50].

67. As per claim 7, Bullard discloses 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 [Bullard, accounting time interval, col 14 lines 45-50].

68. As per claim 8, Bullard discloses calculating one or more quality of service (QOS) metrics from the base metrics [Bullard, audit the classes in quality of service, col 33 lines 17-27].

69. As per claim 9, Bullard discloses 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 [Bullard, combined value over a time period, col 11 lines 32-38].

70. As per claim 10, Bullard discloses (c) 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 [Bullard, col 16 lines 16-61];

(d) if the packet is of a new flow, performing any state operations required for the initial state of the new flow [Bullard, col 16 lines 16-61].

71. As per claim 11, Bullard discloses 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 [Bullard, statistical probability, col 31 lines 7-40].

72. As per claim 12 Bullard discloses 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 [Bullard, report has been generated by a time condition, col 27 lines 55-67].

73. As per claim 13, Bullard discloses reporting is part of the state operations for the state of the flow [Bullard event reporting, col 28 lines 12-25].

74. As per claim 14, Bullard discloses updating the flow-entry, including storing identifying information for future packets to be identified with the flow-entry [Bullard, stored ID, col 23 lines 10-25].

75. As per claim 15, Bullard discloses receiving further packets, wherein the state processing of each received packet of a flow furthers the identifying of the application program of the flow as inherent of new data.

76. As per claim 16, Bullard discloses 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 [Bullard flow probe correlates the state information, col 24 lines 55-67].

77. As per claim 20, Bullard discloses 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 [Bullard, statistical probability, col 31 lines 7-40].

78. As per claim 21, Bullard discloses the statistical processor determines and reports the one or more metrics from time to time [Bullard, statistical probability, col 31 lines 7-40].

Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL.** See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to examiner Thong Vu, whose telephone number is (703)-305-4643.

The examiner can normally be reached on Monday-Thursday from 8:00AM- 4:30PM.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, *Jack Harvey*, can be reached at *(703) 305-9705*.

Any inquiry of a general nature or relating to the status of this application should be directed to the Group receptionist whose telephone number is (703) 305-9700.

Any response to this action should be mailed to: Commissioner of Patent and Trademarks, Washington, D.C. 20231 or faxed to :

After Final	(703) 746-7238
Official:	(703) 746-7239
Non-Official	(703) 746-7240

Hand-delivered responses should be brought to Crystal Park 11,2121 Crystal Drive, Arlington. VA., Sixth Floor (Receptionist).

Thong Vu
Patent Examiner
Art Unit 2142

JACK B. HARVEY
SUPERVISORY PATENT EXAMINER

Notice of References Cited	Application/Control No. 09/608,126	Applicant(s)/Patent Under Reexamination DIETZ ET AL.	
	Examiner Thong H Vu	Art Unit 2142	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-6,625,657 B1	09-2003	Bullard, William Carter Carroll	709/237
	B	US-6,330,226 B1	12-2001	Chapman et al.	370/232
	C	US-6,651,099 B1	11-2003	Dietz et al.	709/224
	D	US-6,424,624 B1	07-2002	Galand et al.	370/231
	E	US-6,279,113 B1	08-2001	Vaidya, Vimal	713/201
	F	US-6,363,056 B1	03-2002	Beigi et al.	370/252
	G	US-6,115,393 A	09-2000	Engel et al.	370/469
	H	US-4,972,453	11-1990	Daniel et al.	379/9.03
	I	US-5,535,338 A	07-1996	Krause et al.	709/222
	J	US-5,802,054	09-1998	Bellenger, Donald M.	370/401
	K	US-5,720,032	02-1998	Picazo, Jr et al	709/250
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	NOV94: Packet Filtering in the SNMP Remote Monitor ; www.skrymir.com/dobbs/articles/1994/9411/9411h/9411h.htm
	V	GTrace -- A Graphical Traceroute Tool" authored by Ram Periakaruppan, Evi Nemeth ; http://www.caida.org/outreach/papers/1999/GTrace/index.xml
	W	
	X	

*A copy of this reference is not being furnished with this Office action (See MPEP § 707 05(a))
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign

US006625657B1

(12) **United States Patent**
Bullard

(10) **Patent No.: US 6,625,657 B1**
(45) **Date of Patent: Sep. 23, 2003**

(54) **SYSTEM FOR REQUESTING MISSING NETWORK ACCOUNTING RECORDS IF THERE IS A BREAK IN SEQUENCE NUMBERS WHILE THE RECORDS ARE TRANSMITTING FROM A SOURCE DEVICE**

(75) Inventor: **William Carter Carroll Bullard**, New York, NY (US)

(73) Assignee: **Nortel Networks Limited**, St. Laurent (CA)

(*) 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.: **09/276,423**

(22) Filed: **Mar. 25, 1999** ✓

(51) **Int. Cl.**^{7} **G06F 15/16**

(52) **U.S. Cl.** **709/237**; 709/248; 709/224

(58) **Field of Search** 709/216, 217, 709/218, 219, 237, 248, 224; 705/40, 35; 707/201; 379/130

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,109,486 A		4/1992	Seymour	395/200
5,230,048 A		7/1993	Moy	395/600
5,465,206 A		11/1995	Hilt et al.	364/406
5,557,746 A		9/1996	Chen et al.	395/200.06
5,668,955 A		9/1997	deCiutiis et al.	379/130
5,757,798 A		5/1998	Hamaguchi	370/397
5,761,502 A		6/1998	Jacobs	395/614
5,778,350 A		7/1998	Adams et al.	707/1
5,781,729 A		7/1998	Baker et al.	395/200.6
5,784,443 A		7/1998	Chapman et al.	379/119
5,793,853 A		8/1998	Sbisa	379/120
5,794,221 A		8/1998	Egendorf	705/40
5,799,321 A	*	8/1998	Benson	707/201
5,802,502 A		9/1998	Gell et al.	705/37
5,815,556 A		9/1998	Thuresson et al.	379/93.25
5,852,812 A		12/1998	Reeder	705/39

(List continued on next page.)

OTHER PUBLICATIONS

XACCT Usage Overview, XACCT Technologies, 1997.

HP and Cisco Deliver Internet Usage Platform and Billing and Analysis Solutions (http://www.hp.com/smartinternet/press/prapr28.html), Hewlett Packard Company, 1998.

Article, Quadri, et al., *Internet Usage Platform White Paper* (http://www.hp.com.smartinternet/press/not.html), Hewlett Packard Company.

Article, *Strategies for Managing IP Data* (http://www.hp.com/smartinternet/press/not.html), Hewlett Packard Company.

Article, Nattkemper, *HP and Cisco Deliver Internet Usage and Billing Solutions* (http:www.interex.org/hpworldnews/hpW806.html), Hewlett Packard Company, Jun. 1, 1999)(?).

HP Smart Internet Billing Solution (http://hpcc925.external.hp.com/smartinternet/solutions/usagebilling.html), HEwlett Packard Company, 1998.

HP Smart Internet Usage Analysis Solution (http://www.hp.com/smartinternet/solutions/usageanalysis.html), Hewlett Packard Company, 1998.

Press Release, *New Cisco IOS NetFlow Software and Utilities Boost Service Provider Revenues and Service Management Capabilities* (http://www.cisco.com/warp/public/cc/cisco/mkt/gen/pr.archive/cros pr.htm), Cisco Systems, Inc., Jul. 1, 1997.

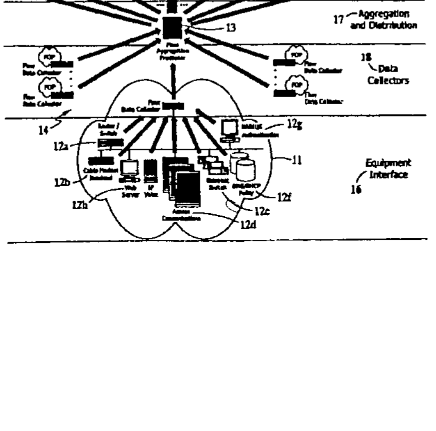
(List continued on next page.)

Primary Examiner—Le Hien Luu
(74) *Attorney, Agent, or Firm*—Withrow & Terranova, PLLC

(57) **ABSTRACT**

A system for collecting and aggregating data from network entities for a data consuming application is described. The system includes a data collector layer to receive network flow information from the network entities and to produce records based on the information. The system also includes a flow aggregation layer fed from the data collection layer and coupled to a storage device. The flow aggregation layer receiving records produced by the data collector layer and aggregates received records. The system can also include an equipment interface layer coupled to the data collector layer and a distribution layer to obtain selected information stored in the storage device and to distribute the select information to a requesting, data consuming application.

6 Claims, 36 Drawing Sheets



U.S. PATENT DOCUMENTS

5,878,420	A		3/1999	de la Salle	707/10
5,920,847	A		7/1999	Kolling et al.	705/40
5,926,104	A		7/1999	Robinson	340/825.22
5,949,782	A		9/1999	Wells	370/395
5,956,391	A		9/1999	Melen et al.	379/114
5,956,690	A		9/1999	Haggerson et al.	705/3
5,958,009	A		9/1999	Friedrich et al.	709/224
5,958,010	A		9/1999	Agarwal et al.	709/224
5,978,780	A		11/1999	Watson	705/40
5,991,746	A	*	11/1999	Wang	705/40
5,999,604	A		12/1999	Walter	379/133
6,002,948	A		12/1999	Renko et al.	455/567
6,009,154	A		12/1999	Rieken et al.	379/114
6,014,691	A		1/2000	Brewer et al.	709/217
6,032,147	A		2/2000	Williams et al.	707/101
6,038,551	A		3/2000	Barlow et al.	705/41
6,047,051	A	*	4/2000	Ginzboorg et al.	379/130
6,047,268	A	*	4/2000	Bartoli et al.	705/35
6,058,380	A		5/2000	Anderson et al.	705/40
6,069,941	A		5/2000	Byrd et al.	379/121
6,078,907	A		6/2000	Lamm	705/40
6,088,706	A		7/2000	Hild	707/202
6,112,236	A		8/2000	Dollin et al.	709/224
6,118,936	A		9/2000	Lauer et al.	395/200.53
6,119,160	A		9/2000	Zhang et al.	709/224
6,151,601	A		11/2000	Papierniak et al.	707/10
6,157,648	A		12/2000	Voit et al.	370/401
6,175,867	B1		1/2001	Taghadoss	709/223
6,199,195	B1		3/2001	Goodwin et al.	717/1
6,230,203	B1		5/2001	Koperda et al.	709/229
6,243,667	B1		6/2001	Kerr et al.	703/27
6,272,126	B1		8/2001	Strauss et al.	370/352
6,282,267	B1		8/2001	Nolting	379/34
6,308,148	B1		10/2001	Bruins et al.	703/27
6,327,049	B1		12/2001	Ohtsuka	358/1.18
6,359,976	B1		3/2002	Kalyanpur et al.	379/134
6,377,567	B1		4/2002	Leonard	379/352
6,381,306	B1		4/2002	Lawson et al.	379/32.01
6,385,301	B1		5/2002	Nolting et al.	379/32.01
6,418,467	B1		7/2002	Schweitzer et al.	709/223

OTHER PUBLICATIONS

Documentation, *NetFlow FlowCollector 2.0* (http://www-.cisco.com/univerca/cc/td/doc/product/rtrmgmt/nfc/nfc 2 0/index.htm), Cisco Systems, Inc. 1988.

Article, Mary Jander, *Hot Products: Network Management–Usage Monitoring Intranet Inspector* (http://www.data.com/issue/99017/manage3.html), Tech Web, CMP Media Inc., Jan. 1999.

Article, Loring Wirbel, *Tools coming from probing, billing of IP packets* (http://www.techweb.com/se/directlink.cgi?EET19981214S0033), EETIMES, Issue 1039, Section: Systems & Software, Dec. 14, 1998, CMP Media Inc.

Article, Loring Wirbel, *Tools enable usage–based billing on the Net* (http:www.eetimes.com/story/OEG19981208S0007), EETIMES, Dec. 8, 1998, CMP Media Inc.

Article, Limor Schweizer, *Meeting th IP Network Billing Challenge* (http://www.tmcnet.com/tmcnet/articles/xacct1298.htm), TMCnet, Dec. 1998(?).

Product Review, *XACCTusage* (http://www.xacct.com/news/xuoverview.htm), internet.com, Internet Product Watch (?).

Article, Loring Wirbel, *In next–generation networks, ISPs are calling the shots* (http://www.techweb.com/se/directlink.cgi?EET19981019S0058), EETIMES, Issue 1031, Section: Communications, Oct. 19, 1998, CMP Media Inc.

Article, Kate Gerwig, *ISPs Take 'Do–It–Yourself' Tack With Billing* (http://www.internetwk.com/news1098/news101398–3htm), CMP's Tech Web, CMP Media Inc., Oct. 12, 1998.

Article, Kathleen Cholewka, *Xacct Makes It Easier To Bill For IP Service* (http://www.zdnet.com/intweek/daily/981005d.html), Inter@ctive Week, ZDNet, Oct. 5, 1998.

Article, Lucas et al., *Mediation in a Multi–Service IP Netwrk* (http://www.xacct.com/news/art092898.html), XACCT Technologies, Inc. Oct. 1, 1998.

Article, Matt Hamblen, *Middleware enables net usage planning* (http:www.xacct.com/news/art092898.html), Computerworld, vol. 32, No. 29, Sep. 28, 1998.

Article, Tim Greene, *XAACT pinpoints IP network usage* (http://www.xacct.com/news/art092198b.html), Network-World, vol. 15, No. 38, Sep. 21, 1998.

Article, Margie Semilof, *Charging for IP use gets easier* (http://www.xacct.com/news/art092198b.html), Computer Reseller News, Sep. 21, 1998.

Article, Kate Gerwig, *Creating Meter Readers For The Internet* (http://www.techweb.com/se/directlink.cgi?INW19980921S0042), Internetweek, Issue: 733, Section: Bandwidth, Sep. 21, 1998, CMP Media Inc.

Article, John Morency, *XaCCT offers multivendor, multi–technology billing model for ISP consumption* (http://www.nwfusion.com/newsletters/nsm/0914nm2.html), Network World Fusion Focus on Netwrok/Systems Management, Network World Inc., Sep. 16, 1998.

Article, Shira Levine, *XACCT brings flexible billing to the IP world* (http://www.americasnetwork.com/news/9809/980915 xacct.html), Advanstar Communications, Sep. 15, 1998.

Telecom Product News, *Accounting and Reporting System for Corporate Network resource and IS Use* (http://www.x-acct.com/news/pressreleases/papers3.html), XACCT Technologies, Inc., Apr. 1998.

Article, *BYTE's Best of Show Award at CeBit Goes To SuperNova;s Visual Concepts* (http://www.byte.com/special/3cebit98.htm), Byte, Mar 23, 1998.

Press Release, *XACCT Teams With Kenan to Provide Advanced Solution For Usage–Based Billing of IP Applications* (http://www.xacct.com/news/pressreleases/papers8.html), XACCT Technologies, Inc. Oct. 26, 1998.

Press Release, *Select and XACCT Partner to Provide IP Usage–based Billing Solution* (http://www.xacct.com/news/pressreleases/papers7.html), XACCT Technologies Inc., Sep. 28, 1998.

Press Release, *XACCT Supports Cisco's New Web–Based Enterprise Management Suite* (http://www.xacct.com/news/pressreleases/papers6.html), XACCT Technologies, Inc., Sep. 22, 1998.

Press Release, *XACCT Technologies Enables Usage–Based Billing for Internet*)http://www.xacct.com/news/pressreleases/papers4.html), XACCT Technologies, Inc., Sep. 21, 1998.

News Release, *XACCT Technologies Now shipping Accounting and Reporting System for Corporate network resource and ISP Use* (http://www.xacct.com/news/pressreleases/papers1.html), XACCT Technologies, Inc., Mar. 19, 1998.

News Release, *XACCT Technologies Releases Accounting and Reporting System for ISP and Corporate Network Resource Use* (http://www.xacct.com/news/pressreleases/papers.html), XACCT Technology, Inc., Dec. 10, 1997.

* cited by examiner

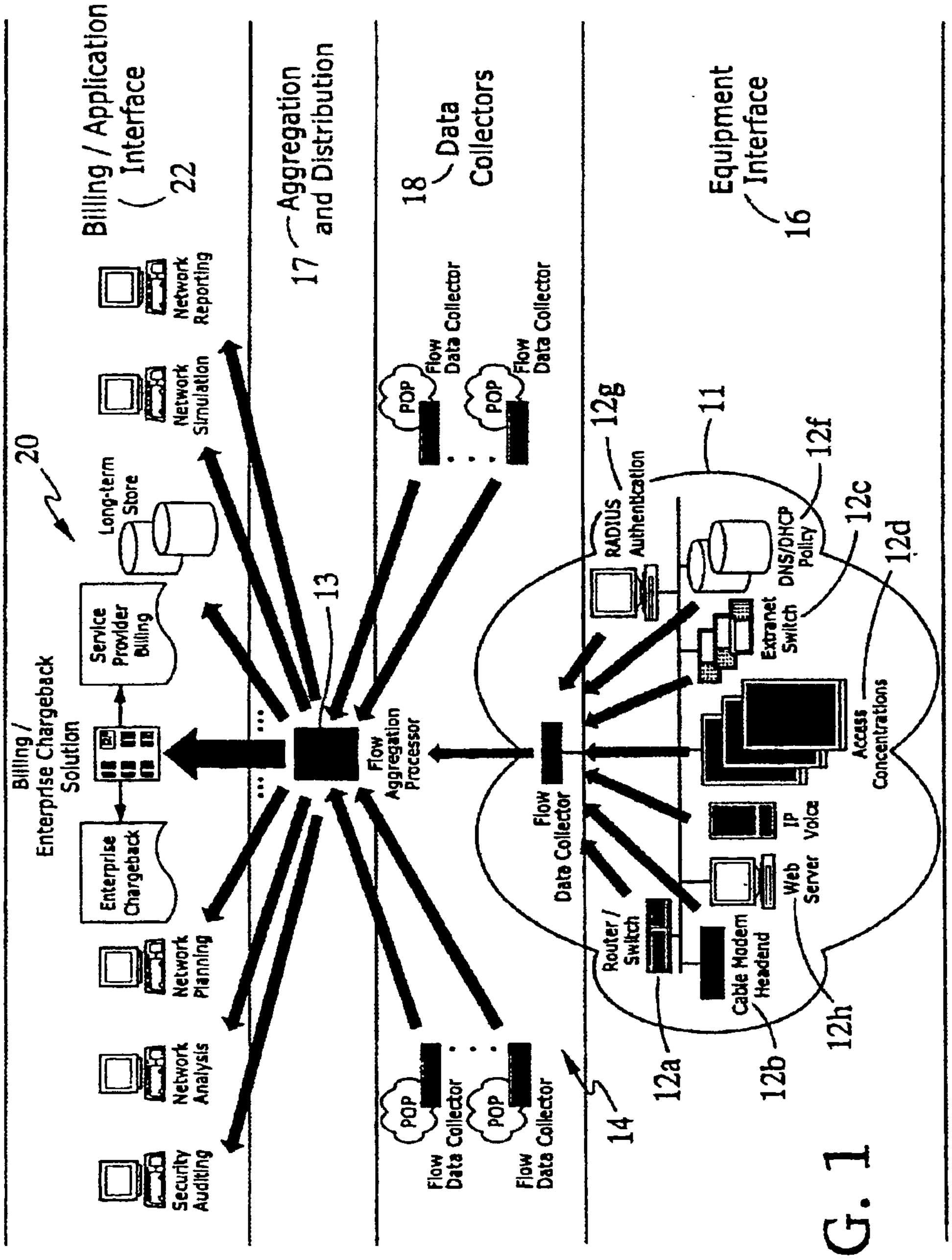

Billing / Application Interface
22
17 Aggregation and Distribution
18 Data Collectors
Equipment Interface
16
20
Billing / Enterprise Chargeback Solution
Enterprise Chargeback
Service Provider Billing
Long-term Store
Security Auditing
Network Analysis
Network Planning
Network Simulation
Network Reporting
13
Flow Aggregation Processor
POP
Flow Data Collector
POP
Flow Data Collector
POP
Flow Data Collector
POP
Flow Data Collector
14
Flow Data Collector
11
12g
RADIUS Authentication
12f
DNS/DHCP Policy
12c
Extranet Switch
12d
Access Concentrations
IP Voice
Web Server
12h
Router / Switch
12a
Cable Modem Headend
12b

FIG. 1

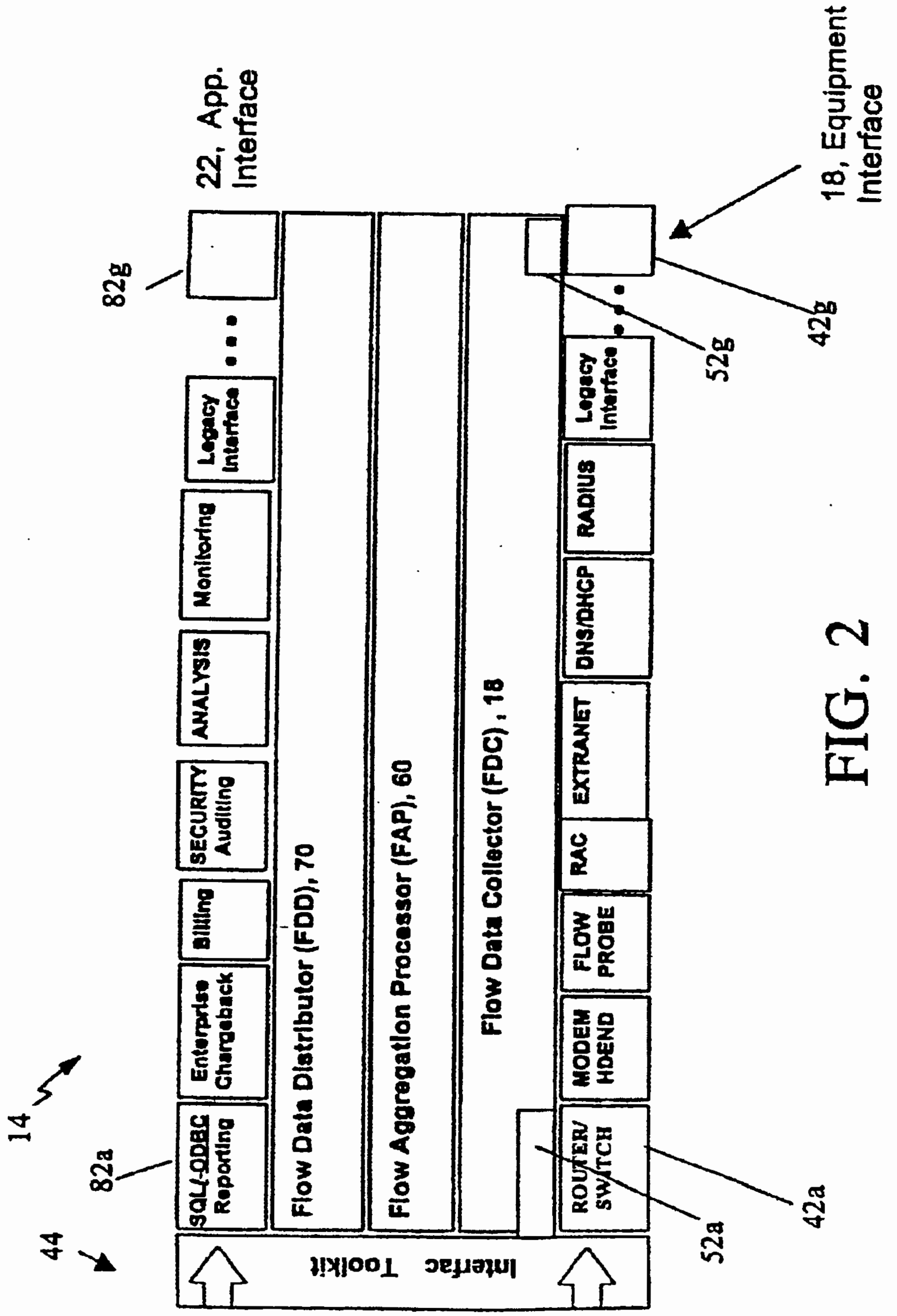

14
44
82a
82g
SQL/-ODBC Reporting
Enterprise Chargeback
Billing
SECURITY Auditing
ANALYSIS
Monitoring
Legacy Interface
22, App. Interface
Flow Data Distributor (FDD), 70
Flow Aggregation Processor (FAP), 60
Flow Data Collector (FDC), 18
Interface Toolkit
ROUTER/ SWITCH
MODEM HDEND
FLOW PROBE
RAC
EXTRANET
DNS/DHCP
RADIUS
Legacy Interface
18, Equipment Interface
52a
42a
52g
42g

FIG. 2

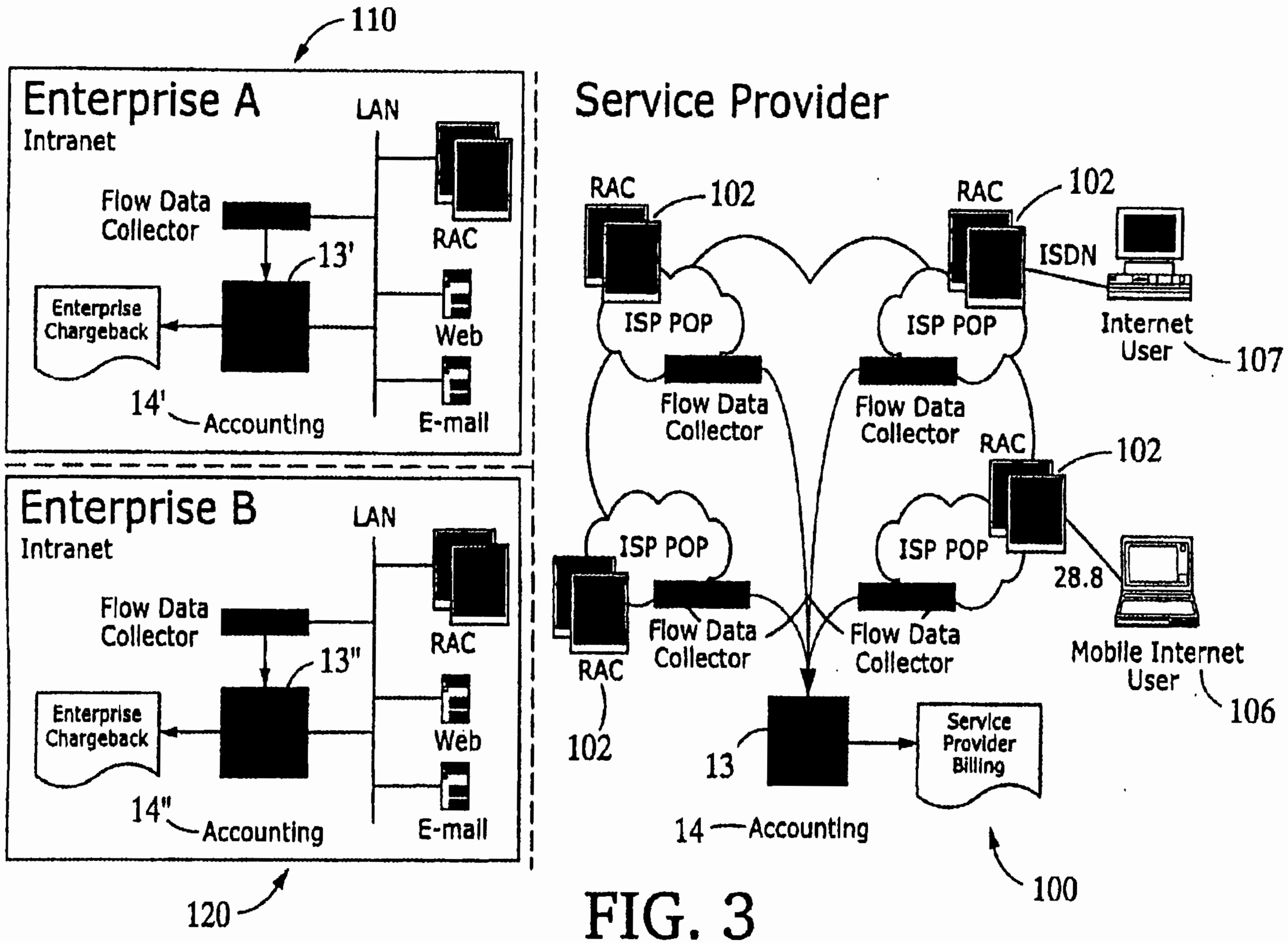

110
Enterprise A
Intranet
LAN
Flow Data Collector
RAC
13'
Enterprise Chargeback
Web
14'
Accounting
E-mail
Enterprise B
Intranet
LAN
Flow Data Collector
RAC
13"
Enterprise Chargeback
Web
14"
Accounting
E-mail
120
Service Provider
RAC
102
RAC
102
ISDN
ISP POP
ISP POP
Internet User
107
Flow Data Collector
Flow Data Collector
RAC
102
ISP POP
ISP POP
28.8
RAC
Flow Data Collector
Flow Data Collector
Mobile Internet User
106
102
13
Service Provider Billing
14
Accounting
100

FIG. 3

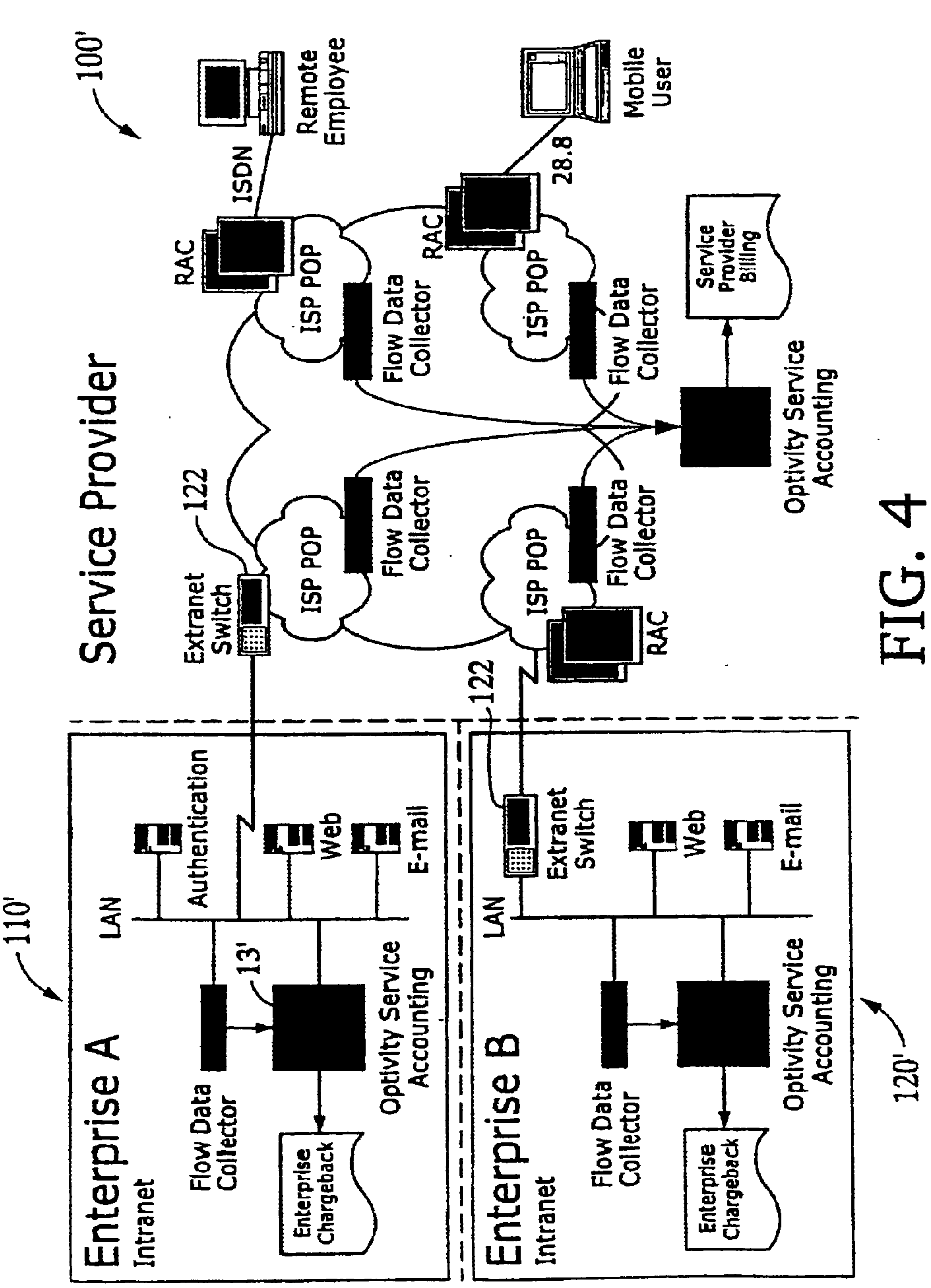

Service Provider
100'
Remote Employee
ISDN
RAC
ISP POP
Flow Data Collector
Mobile User
28.8
RAC
ISP POP
Flow Data Collector
Service Provider Billing
Optivity Service Accounting
122
Extranet Switch
ISP POP
Flow Data Collector
ISP POP
Flow Data Collector
RAC
122
110'
Enterprise A
Intranet
LAN
Authentication
Web
E-mail
13'
Flow Data Collector
Optivity Service Accounting
Enterprise Chargeback
Enterprise B
Intranet
LAN
Extranet Switch
Web
E-mail
Flow Data Collector
Optivity Service Accounting
Enterprise Chargeback
120'

FIG. 4

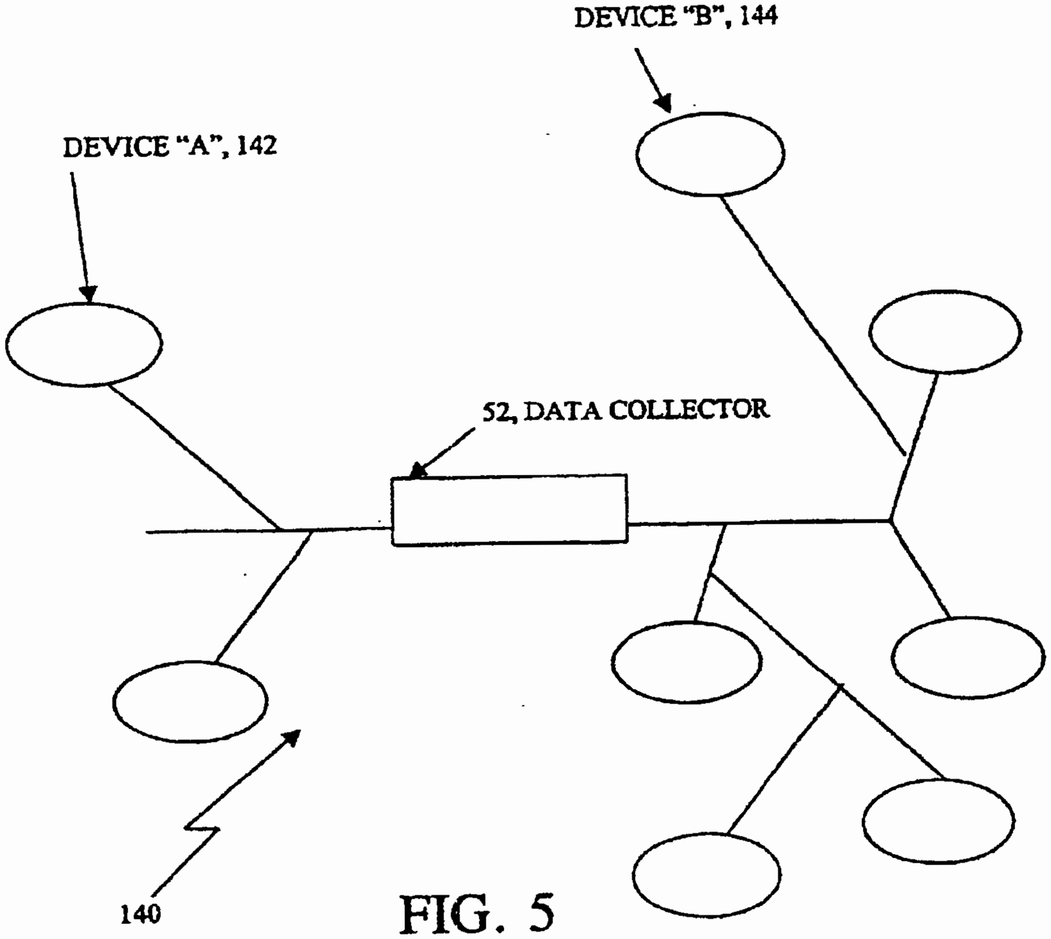


FIG. 5

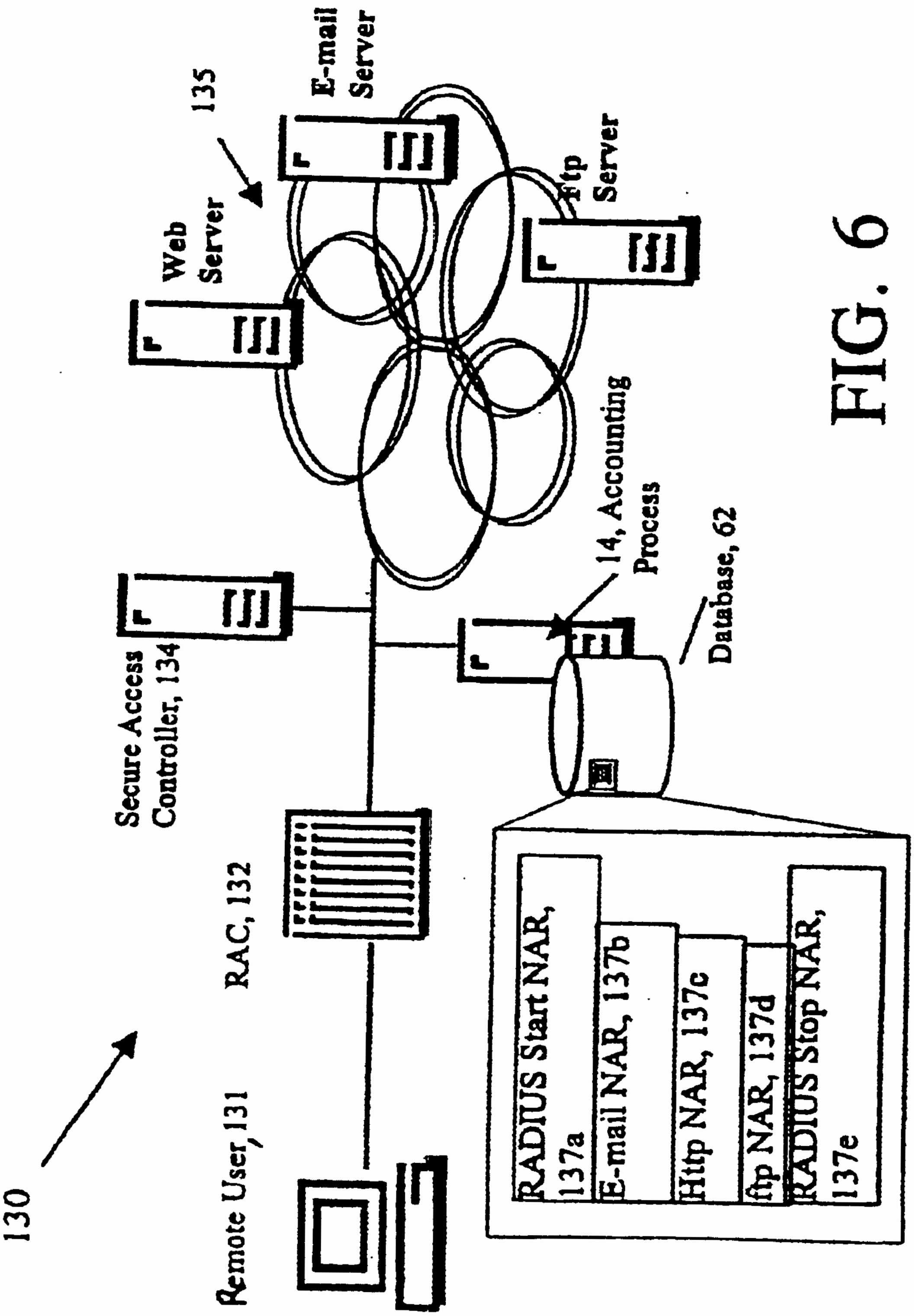

130
Remote User, 131
RAC, 132
Secure Access Controller, 134
Web Server
135
E-mail Server
Ftp Server
14, Accounting Process
Database, 62
RADIUS Start NAR, 137a
E-mail NAR, 137b
Http NAR, 137c
ftp NAR, 137d
RADIUS Stop NAR, 137e

FIG. 6

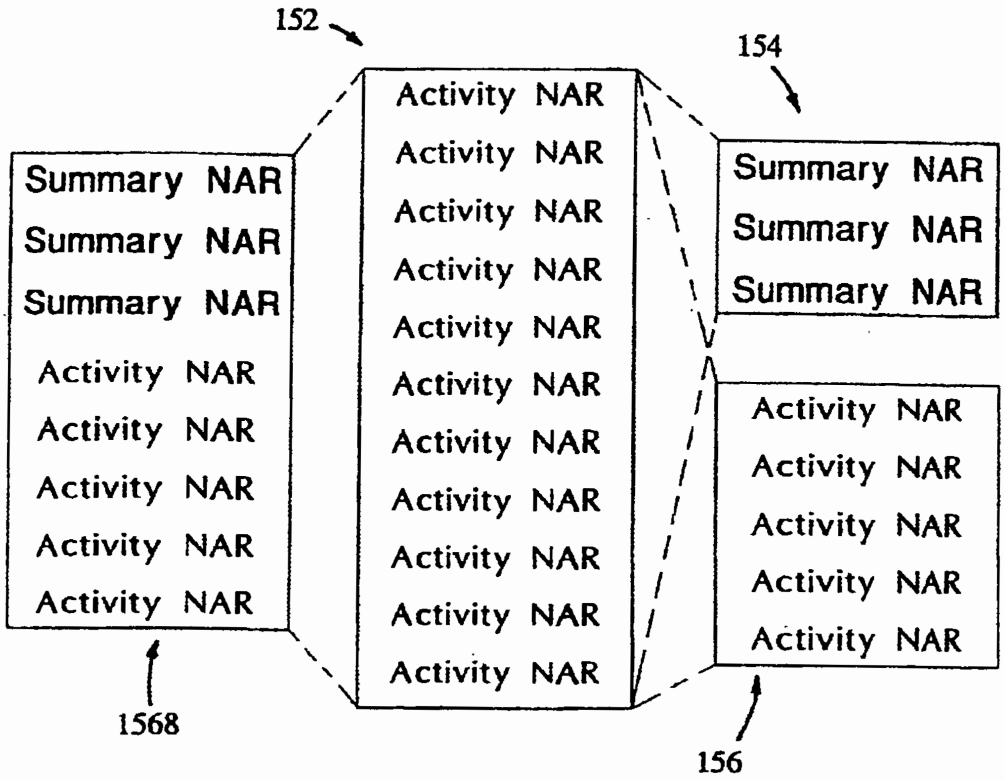

152
154
Summary NAR
Summary NAR
Summary NAR
Activity NAR
Activity NAR
Activity NAR
Activity NAR
Activity NAR
Activity NAR
Activity NAR
Activity NAR
Activity NAR
Activity NAR
Activity NAR
Activity NAR
Activity NAR
Activity NAR
Activity NAR
Activity NAR
Summary NAR
Summary NAR
Summary NAR
Activity NAR
Activity NAR
Activity NAR
Activity NAR
Activity NAR
1568
156

FIG. 7

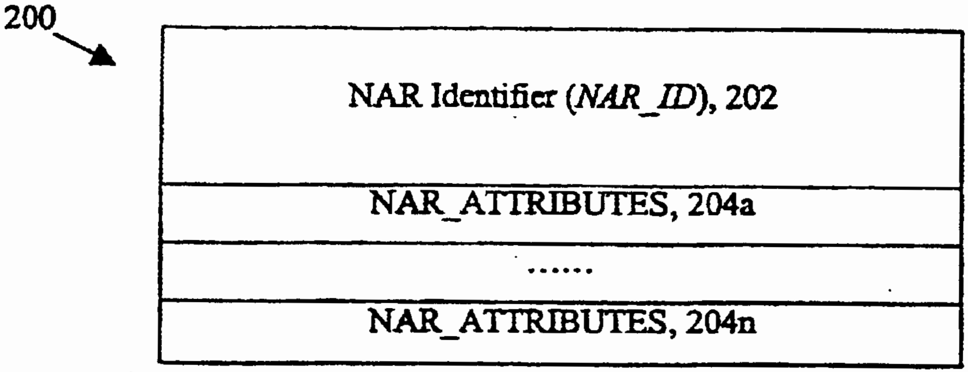

200
NAR Identifier (*NAR_ID*), 202
NAR_ATTRIBUTES, 204a
......
NAR_ATTRIBUTES, 204n
204

FIG. 8A

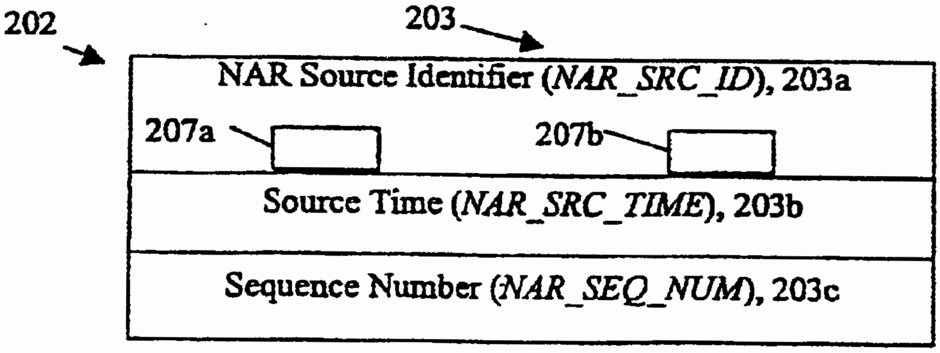

202
203
NAR Source Identifier (*NAR_SRC_ID*), 203a
207a
207b
Source Time (*NAR_SRC_TIME*), 203b
Sequence Number (*NAR_SEQ_NUM*), 203c

FIG. 8B

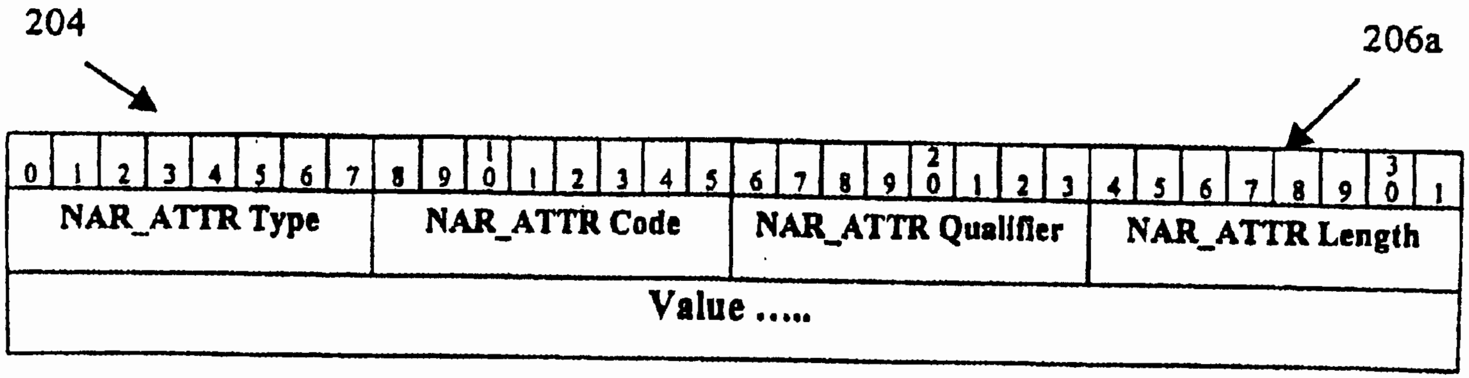

204
206a
0 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 20 1 2 3 4 5 6 7 8 9 30 1
NAR_ATTR Type
NAR_ATTR Code
NAR_ATTR Qualifier
NAR_ATTR Length
Value .....

FIG. 9A

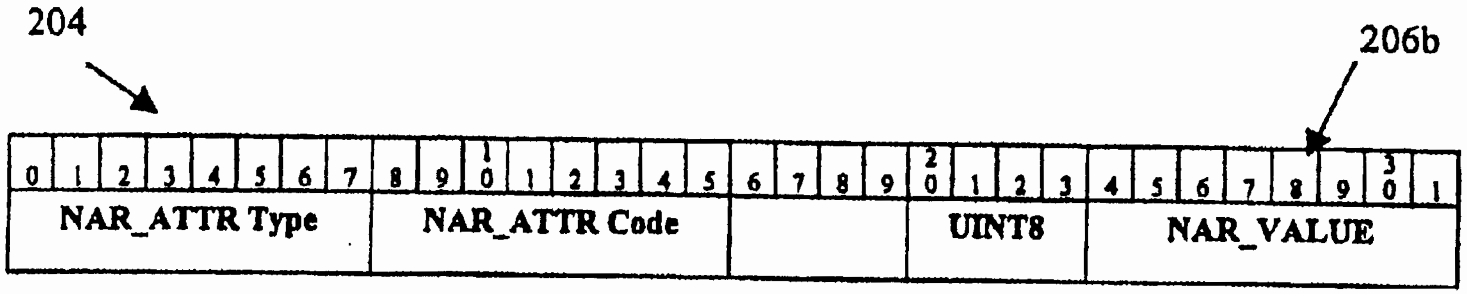
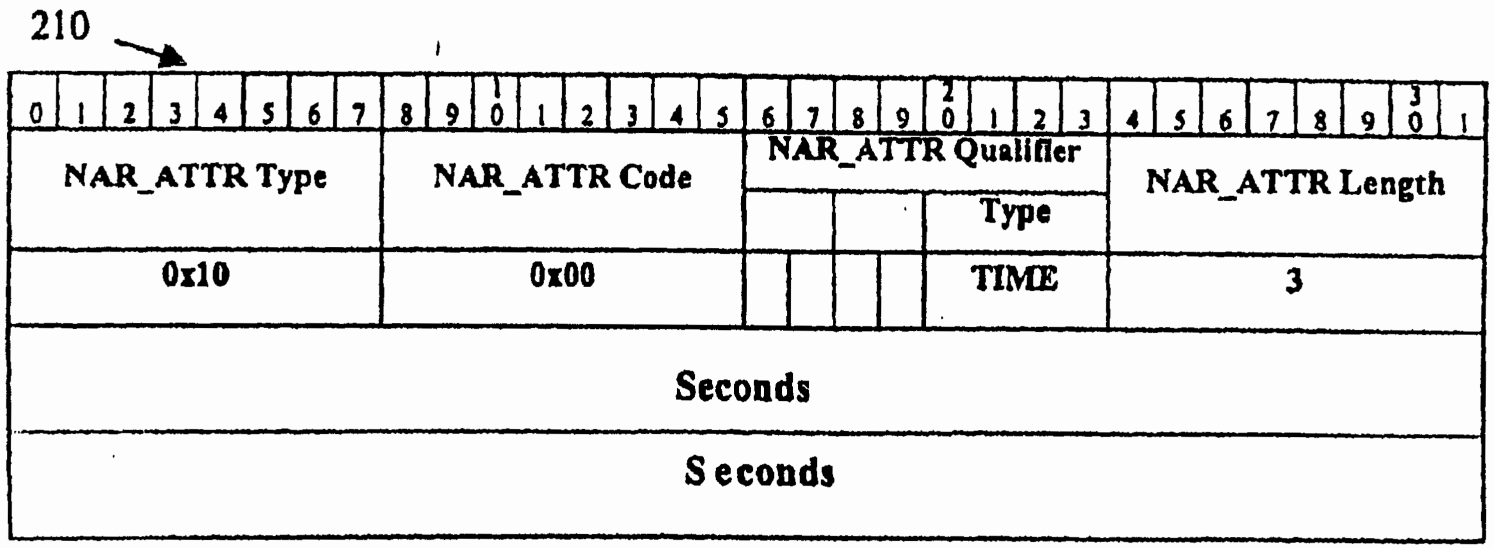

204
206b
0 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 20 1 2 3 4 5 6 7 8 9 30 1
NAR_ATTR Type
NAR_ATTR Code
UINT8
NAR_VALUE

FIG. 9B



0 1 2 3 4 5 6 7	8 9 10 1 2 3 4 5	6 7	8 9	20 1 2 3	4 5 6 7 8 9 30 1
NAR_ATTR Type	NAR_ATTR Code	NAR_ATTR Qualifier			NAR_ATTR Length
				Type	
0x10	0x00			TIME	3
Seconds					
S econds					

FIG. 10

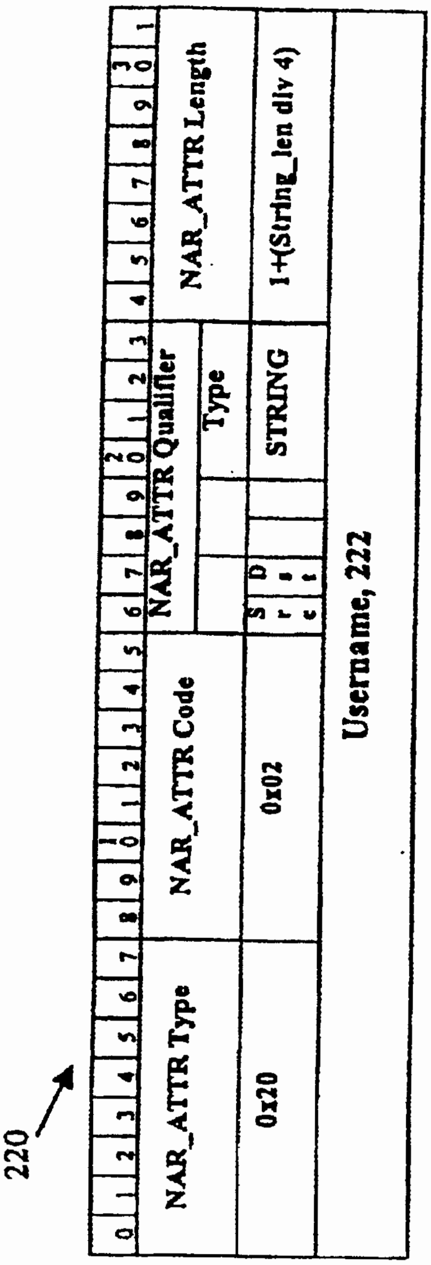


FIG. 11A

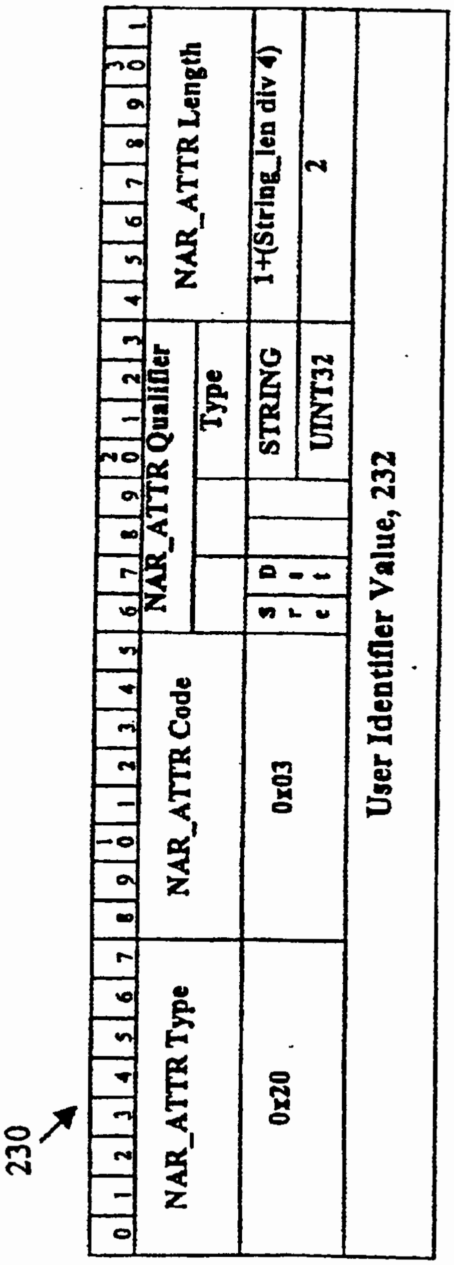
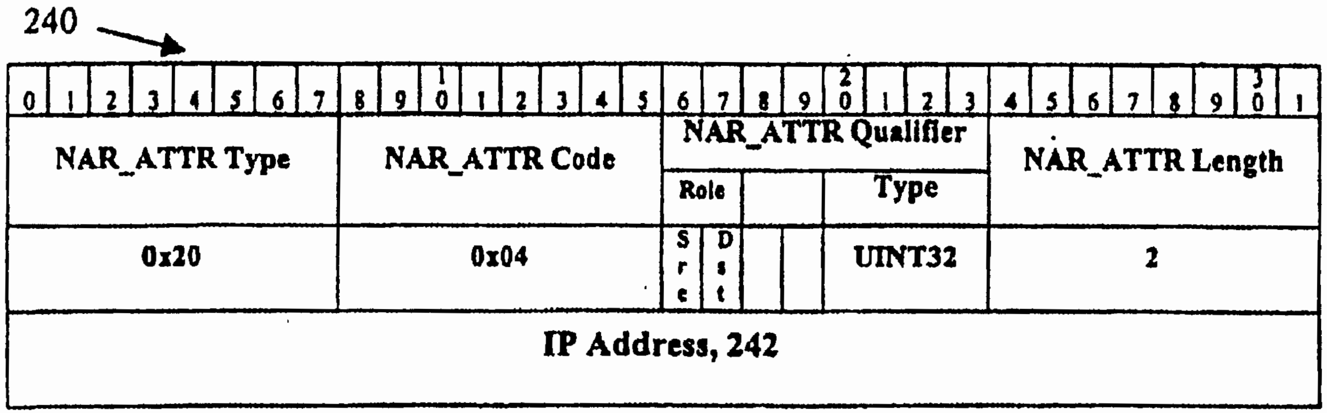
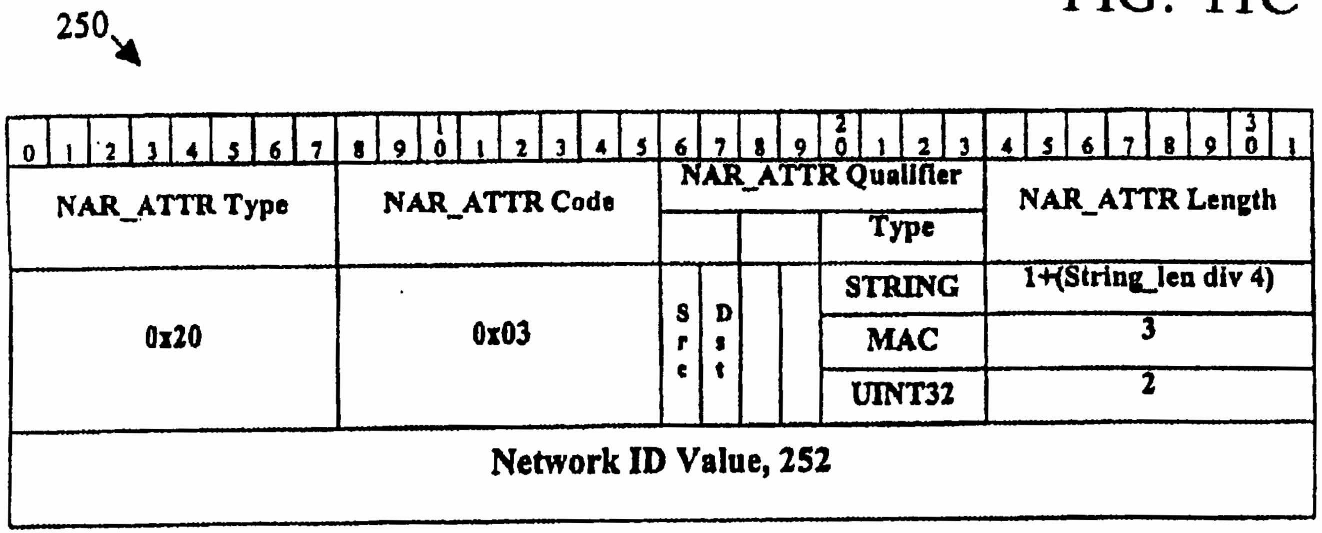


FIG. 11B



NAR_ATTR Type	NAR_ATTR Code	NAR_ATTR Qualifier					NAR_ATTR Length
		Role				Type	
0x20	0x04	Src	Dst			UINT32	2
IP Address, 242							

FIG. 11C



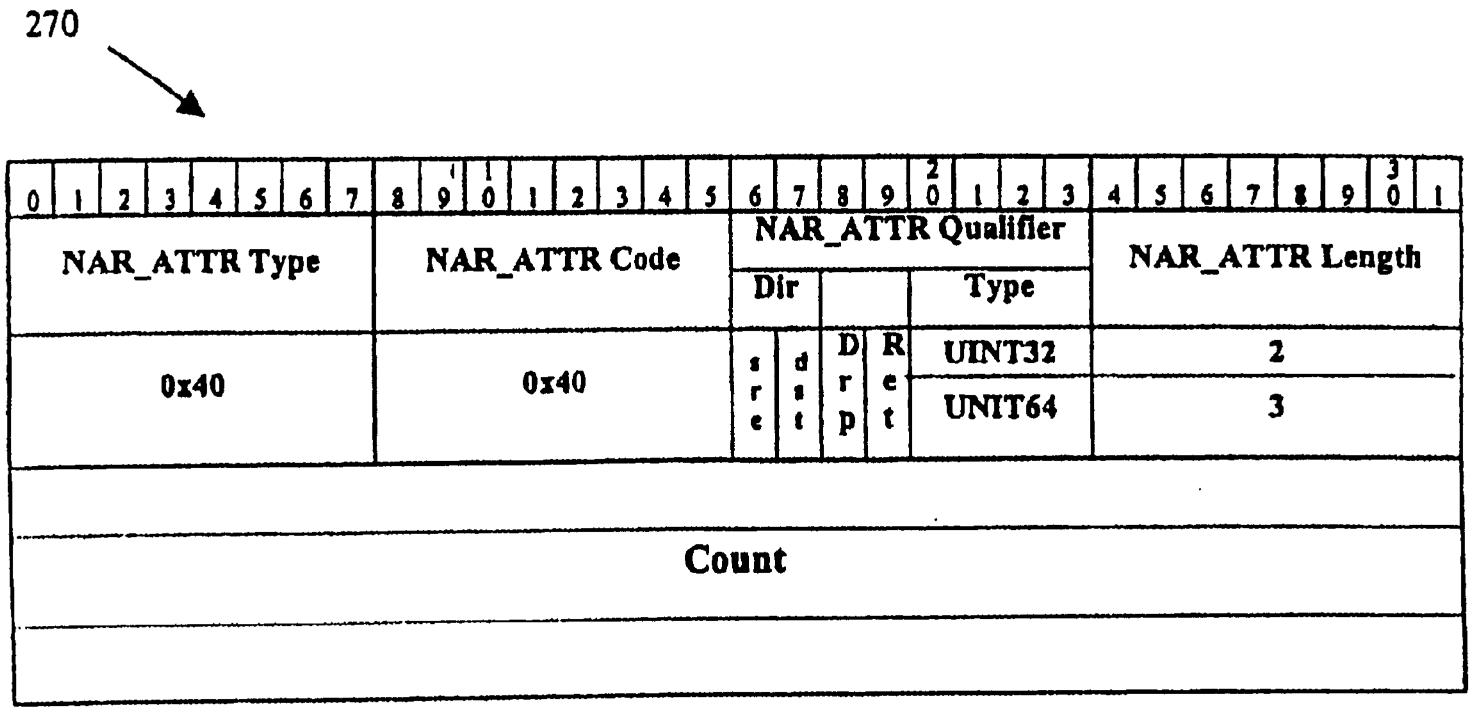
NAR_ATTR Type	NAR_ATTR Code	NAR_ATTR Qualifier					NAR_ATTR Length
						Type	
0x20	0x03	Src	Dst			STRING	1+(String_len div 4)
						MAC	3
						UINT32	2
Network ID Value, 252							

FIG. 11D

260

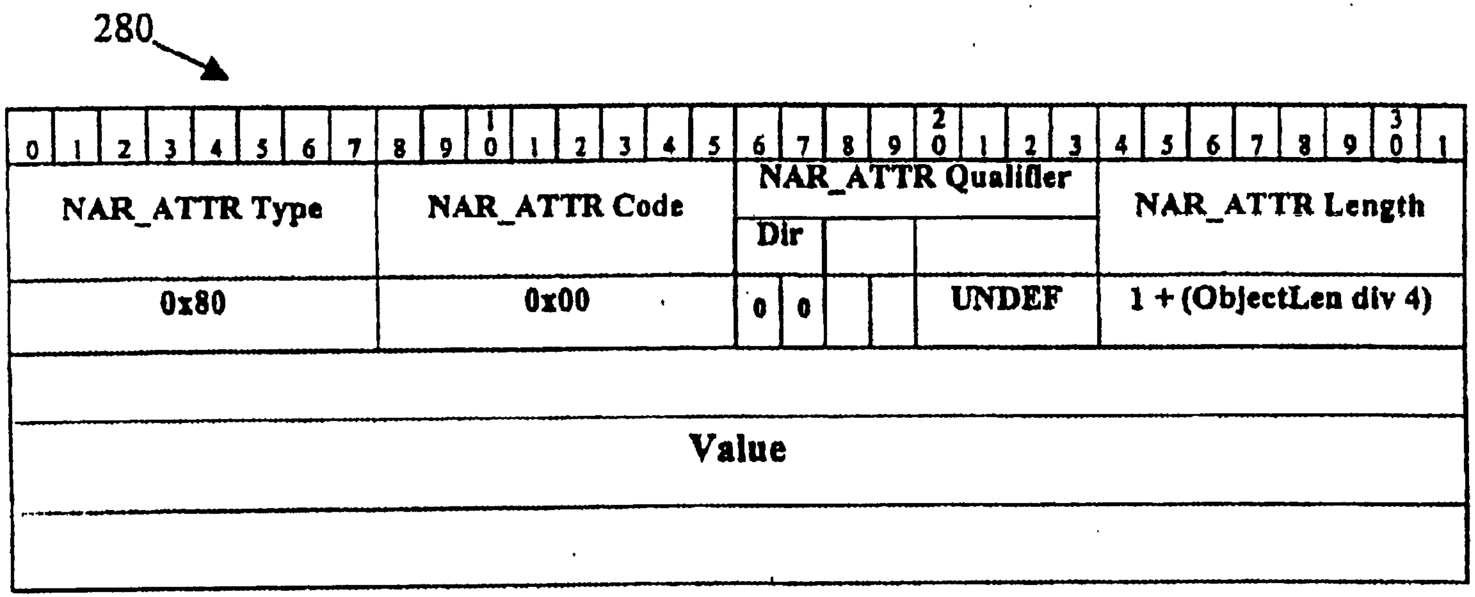
<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="8" rowspan="2">NAR_ATTR Type</td><td colspan="8" rowspan="2">NAR_ATTR Code</td><td colspan="8">NAR_ATTR Qualifier</td><td colspan="8" rowspan="2">NAR_ATTR Length</td></tr>
<tr><td colspan="2"></td><td colspan="2"></td><td colspan="4">Type</td></tr>
<tr><td colspan="8">0x20</td><td colspan="8">0x06</td><td></td><td></td><td></td><td></td><td colspan="4">FLOW</td><td colspan="8">5</td></tr>
<tr><td colspan="32">IP Source Address, 262</td></tr>
<tr><td colspan="32">IP Destination Address, 263</td></tr>
<tr><td colspan="16">Transport Protocol, 264</td><td colspan="16">Type of Service, 265</td></tr>
<tr><td colspan="16">Source Port, 266</td><td colspan="16">Destination Port, 267</td></tr>
</table>

FIG. 11E



0 1 2 3 4 5 6 7	8 9 10 1 2 3 4 5	6	7	8	9	20 1 2 3	4 5 6 7 8 9 30 1
NAR_ATTR Type	NAR_ATTR Code	NAR_ATTR Qualifier					NAR_ATTR Length
		Dir				Type	
0x40	0x40	src	dst	Drp	Ret	UINT32 UNIT64	2 3
Count							

FIG. 12



<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="8" rowspan="2">NAR_ATTR Type</td><td colspan="8" rowspan="2">NAR_ATTR Code</td><td colspan="8">NAR_ATTR Qualifier</td><td colspan="8" rowspan="2">NAR_ATTR Length</td></tr>
<tr><td colspan="2">Dir</td><td colspan="2"></td><td colspan="4"></td></tr>
<tr><td colspan="8">0x80</td><td colspan="8">0x00</td><td>0</td><td>0</td><td></td><td></td><td colspan="4">UNDEF</td><td colspan="8">1 + (ObjectLen div 4)</td></tr>
<tr><td colspan="32"></td></tr>
<tr><td colspan="32">Value</td></tr>
<tr><td colspan="32"></td></tr>
</table>

FIG. 13A

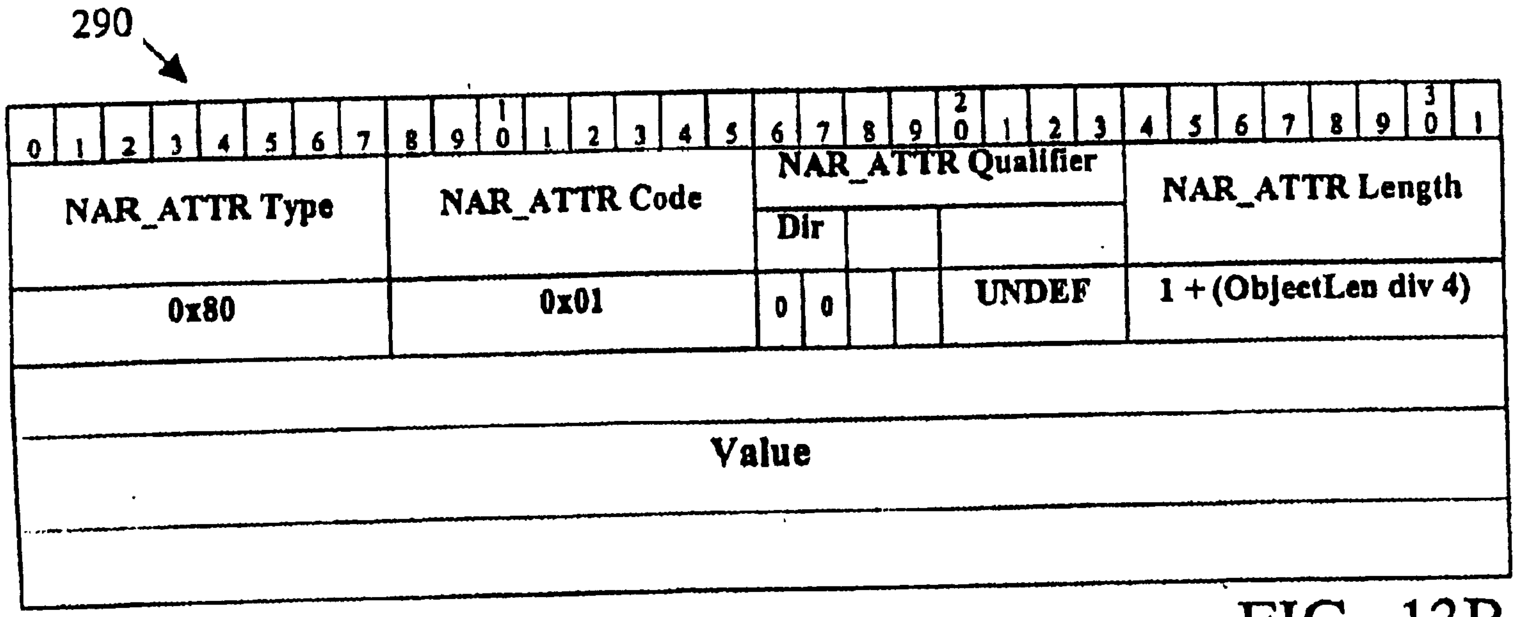

290
0 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 20 1 2 3 4 5 6 7 8 9 30 1
NAR_ATTR Type
NAR_ATTR Code
NAR_ATTR Qualifier
Dir
NAR_ATTR Length
0x80
0x01
0
0
UNDEF
1 + (ObjectLen div 4)
Value

FIG. 13B

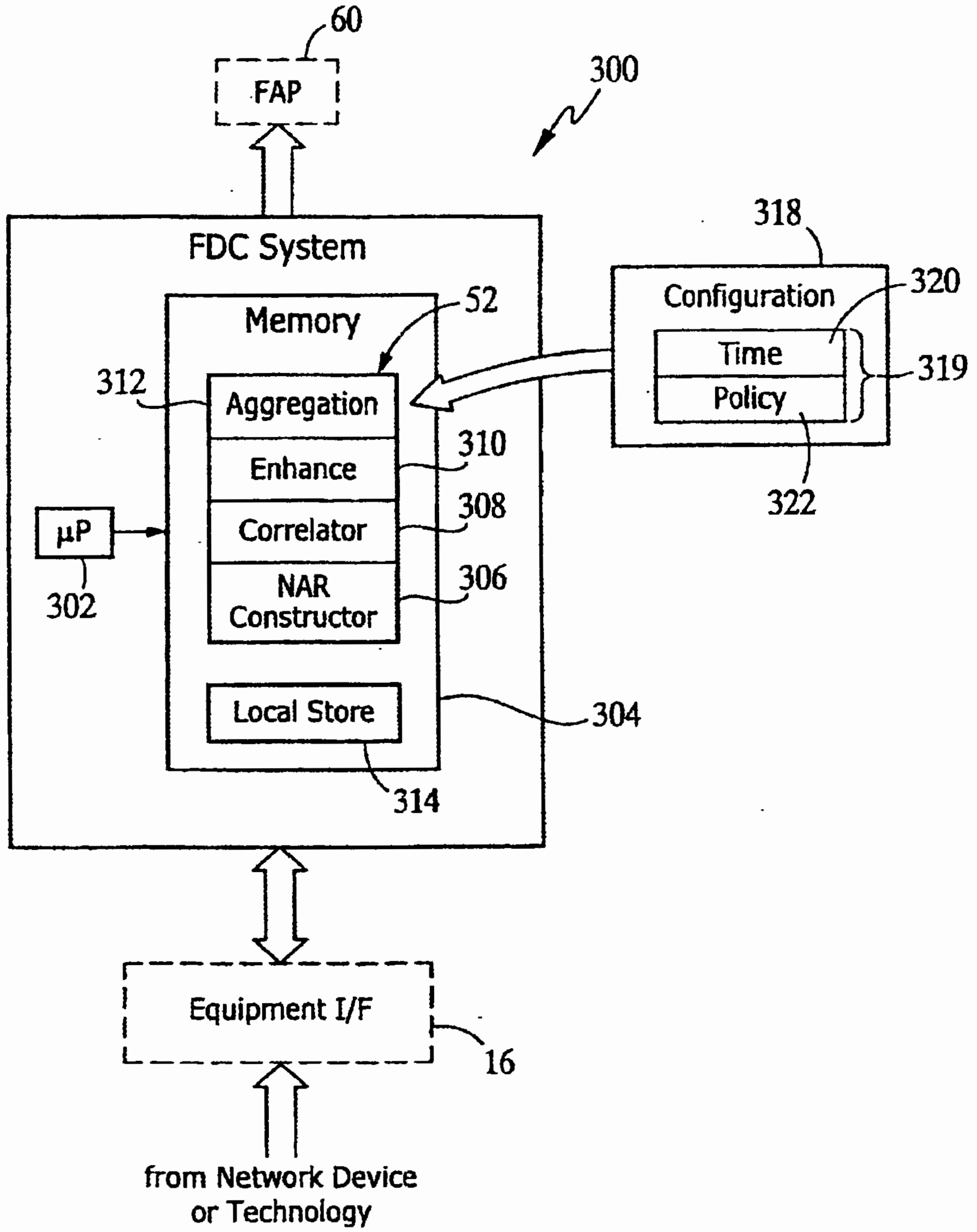

60
FAP
300
FDC System
Memory
52
312
Aggregation
Enhance
310
Correlator
308
NAR
Constructor
306
µP
302
Local Store
304
314
318
Configuration
320
Time
Policy
319
322
Equipment I/F
16
from Network Device
or Technology

FIG.14

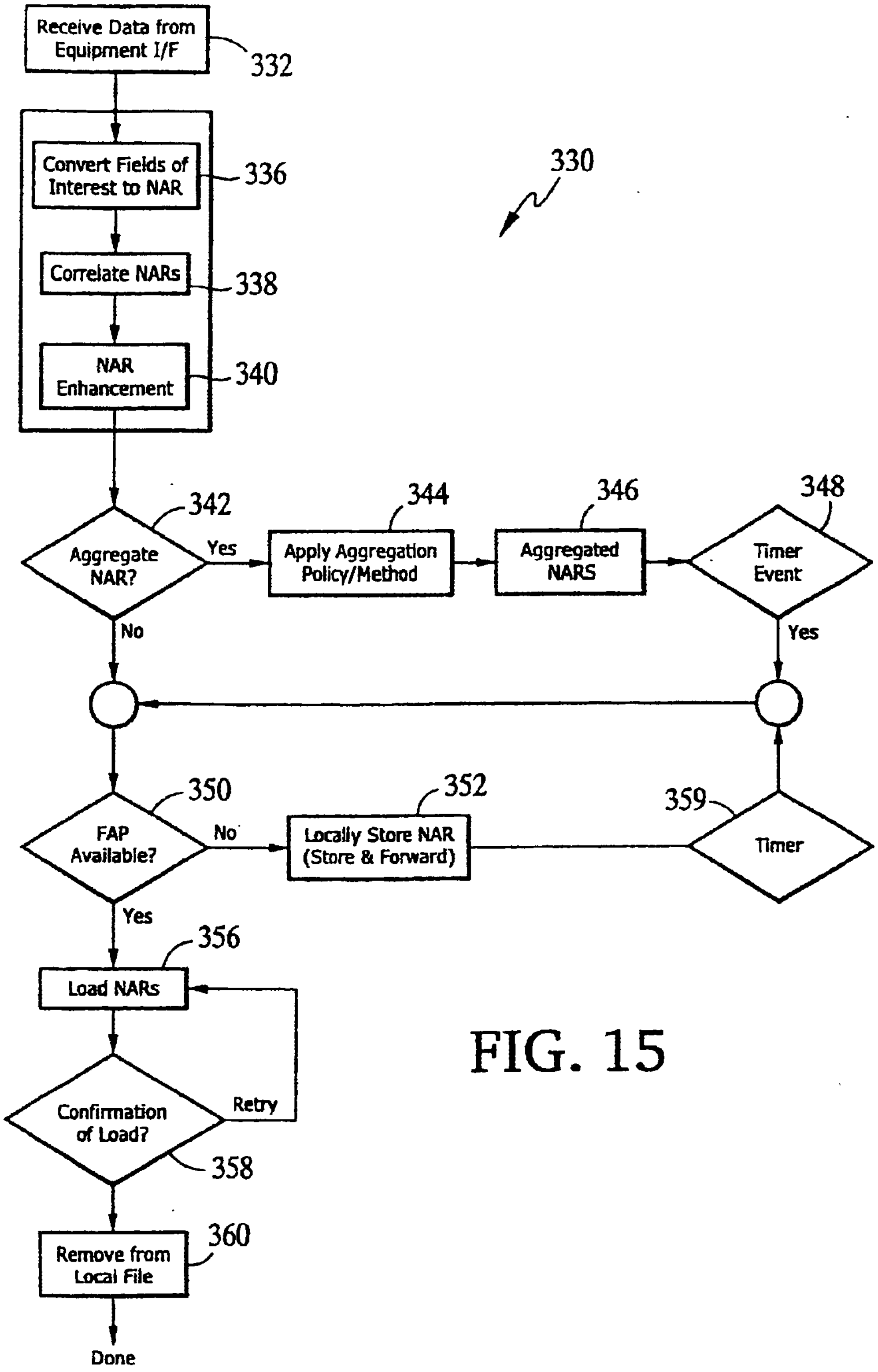

Receive Data from Equipment I/F
332
Convert Fields of Interest to NAR
336
330
Correlate NARs
338
NAR Enhancement
340
342
Aggregate NAR?
Yes
344
Apply Aggregation Policy/Method
346
Aggregated NARS
348
Timer Event
Yes
No
350
FAP Available?
No
352
Locally Store NAR (Store & Forward)
359
Timer
Yes
356
Load NARs
Confirmation of Load?
Retry
358
360
Remove from Local File
Done

FIG. 15

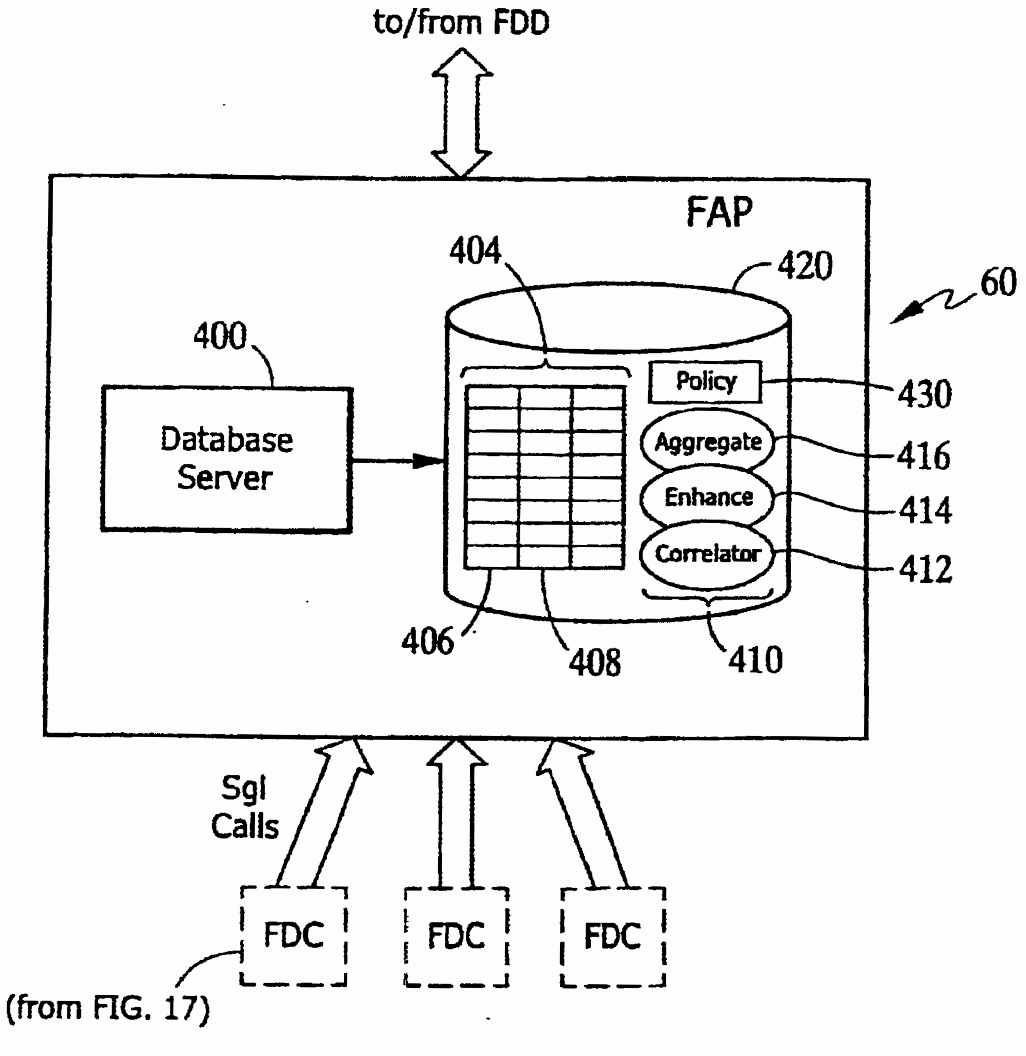

to/from FDD
FAP
404
420
60
400
Database Server
Policy
430
Aggregate
416
Enhance
414
Correlator
412
406
408
410
Sgl Calls
FDC
FDC
FDC
(from FIG. 17)

FIG. 16

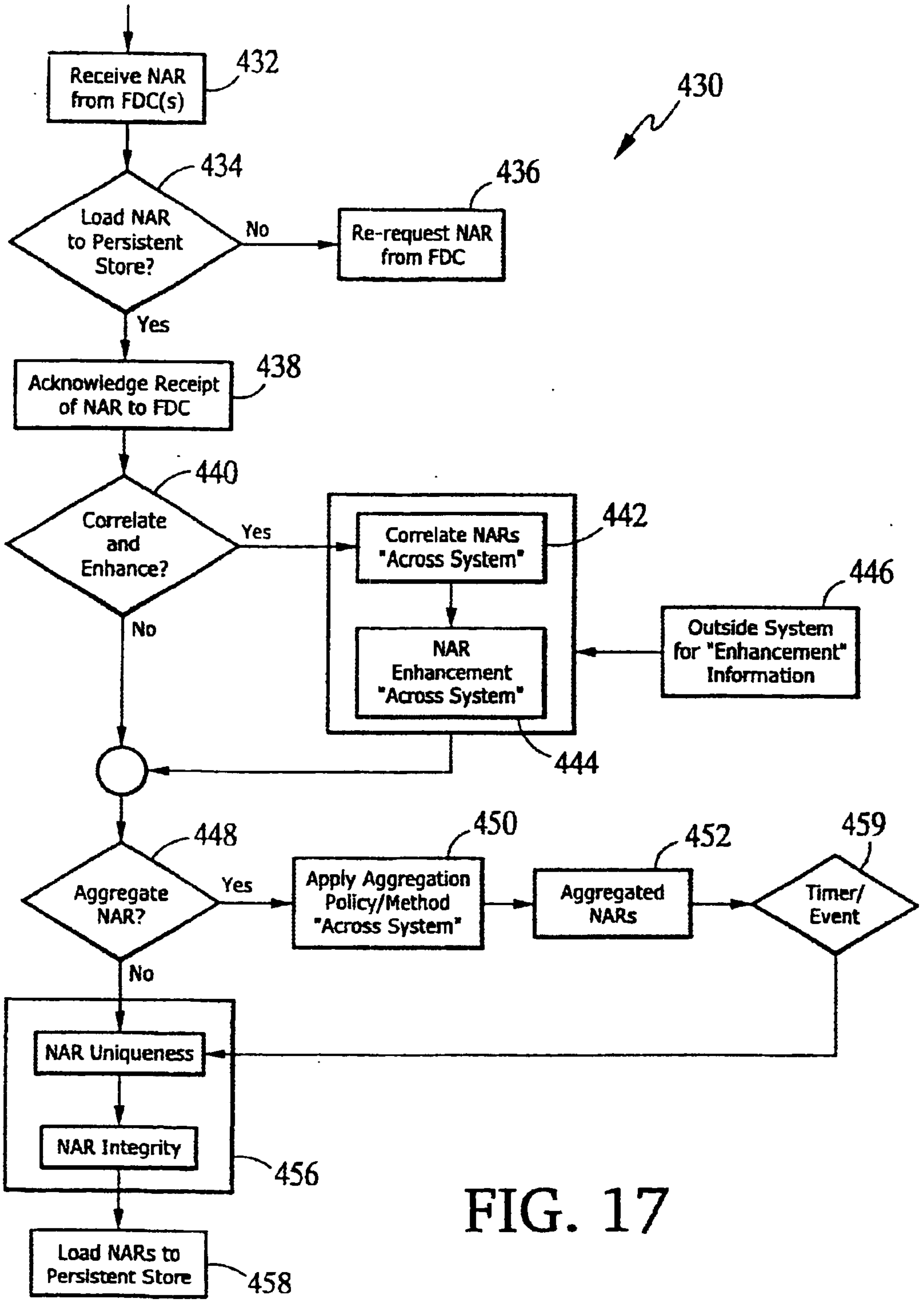

Receive NAR from FDC(s)
432
430
Load NAR to Persistent Store?
434
No
Re-request NAR from FDC
436
Yes
Acknowledge Receipt of NAR to FDC
438
Correlate and Enhance?
440
Yes
Correlate NARs "Across System"
442
NAR Enhancement "Across System"
444
Outside System for "Enhancement" Information
446
No
Aggregate NAR?
448
Yes
Apply Aggregation Policy/Method "Across System"
450
Aggregated NARs
452
Timer/ Event
459
No
NAR Uniqueness
NAR Integrity
456
Load NARs to Persistent Store
458

FIG. 17

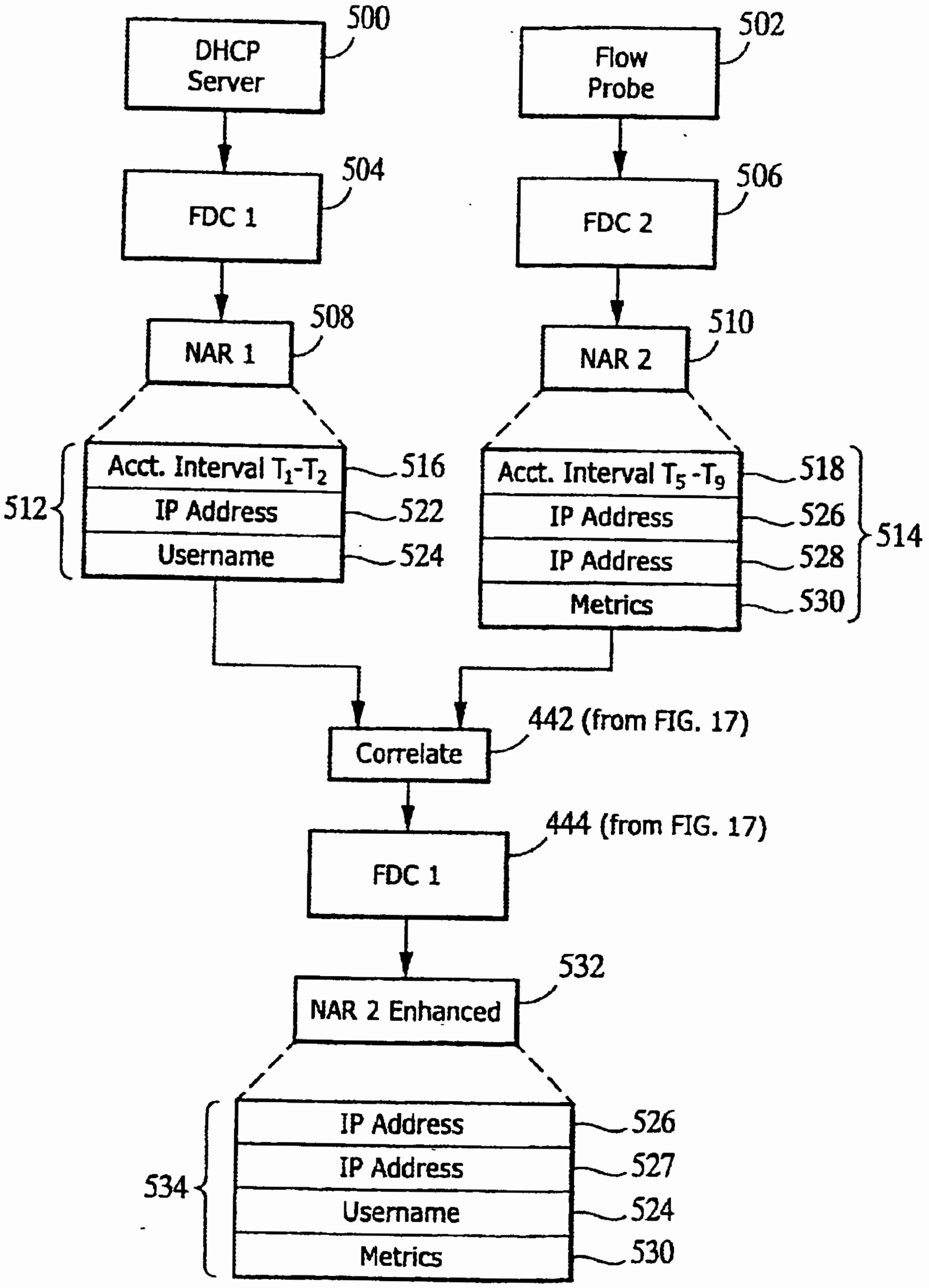

DHCP Server
500
Flow Probe
502
FDC 1
504
FDC 2
506
NAR 1
508
NAR 2
510
Acct. Interval T_1-T_2
516
IP Address
522
Username
524
512
Acct. Interval T_5-T_9
518
IP Address
526
IP Address
528
Metrics
530
514
Correlate
442 (from FIG. 17)
FDC 1
444 (from FIG. 17)
NAR 2 Enhanced
532
IP Address
526
IP Address
527
Username
524
Metrics
530
534

FIG. 18

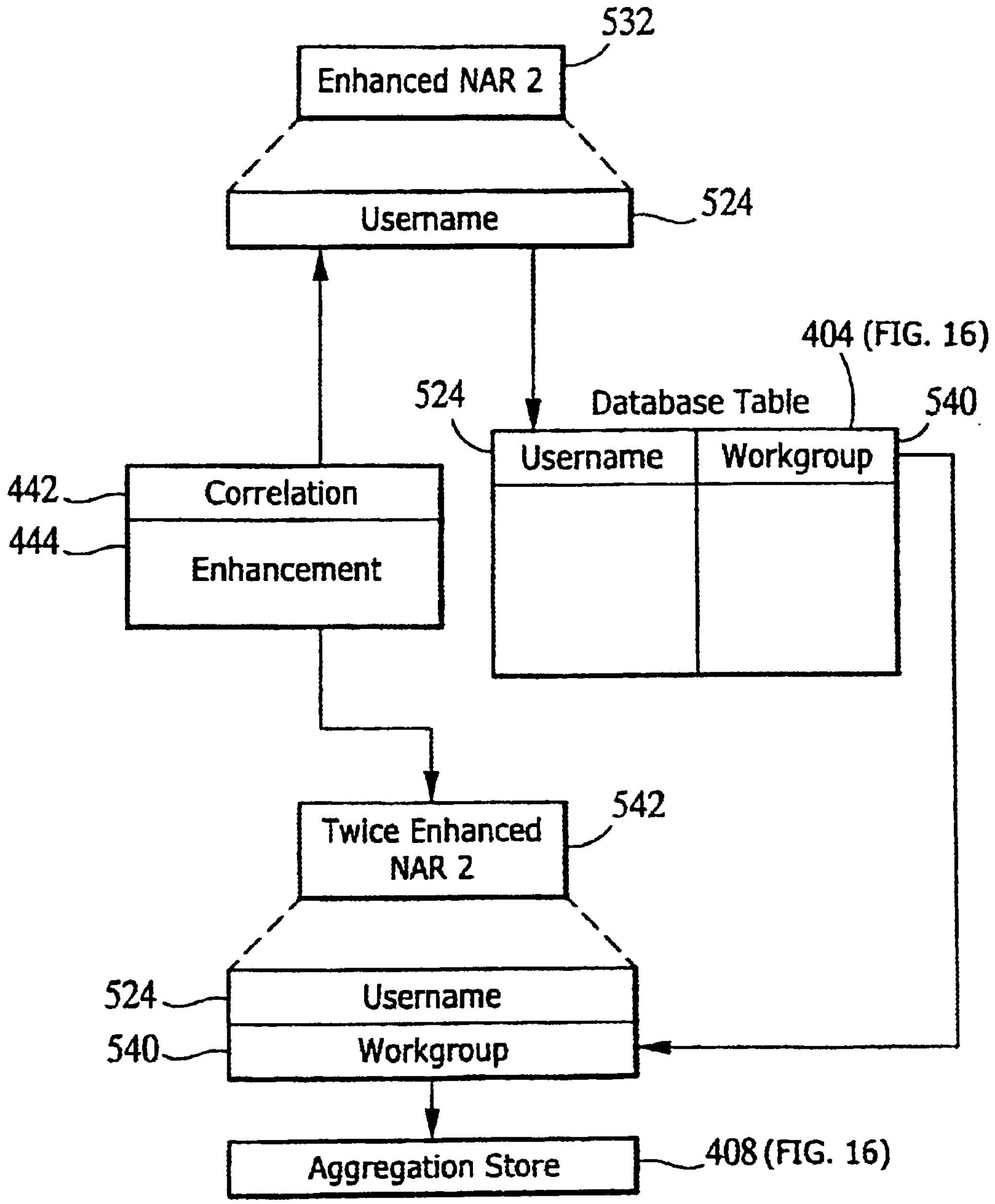

532
Enhanced NAR 2
Username
524
404 (FIG. 16)
524
Database Table
540
Username
Workgroup
442
444
Correlation
Enhancement
542
Twice Enhanced NAR 2
524
540
Username
Workgroup
Aggregation Store
408 (FIG. 16)

FIG. 19

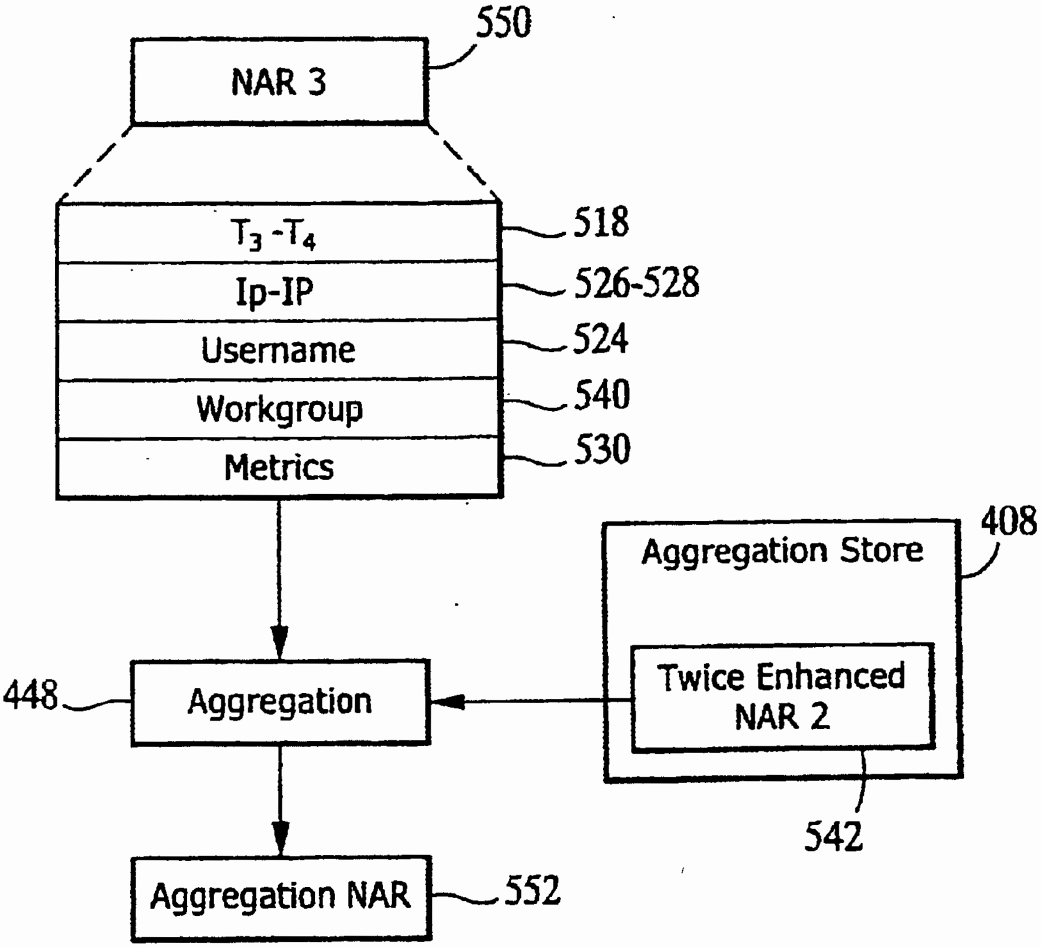

550
NAR 3
T_3 -T_4
518
Ip-IP
526-528
Username
524
Workgroup
540
Metrics
530
Aggregation Store
408
Twice Enhanced NAR 2
542
448
Aggregation
Aggregation NAR
552

FIG. 20

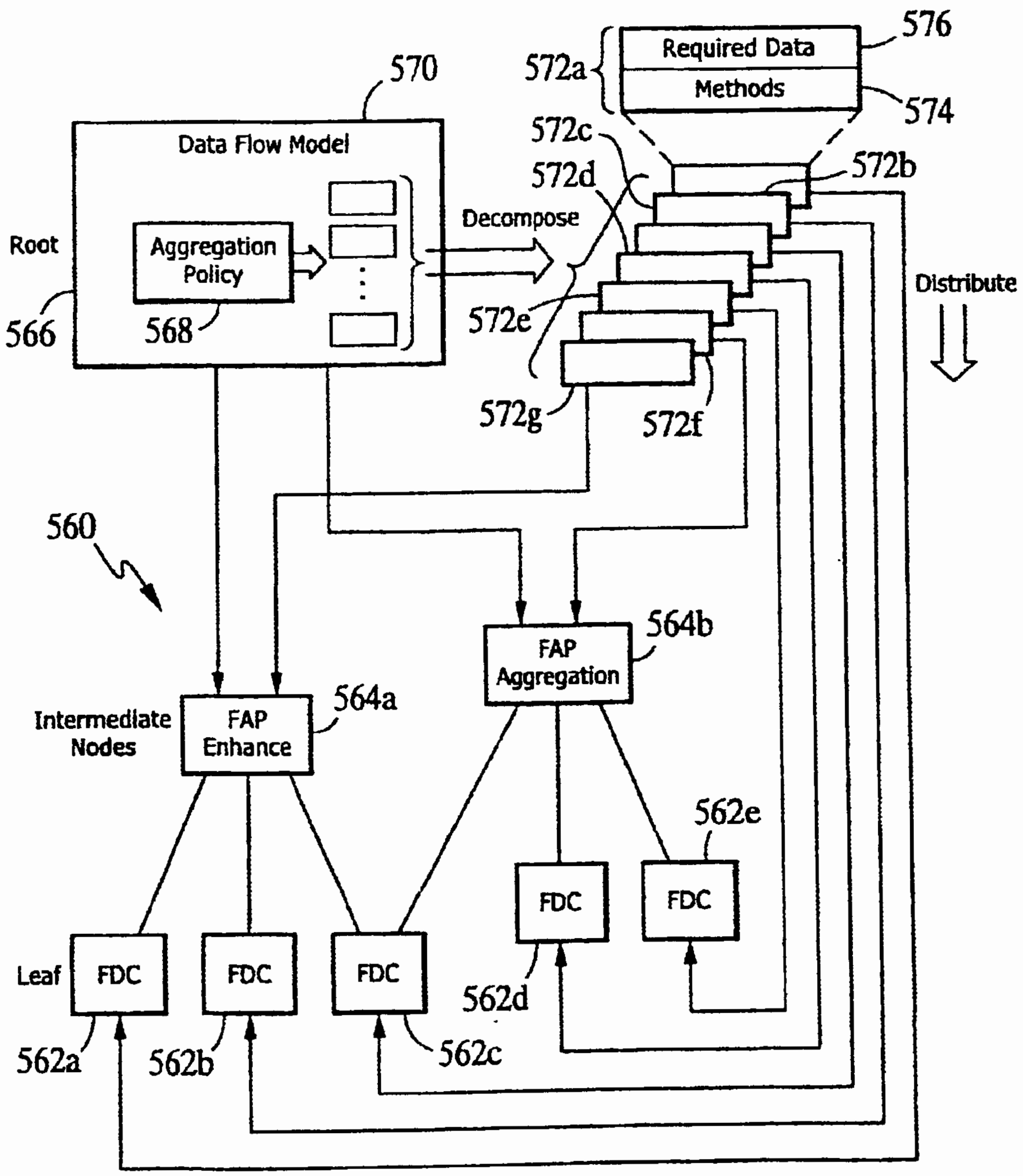

576
Required Data
Methods
574
572a
570
572c
572d
572b
Data Flow Model
Root
Aggregation Policy
566
568
Decompose
Distribute
572e
572g
572f
560
FAP Aggregation
564b
Intermediate Nodes
FAP Enhance
564a
562e
FDC
FDC
562d
Leaf
FDC
FDC
FDC
562a
562b
562c

FIG. 21

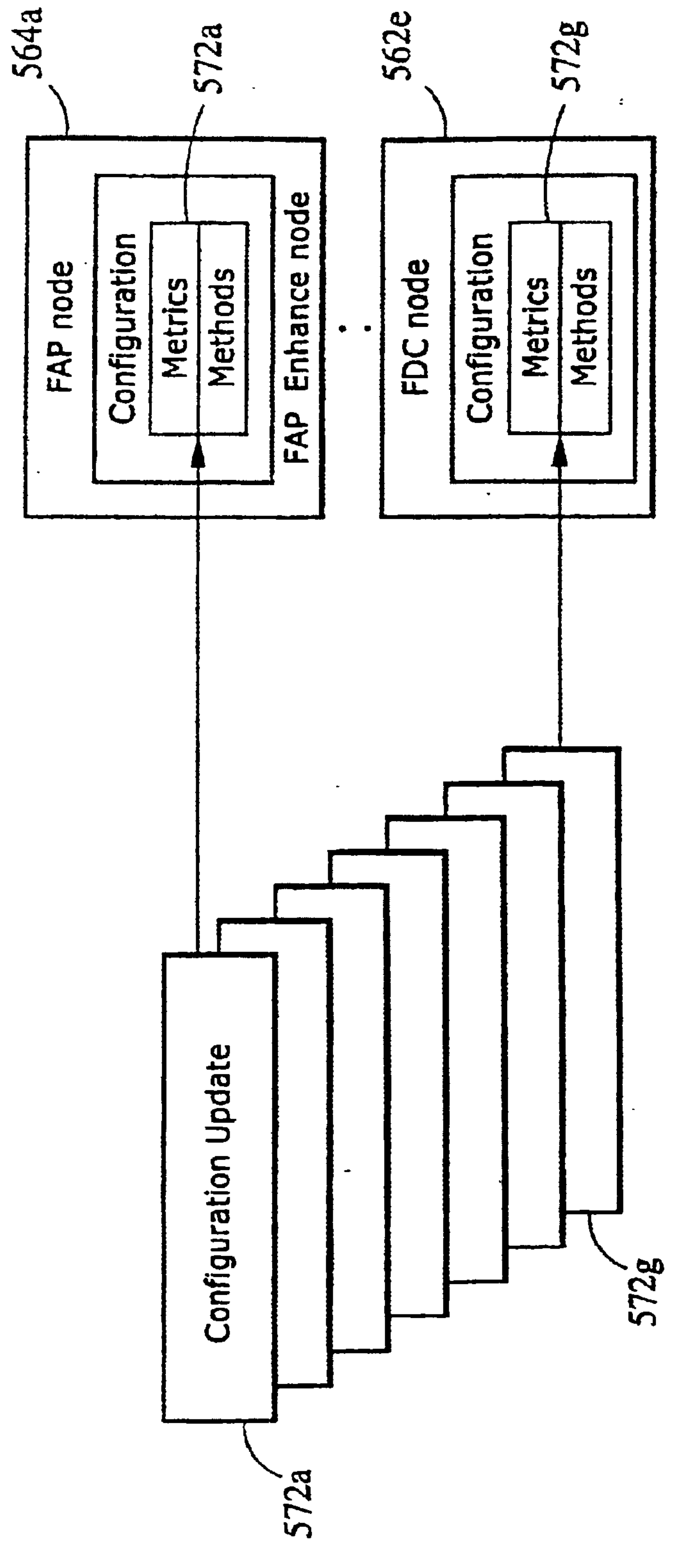

564a
572a
562e
572g
FAP node
Configuration
Metrics
Methods
FAP Enhance node
FDC node
Configuration
Metrics
Methods
Configuration Update
572a
572g

FIG. 22

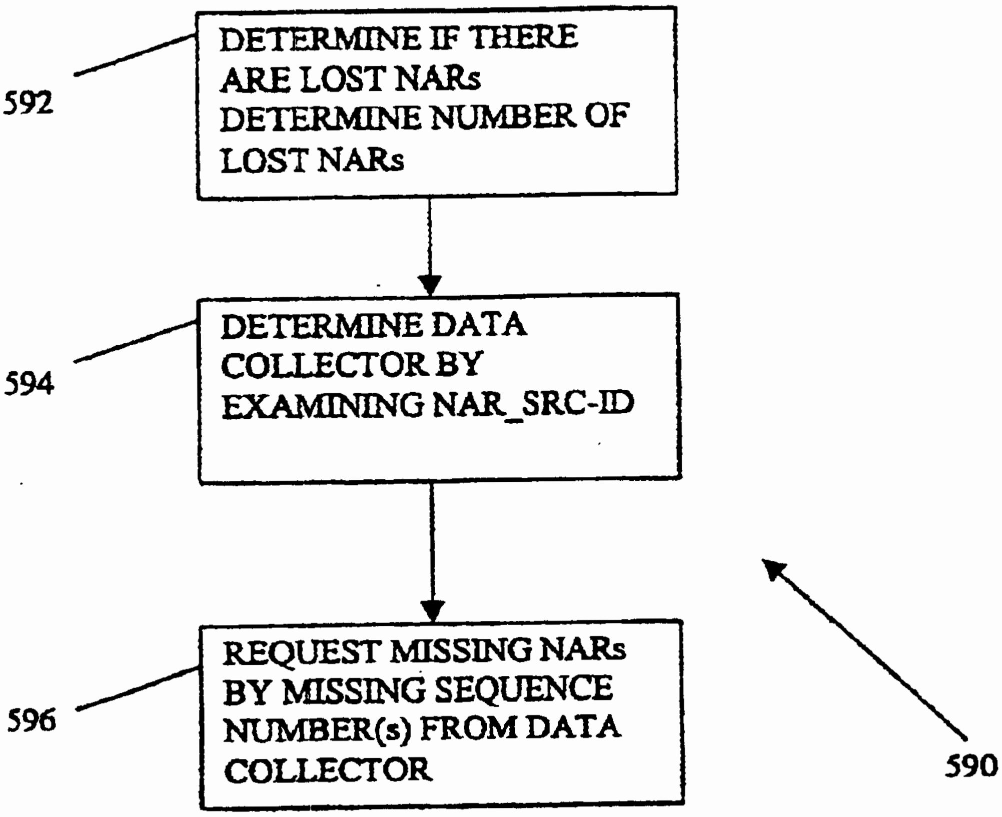

592
DETERMINE IF THERE ARE LOST NARs DETERMINE NUMBER OF LOST NARs
594
DETERMINE DATA COLLECTOR BY EXAMINING NAR_SRC-ID
596
REQUEST MISSING NARs BY MISSING SEQUENCE NUMBER(s) FROM DATA COLLECTOR
590

FIG. 23

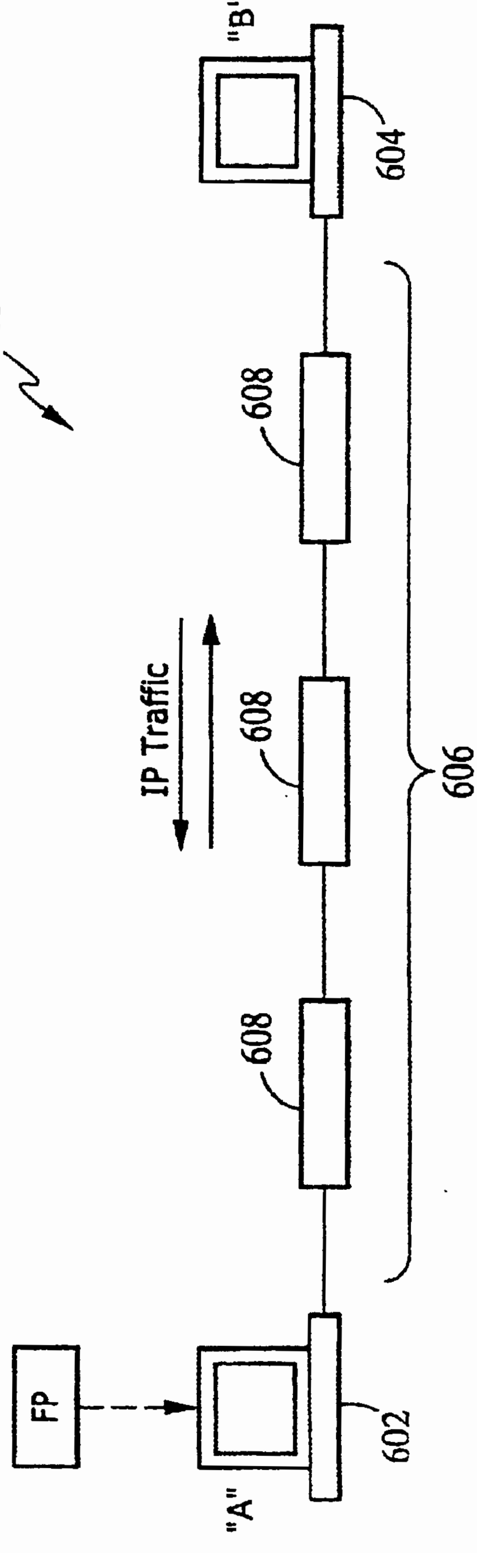


FIG. 24

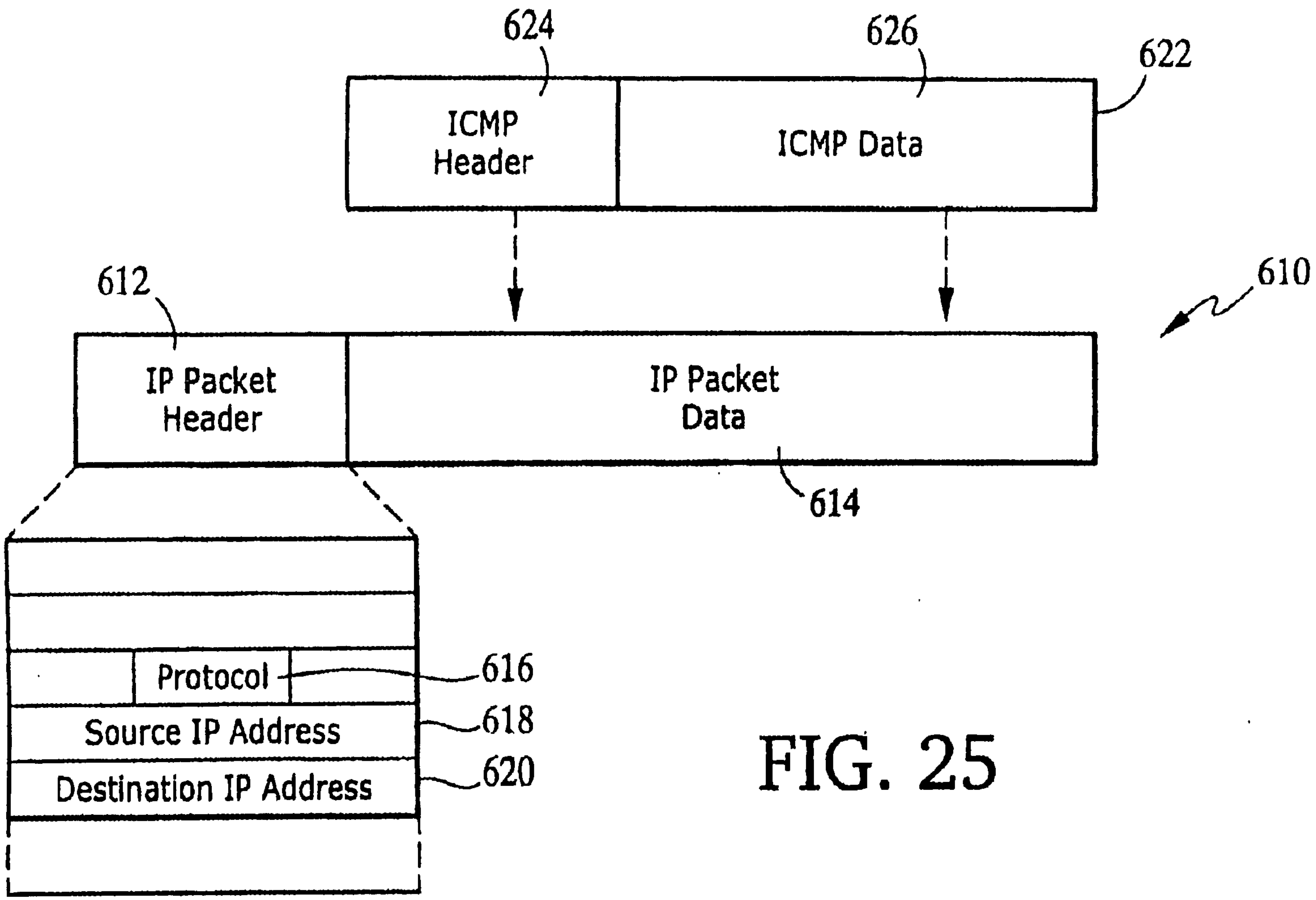

624
626
622
ICMP Header
ICMP Data
612
610
IP Packet Header
IP Packet Data
614
Protocol
616
Source IP Address
618
Destination IP Address
620

FIG. 25

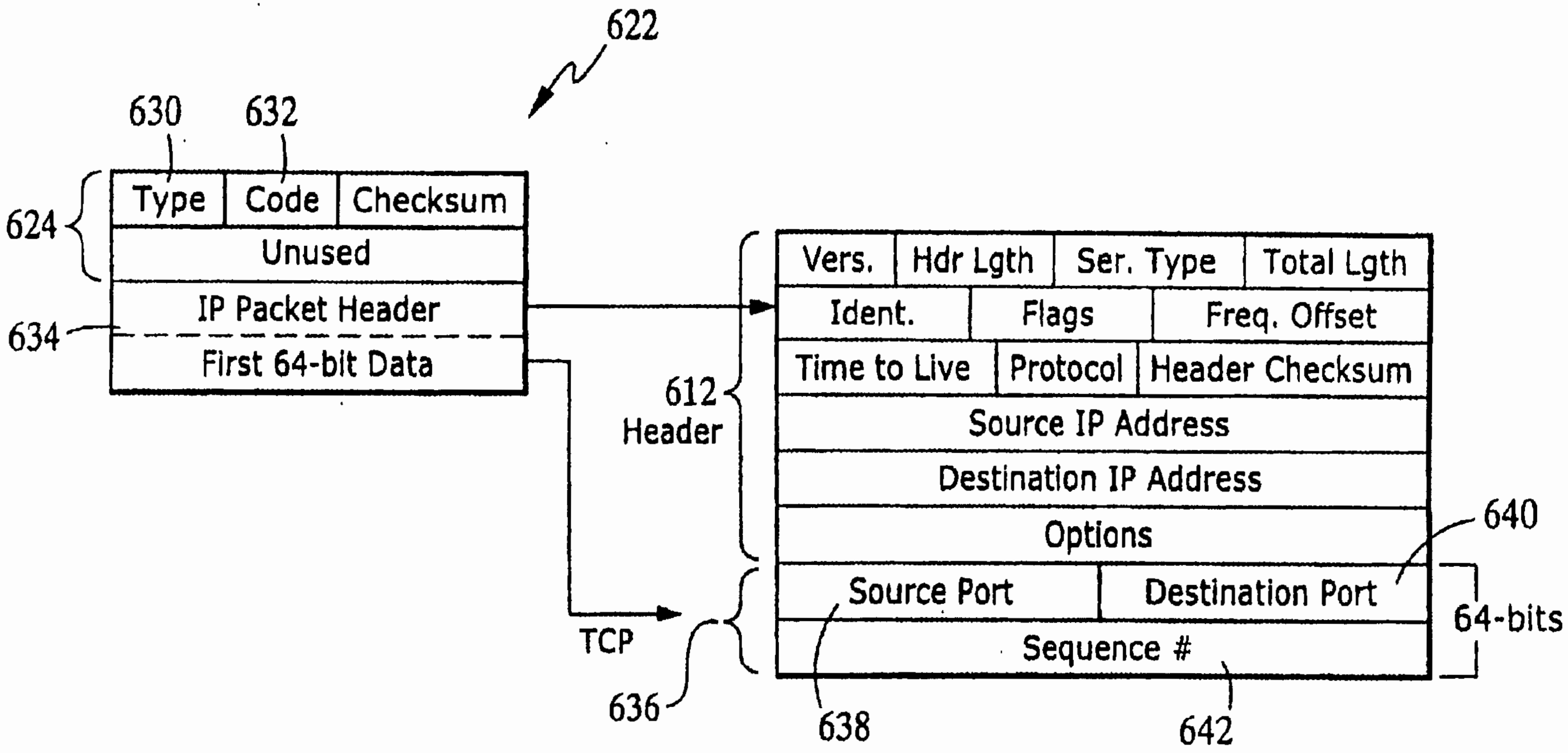

622
630
632
624
Type
Code
Checksum
Unused
IP Packet Header
634
First 64-bit Data
612
Header
Vers.
Hdr Lgth
Ser. Type
Total Lgth
Ident.
Flags
Freq. Offset
Time to Live
Protocol
Header Checksum
Source IP Address
Destination IP Address
Options
640
Source Port
Destination Port
64-bits
TCP
Sequence #
636
638
642

FIG. 26

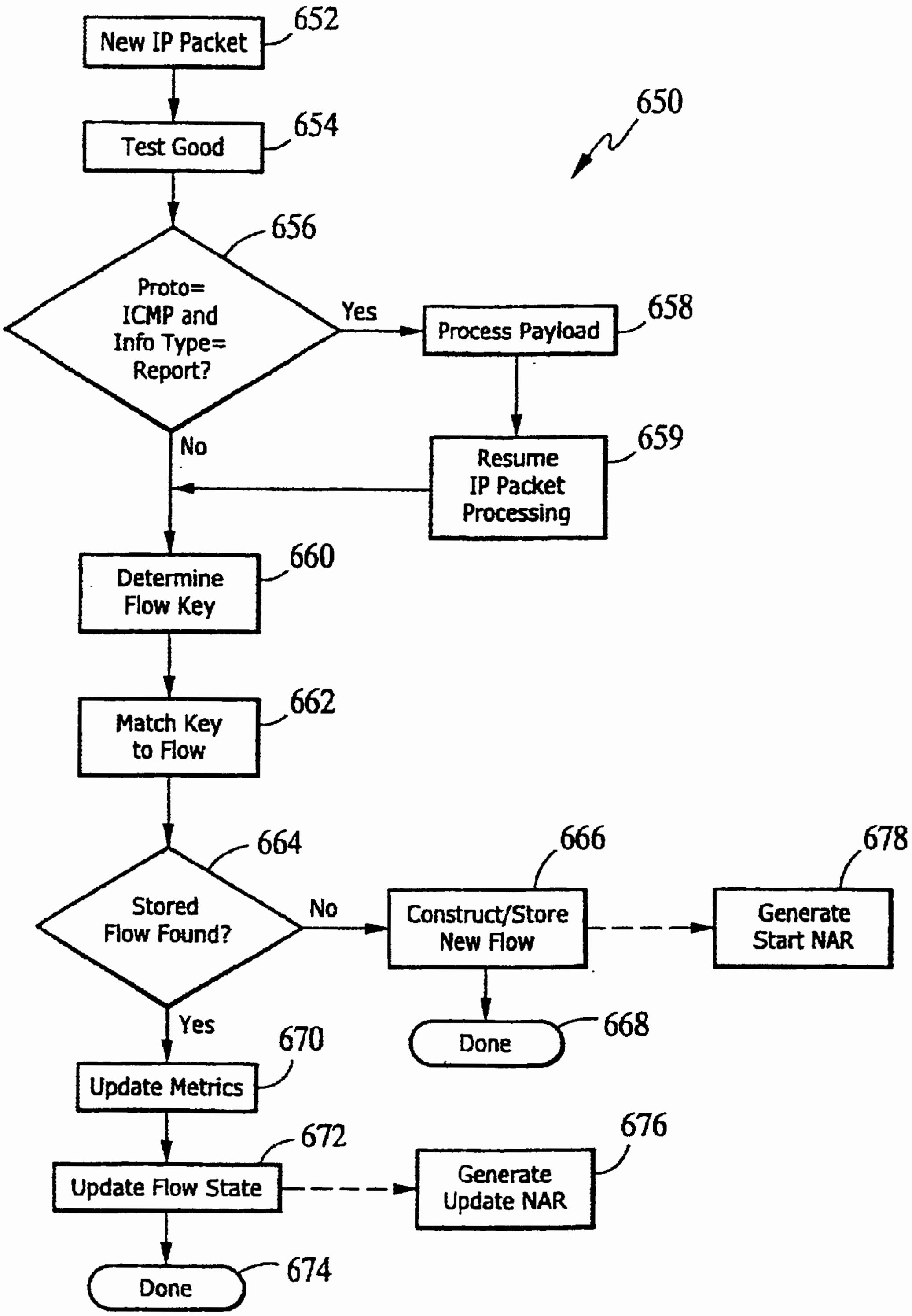

New IP Packet
652
Test Good
654
650
Proto= ICMP and Info Type= Report?
656
Yes
Process Payload
658
No
Resume IP Packet Processing
659
Determine Flow Key
660
Match Key to Flow
662
Stored Flow Found?
664
No
Construct/Store New Flow
666
Generate Start NAR
678
Done
668
Yes
Update Metrics
670
Update Flow State
672
Generate Update NAR
676
Done
674

FIG. 27

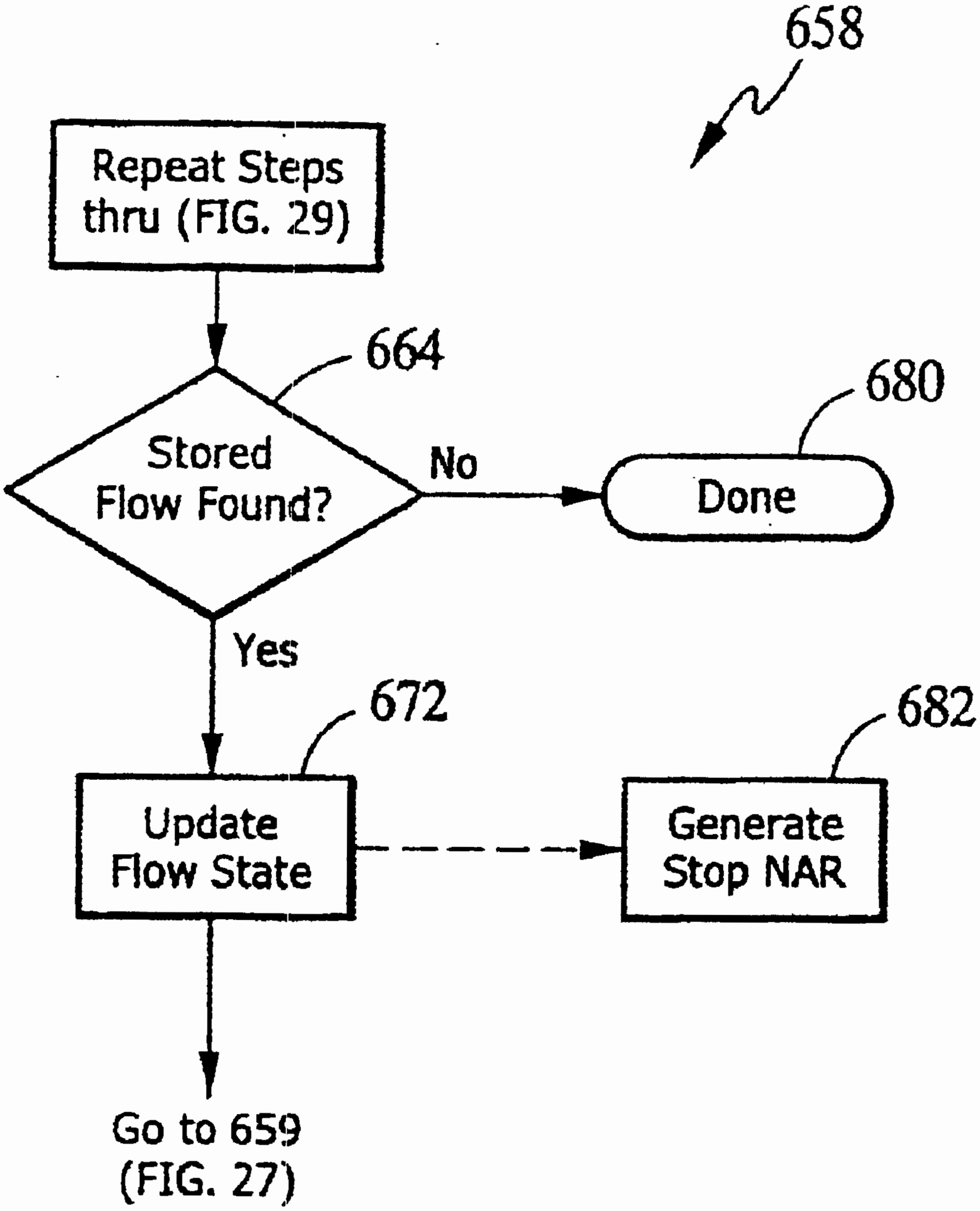

658
Repeat Steps thru (FIG. 29)
664
Stored Flow Found?
No
680
Done
Yes
672
Update Flow State
682
Generate Stop NAR
Go to 659 (FIG. 27)

FIG. 28

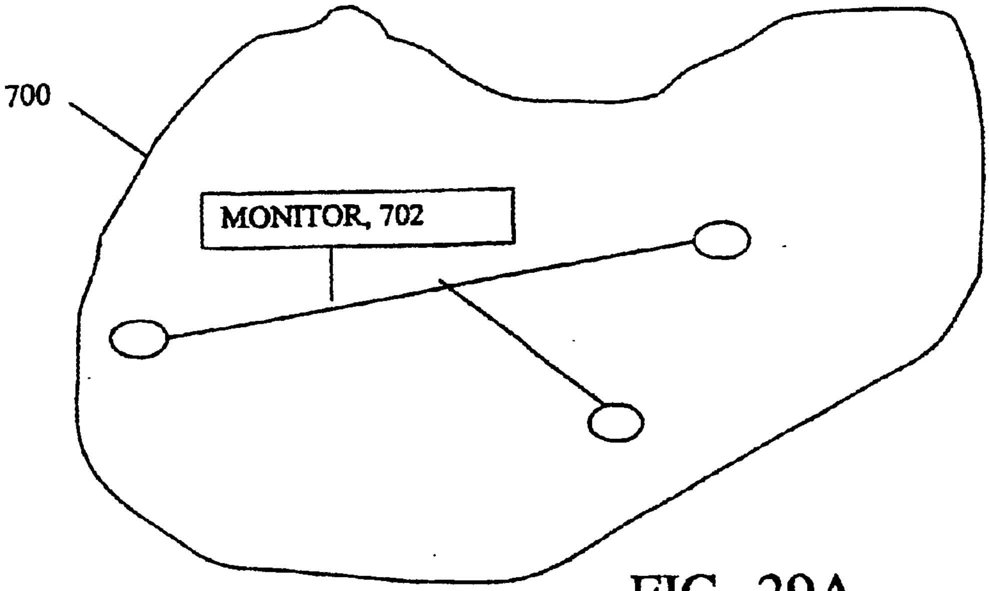


FIG. 29A

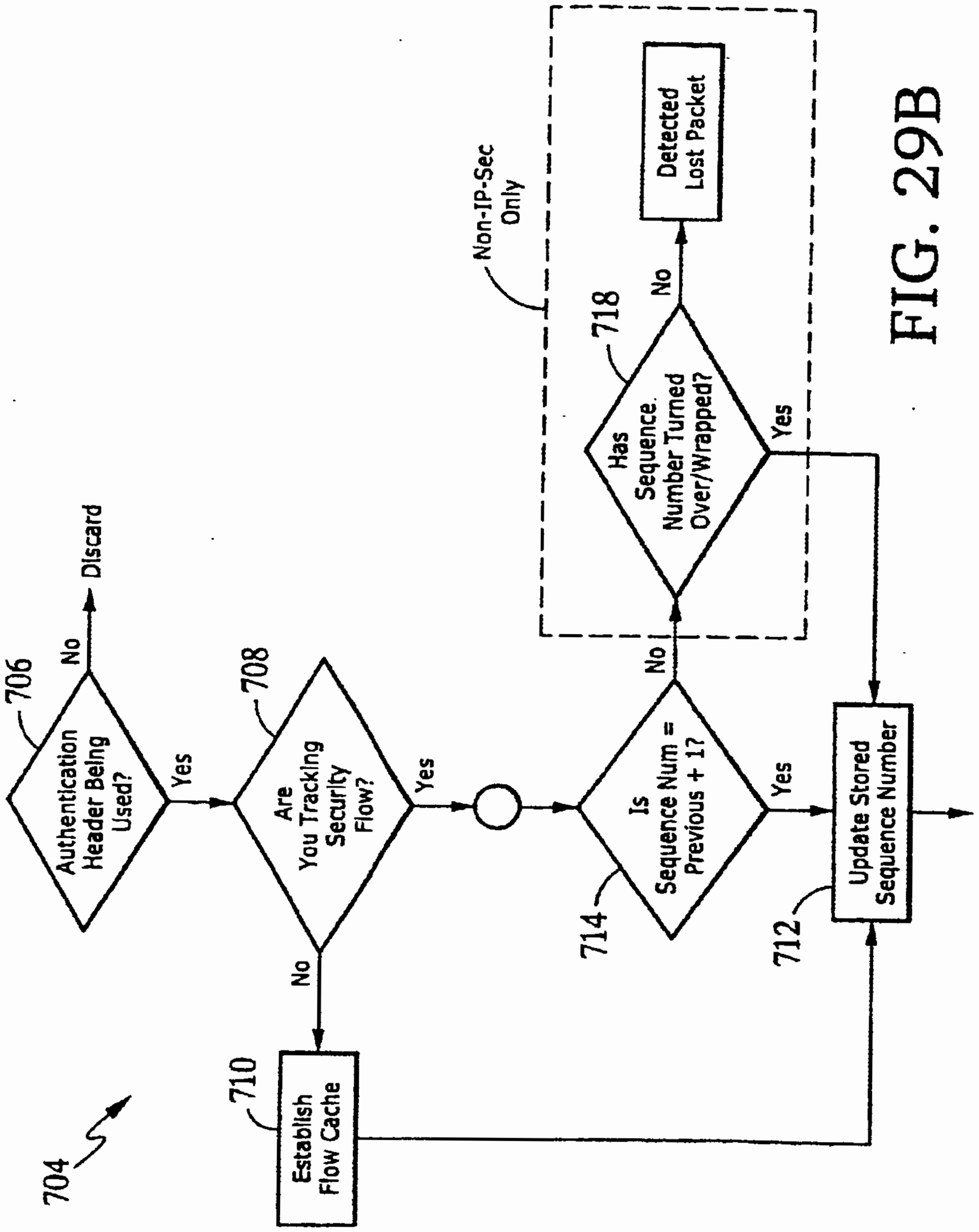

704
706
Authentication Header Being Used?
No
Discard
Yes
708
Are You Tracking Security Flow?
No
710
Establish Flow Cache
Yes
714
Is Sequence Num = Previous + 1?
Yes
712
Update Stored Sequence Number
No
Non-IP-Sec Only
718
Has Sequence Number Turned Over/Wrapped?
Yes
No
Detected Lost Packet

FIG. 29B

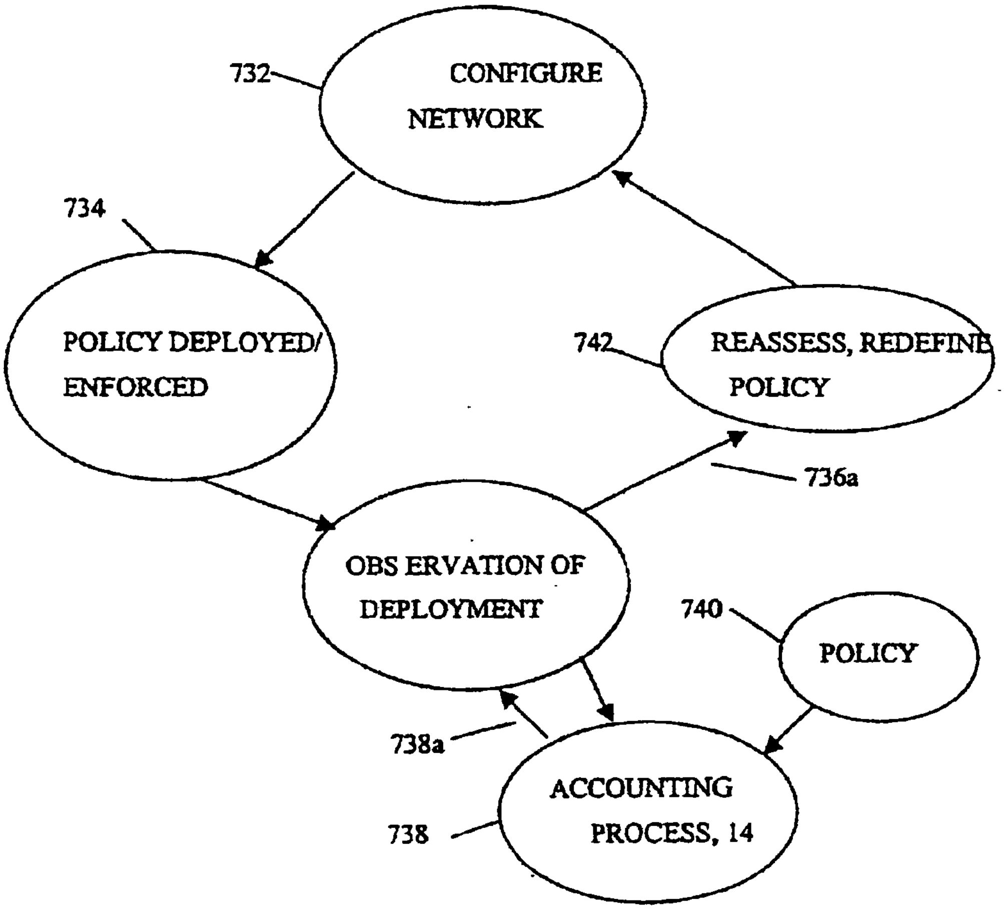

732
CONFIGURE NETWORK
734
POLICY DEPLOYED/ ENFORCED
742
REASSESS, REDEFINE POLICY
736a
OBS ERVATION OF DEPLOYMENT
740
POLICY
738a
738
ACCOUNTING PROCESS, 14

FIG. 30

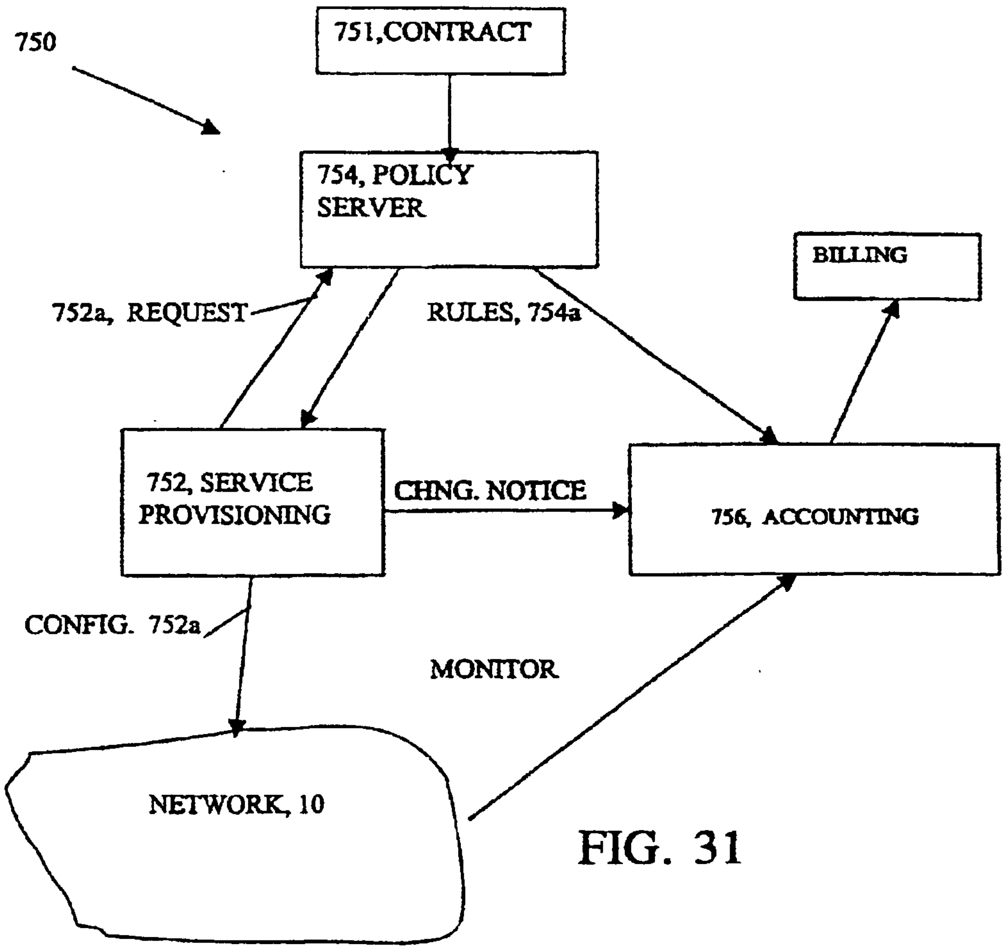

750
751,CONTRACT
754, POLICY SERVER
BILLING
752a, REQUEST
RULES, 754a
752, SERVICE PROVISIONING
CHNG. NOTICE
756, ACCOUNTING
CONFIG. 752a
MONITOR
NETWORK, 10

FIG. 31

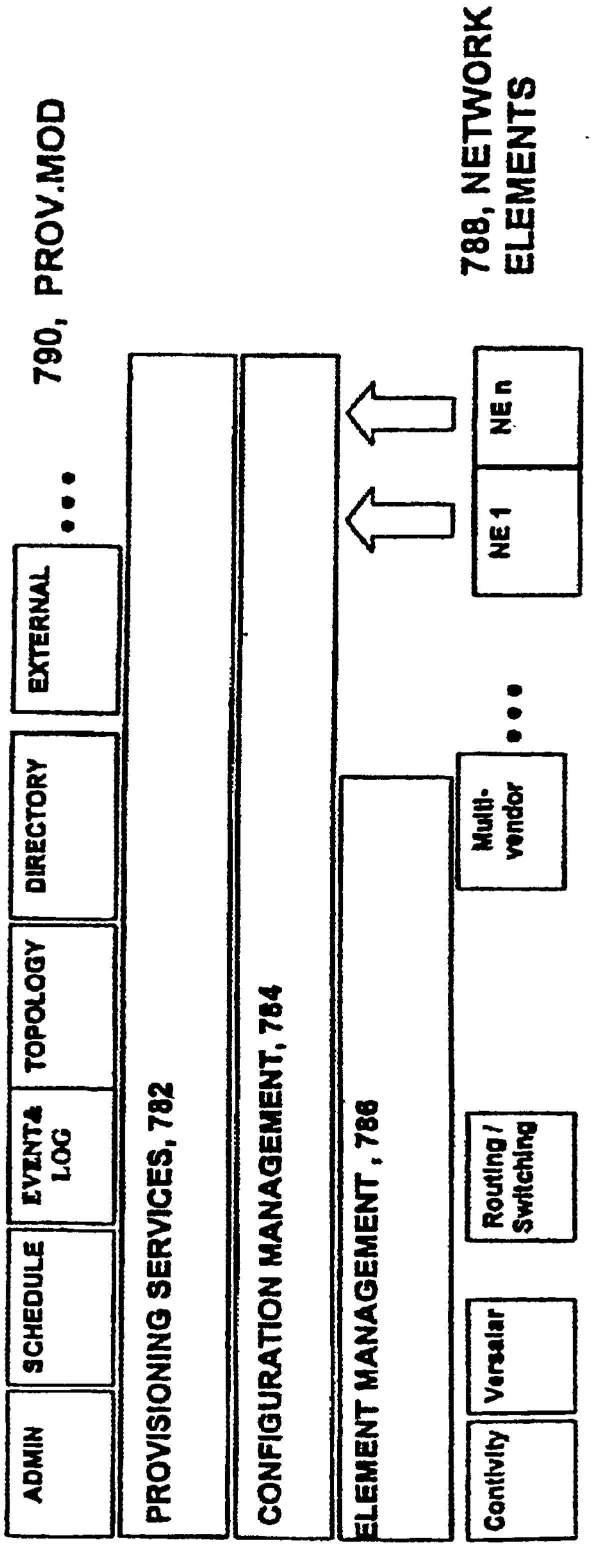

ADMIN
SCHEDULE
EVENT& LOG
TOPOLOGY
DIRECTORY
EXTERNAL
790, PROV.MOD
PROVISIONING SERVICES, 782
CONFIGURATION MANAGEMENT, 784
ELEMENT MANAGEMENT , 786
Contivity
Versalar
Routing / Switching
Multi-vendor
NE 1
NE n
788, NETWORK ELEMENTS

FIG. 32

SYSTEM FOR REQUESTING MISSING NETWORK ACCOUNTING RECORDS IF THERE IS A BREAK IN SEQUENCE NUMBERS WHILE THE RECORDS ARE TRANSMITTING FROM A SOURCE DEVICE

BACKGROUND

This invention relates to information management for Internet protocol (IP) packet transmission.

Data collection systems are used to collect information from network traffic flow on a network. These data collection systems are designed to capture one type of network traffic from one source type and deliver the data to one application type such as a billing application.

SUMMARY

According to an aspect of the invention, a method for tracking network accounting records in an accounting process that collects and correlates information derived from network data includes producing a network accounting record that has an identifier that uniquely identifies the record within the accounting process with the identifier including a sequence number that specifies a sequence number for network accounting records that originate from the source device and determining when there is a break in the sequence numbers of network accounting records produced from the device. The method also includes requesting missing network accounting records when there is a break in the sequence.

One or more of the following advantage may be provided by one or more aspects of the invention.

The records produced in the accounting system have a sequence number that allows components that are in the next level to detect if there are missing records in a collection of records and can be used to give a sense of how often records are produced in a given time period. With this information being part of every record, an accounting process can determine a sense of the functional capabilities of the intermediate components and detect some aspects of the communication channel between components.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a server running an accounting application monitoring a network.

FIG. 2 is an architectural block diagram of the accounting application used in FIG. 1.

FIG. 3 is a block diagram of accounting support in an access process used by an Internet/Intranet service provider or a large enterprise.

FIG. 4 is a block diagram of accounting support in an access process used by an Internet/Intranet service provider or a large enterprise using an Extranet switch.

FIG. 5 is graph depiction of a network including data collectors disposed in the network.

FIG. 6 is a flow diagram showing a typical data flow process using an accounting process.

FIG. 7 is a diagram show exemplary network accounting records.

FIGS. 8A–8B, 9A–9B, 10, 11A–11E, 12 and 13A–13B, are schematic views of data structures used in network accounting records.

FIG. 14 is a block diagram of a flow data collector system.

FIG. 15 is a flow diagram of the flow data collection process of the flow data collector of FIG. 14.

FIG. 16 is a block diagram of the flow aggregation processor (FAP).

FIG. 17 is a flow diagram of the flow aggregation process performed by the FAP of FIG. 16.

FIGS. 18–20 are examples of the FAP enhancement and aggregation portions of the flow aggregation process shown in FIG. 17.

FIG. 21 is a hierarchical representation of an aggregation adjustment scheme for adjusting the aggregation activity at the levels of the flow aggregation processor and the data collectors.

FIG. 22 is an example of a configuration file update for aggregation (policy) adjustment.

FIG. 23 is a flow chart of an information management process.

FIG. 24 is a representation of a network communications path between two end stations in a network.

FIG. 25 is an illustration of an ICMP message encapsulated in an Internet Protocol (IP) packet and the formats of the ICMP message and the IP packet.

FIG. 26 is an illustration of the format of an ICMP error reporting message header and datagram prefix.

FIG. 27 is a flow probe IP packet processing mechanism.

FIG. 28 is the payload processing/protocol correlation of the IP packet processing mechanism of FIG. 26.

FIGS. 29A–29B are diagrams depicting a protocol independent, packet loss detection monitor.

FIG. 30 is a diagram depicting a process to capture quality of service.

FIG. 31 is a diagram of a service management process.

FIG. 32 is a diagram showing an architecture of a service provisioning application.

DETAILED DESCRIPTION

Architecture

Referring now to FIG. 1, an exemplary arrangement 10 for collecting information from a network is shown. The network includes various network devices 12. The network devices 12 can be disparate, i.e., different equipment types, operating under different protocols and formats. The network devices 12 are coupled to an accounting process 14 via an equipment interface 16.

The accounting process 14 includes a flow data collection layer 18 that runs as client processes with the equipment interfaces on or close to the network devices 12. Individual and multiple data collectors (not referenced) can be disposed at points of presence (POP) in a network 11. The accounting process 14 includes a flow aggregation and distribution process 17 that runs as a server process on a server 15. The accounting process 14 assembles the data into a format that can be used by billing or other user defined data consuming applications 20 that interface to the accounting process 14, through a data consuming application interface 22. Thus, the accounting process 14 collects via the data collector layer 18 multiple and diverse types of data from the network 11, normalizes the data into a consistent accounting record, and provides open interfaces to one or more applications, such as billing via the application interface 22.

The network devices 12, e.g., switches, routers, remote access concentrators, and so forth can produce data of various types and formats which are all handled in the accounting process 14. Examples of the network devices 12 include a router or switch 12*a*, cable or telephone modems 12*b*, a flow probe 12*c*, a remote access concentrator 12*d* an Extranet switch 12*e*, a directory naming service (DNS) server 12*f*, a RADIUS server 12*g* and web server 12*h*. One particular source of data, the flow probe 12*c* will be described below in conjunction with FIGS. 24–28. The network devices 12 can include a "Remote Authentication Dial-In User Service" (RADIUS) server 12*g* that produces RADIUS accounting records using an existing RADIUS accounting process (not shown). The accounting process 14 can interface to the existing RADIUS accounting process and can use existing RADIUS records without modifying the existing RADIUS accounting environment. RADIUS is a well-accepted standard in the industry and is used across a number of different types of technologies (dial-in, cable, DSL, VOIP, etc.), with the most prominent being dial-in access. So, by supporting RADIUS records the accounting process 14 provides the ability to fit into an existing network environment without modification.

The accounting process 14 enables users such as an Enterprise or an Internet Service Provider to maintain an existing accounting configuration. Information sources can include network traffic flow, RADIUS accounting data, RMON/RMON2 data, SNMP-based data, and other sources of network usage data. The accounting process 14 collects data via the data collector layer 16 from multiple disparate sources and produces new type of composite records. These new composite records results is new information which provides a source for network accounting, billing, management, capacity planning, and so forth.

The accounting process 14, as will be described in FIG. 2, has a core process that can handle data records from each of the equipment types above, as well as other equipment types, and can provide data to each of the plurality of user-defined data consuming applications. This accounting process 14 provides flexibility in choosing data consuming applications that use accounting data. Such applications can include billing, enterprise charge-back or cost allocations, capacity planning, trending, application monitoring, user profiling and so forth.

Accounting Architecture

Referring now to FIG. 2, the equipment interface layer 16 of the accounting process 14 includes various equipment interfaces 42*a*–42*i* which are, respectively, an interface 42*a* for the router/switch 12*a*, an interface 42*b* for the cable/modem head end 12*b*, and an interface 42*c* for the flow probe 12*c*. The equipment interface layer 16 also includes additional interfaces such as an interface 12*d* for a remote access concentrator 12*d*, an interface 12*e* for an Extranet switch 12*e*, an interface 42*f* for a DNS server 12*f*, and an interface 42*g* for a RADIUS server 12*g*. The equipment interface can have additional interfaces that can be specified, as new equipment is added. The interfaces 42*a*–42*g* can be developed by an interface toolkit 44. The interface toolkit 44 permits a user to construct a new equipment interface type to couple the accounting process 14 to a new equipment source type.

The accounting process 14 also includes a data collector layer 18. The data collector layer 18 is a distributed layer comprised of individual data collectors, e.g., 52*a*–52*g*. The data collector layer 18 collects data in the form of raw accounting information specific to the device type. The data collector collects data via the aforementioned equipment interfaces 42*a*–42*g*. The data collectors 52*a*–52*g* collect the data and convert data into normalized records herein referred to as Network Accounting Records (NARs). Each of the data collectors 52*a*–52*g*, as appropriate, forwards network accounting records (NARs) to a flow aggregation process 60.

The data collectors 52*a*–52*g* support several different collection models. For example, the data collectors 52*a*–52*g* can support a so-called "push model" in which a connected device initiates a transmission of data to the accounting process 14. The data collectors 52*a*–52*g* also can support a "pull model" in which the accounting process 14 initiates a connection to an equipment interface for the purpose of obtaining data. In addition, the data collectors 52*a*–52*g* can support an "event driven model" in which an event that occurs in either the equipment interface layer 16 or in the accounting process 14 initiates a transfer based on some threshold or criteria being met by the equipment layer 16 or accounting process 14 within which the event occurred.

The data collectors 52*a*–52*g* are distributed throughout the network. The data collectors 52*a*–52*g* are placed close to or within the network device that the collector is assigned to. That is, the data collector can be in-line or out-of-line relative to the device monitored. The data collectors 52*a*–52*g* can be anywhere. The data collectors 52*a*–52*g* can be completely uncoupled from the devices except for communication paths. As new network devices 12 are added to the accounting support arrangement 10, new data collectors are also deployed.

The accounting process 14 also includes a flow aggregation process 60 that is part of the aggregation and distribution process 17 (mentioned above). The flow aggregation process 60 is a central collection point for all network accounting records (NAR's) produced from various data collectors 52*a*–52*g* in the data collection layer 18. The flow aggregation process 60 receives NAR's from various data collectors 52*a*–52*g* and aggregates, i.e., summarizes related information from the received NARs across the accounting support arrangement 10. The aggregation layer 60 produces Summary NAR's i.e., enhanced and unique network accounting records. That is, the flow aggregation process aggregates the records across the network devices; whereas, individual data collectors 52*a*–52*g* can aggregate accounting records from individual data sources. Aggregation will be described below in FIGS. 14–23.

The accounting architecture also includes a data distributor layer 70 (part of the aggregation and distribution process 17). The data distribution layer 70 provides a flexible data distribution mediation between the flow aggregation process 60 and a user-defined application, via an application interface layer 22. Data distributor layer 70 presents information to the application interface layer 22, with a pre-defined format, protocol and schedule that is determined by requirements of a user application. The data distributor layer 70 can support push, pull and event driven data distribution models. The application interface layer 22, is comprised of individual application interfaces 82*a*–82*g* that are provided by the toolkit 44. The toolkit 44 as with the network device interfaces 42*a*–42*g* can be used to produce additional application interfaces 82.

Exemplary Configurations

Referring now to FIG. 3, the accounting process 14 can, in general, support any configuration. Exemplary configurations used by an Internet service provider 100, an Enterprise A that host its own remote access 110, and an Enterprise B that uses the Internet service provider 120, are shown.

As shown in FIG. 3, for the Internet service provider, data collectors 52*a*–52*d* are distributed at specific Points of Presence (POP), such as remote access concentrators 102 managed by the Internet service provider. The remote access concentrators allow, a mobile user 106 or an Internet user 107 with remote access to access an enterprise over the Internet, via the Internet service provider. In this example the Internet service provider arrangement 100 and the large

Enterprise arrangements **110** and **120** include servers **13**, **13'**, and **13"** that run accounting processes **14**, **14'** and **14"**. The accounting processes **14**, **14'** and **14"** each independently manage and collect information regarding network traffic usage.

The Internet service provider arrangement **100** includes the accounting server **13** that runs the accounting process **14**. The accounting process **14** includes a flow data collector layer **18** that gathers data from the service provider network **100**. The flow data collector layer **18** includes distributed, individual flow data collectors *52a–52d*. The distributed, flow data collectors *52a–52d* collect transaction specific details about a user's connection type and actual network usage. These data are converted into the NARs in the distributed, flow data collectors *52a–52d*, as mentioned above. The NARs are aggregated over the entire system by the flow aggregation layer **60** (FIG. 2).

Data is made available to the Internet service provider via the data distribution layer (FIG. 2) so that the Internet service provider can analyze data in order to differentiate service offerings to different users. The Internet service provider can evaluate and use such detailed accounting data collected from various sources to migrate from a single flat rate fee business model to more flexible charging. For example, analysis of the data can enable the Internet service provider to develop new service options that can take into consideration bandwidth usage, time of day, application usage and so forth. In addition, Internet service providers can offer discounts for web hits that may exist in an Internet service provider cache, thereby minimizing the need to look up an address for a requested site on the Internet and can provide free E-mail usage while charging for other types of applications such as file transfer protocol and web traffic.

The data can also be used by other applications such as network planning, security, auditing, simulation, flow profiling capacity planning and network design and so forth. Thus, the Internet service provider can independently monitor and evaluate network traffic caused by remote employees and mobile users, for example.

Similarly, other instances **14'**, **14"** of the accounting process can be used by enterprises, as also shown in FIG. **3**. For example, an enterprise may host its own remote access, as shown for Enterprise A and would include a server **13'** running an accounting process **14'**. An enterprise could use the Internet service provider as shown for Enterprise B, and still have a server **13"** running an accounting process **14"**. The accounting process **14'**, **14"** includes an associated data collector that is coupled to enterprise A and enterprise B local area networks or other network arrangement. In this model, the enterprises use data from the accounting process **14'**, **14"** for enterprise charge-back functions such as billing departments for Internet usage within the enterprise and so forth.

Different instances of the accounting process are used by both the Internet service provider and enterprise A and Enterprise B sites. The instances **14**, **14"**, **14"** of the accounting process are independent they do not need to exchange accounting data. Rather, they exist as separate, independent accounting domains.

Referring now to FIG. 4, a similar access configuration **100'**, as the configuration **100** (FIG. 3) can be used with an Extranet switch **122**. Extranet access allows remote users to dial into an Internet service provider (ISP) and reach a corporate or branch office via an ISP. The Extranet switch allows Internet users access to corporate databases, mail servers and file servers, for example. It is an extension of the Internet in combination with a corporate Intranet. In this configuration, the Extranet switch **122** can be owned and operated by an Internet service provider as shown with enterprise A, or it could, alternatively, be owned and operated by an enterprise, as shown with enterprise B. Users would access the corporate network of either enterprise A or enterprise B, via the Internet service provider with various types of tunneling protocols such as L2TP, L2F, PPTP or IPSec, and so forth. The accounting server **13** located at the service provider and also accounting servers **13'**, **13"** within enterprise A and enterprise B allow each the Internet service provider and each of enterprises A and B to run accounting process **14'**, **14"** on the servers **13'**, **13"** to monitor and collect network data.

Transaction Flow Model

Referring now to FIG. **5**, a graph **140** depiction of a very large scale network includes a device "A" **142** communicating with a device "B" **144**. The graph **140** includes nodes (not all numbered) that can represent routers, switches, flow probes, etc. that have interfaces (not shown) which maintain statistics on information passed through the interfaces. For example, a switch may have a number of Ethernet ports and a host could be connected to one of the ports and in communication with one of the interfaces to transfer information over the network. The interface would have counters that are used to track "packet's in, "packet's out", "bytes in, bytes out", and so forth.

In this case of the host connected to the port, or a router or some other device being connected to the port, there is no other connection that the host, router or other device is aware of other than the entire network. This is an example of a "connectless oriented" protocol. A data collector **52** can be disposed in the network in a path between the entities "A" and "B", such that the data collector **52** monitors some of the packets that comprise a flow between "A" and "B." As a single point monitor, the data collector has no concept that there are two ends communicating. The data collector **52** identifies these entities "A" and "B" in various NARs produced by the data collector **52**. At later stage in the processing, either in the data collector **52** or elsewhere in the accounting process **14** the NARs are correlated so that the NARs or some aggregated NAR produced by the data collector **52** or the rest of the accounting process **14** can be associated with the accountable entities "A" and "B" to thus identify a connection between entities "A" and The data collectors *52a–52g* (FIG. 2) develop a description of the connection. For a router, normally an address of the object that is connected to that interface might serve as an address. A switch has an IP address that can be used as the destination. The actual device that is connected to the switch or router can be used as an accountable object. Although the traffic is not destined for the router, the data collector can use those identifiers as keys to the endpoints "A", "B."

In many cases, the protocol can explicitly determine connections. For example, the TCP/IP protocol is explicitly a "connection oriented" protocol used in the Internet. When the data collector **52** needs to determine a connection, the data collector **52** can exploit the "connection oriented" nature of certain types of protocols such as the TCP/IP protocol. When the data collector **52** tracks a TCP/IP connection, the data collector **52** can determine exactly that A and B are connected, when the connection starts, stops, and updates. With other protocols such as a "connectionless" protocol, and even in some complex environments such as a virtual private network or a proxy server, the data collector **52** does not necessarily know the real endpoints. The data collector **52** only knows that some entity is talking to some other entity.

Thus, the data collector 52 is a single point monitor, that monitors traffic at one point in the network and converts the traffic into a "pipe oriented" or "flow oriented" accounting information. The data collector 52 identifies a source and a destination of the traffic. That is, the data collector develops a "connection oriented tracking." By distributing data collectors 52*a*–52*g* (FIG. 2) through out the network the network can be modeled as pipes having two endpoints. A data collector can be disposed in a partial pipe. The data collector 52 determines that one end of the pipe refers to "A" and the end of the pipe refers to "B." The data collector 52 can be disposed anywhere along the network.

The graph 140 represents the network as a directed graph, including partial segments. The endpoints of those partial segments can act as proxy entities to the actual accountable objects. Once independent accounting records that relate to these two entities A and B are aggregated in the accounting process 14, the accounting process 14 can identify that A and B are connected and have particular metrics.

Some equipment have a half pipe model that generate independent accounting records for each half pipe. The data collectors can assemble full pipe information from half pipe information. The accounting process could be coupled to equipment that gives a half pipe model for A communicating with B and a separate one for B communicating with A. The data collectors 52*a*–52*g* combine information from these two half pipes into a bidirectional flow.

Referring now to FIG. 6, an example of data flow 130 through the accounting process 14 is shown. In this example, data flow is initiated by a user 131 making a call to a remote access concentrator (RAC) 132. Upon receiving the call, the RAC 132 authenticates the user against a secure access controller 134. After verification, the RAC 132 connects the user to the network 135 and sends a RADIUS Start record (not shown) to the accounting process 14. The accounting process 14 generates a RADIUS Start NAR 137*a* and stores the RADIUS start NAR in a database 62. At that point, the remote user may check e-mail, look at a web server and transfer a file. For each transaction, the accounting process 14 captures the IP traffic, generating a e-mail, http, and ftp network accounting records 137*b*–137*d*, respectively. These are stored in the database 62. Upon completion of these transactions the user would log out of the network, at which time the RAC would send the accounting process 14 a RADIUS Stop record. The accounting process 14 generates a RADIUS Stop NAR 137*e* and stores the RADIUS stop NAR in the database 62. All of these records reflecting the user's transactions could be viewed and reported in flexible ways dependent on the needs of an end-user application.

Network Accounting Records (NARs)

The data collector 52 translates collected information into network accounting records (NARs). A NAR includes a header, an accounting entity identifier and metrics. The network accounting record (NAR) is a normalized data record that contains information about a user's network usage activity.

Referring now to FIG. 7 a base level "activity" NAR that includes source, destination, protocol, source port, destination port, byte and packet counts, etc. The base level activity NAR can be combined and aggregated in many different and flexible ways to support various accounting functions. The NAR is an activity record corresponding to some level of detail. Detail can vary based on the level of aggregation being applied, in accordance with the needs of the end-user/application.

FIG. 7 has at one level 152 a plurality of exclusively "Activity NARs" which could correspond to a very low level of detail, or could be the result of a prior aggregation providing a higher level view of the information. Thus, FIG. 7 shows a collection 152 of exclusively activity NARs. From base level data, additional "views" of the NAR could be produced, such as a set of "Summary NARs" 154, or another set of Activity NARs 156 which could be a result of further aggregation of the base level information, or lastly a combination of a set of Summary NARs and Activity NARs 158. The summary NAR is produced by the central aggregation layer 60 and can include user identifying information, protocol information, connection time information, and data information, and so forth.

The specifics of what can be combined and aggregated will described below. Thus, the accounting process 14 use of NARs provides the ability to combine and aggregate base level activity information in a flexible way to meet the specific needs of the end-user/application.

TABLE 1 below corresponds to the fields that can be captured in a NAR. This is essentially the activity NAR. The NAR contains these fields, which can then be combined and used to form other activity NARs or summary NARs.

Column Name	Description
OSA_SOURCE_TYPE	List all of the possible data sources from which data can be collected. Reference to OSA_SOURCE_TYPE TABLE.
OSA_SOURCE_SERIAL_NUM	Number which uniquely identifies an OSA FDC.
START_TIME_SEC	Indicates the date and time at which a record was recorded.
START_TIME_USEC	Microseconds component of START_TIME_SEC.
SEQUENCE_NUMBER	Sequence number assigned by the source of the NAR to uniquely identify the record and ensure integrity.
USER_NAME	The user associated with the record.
EVENT	Event type of the record (i.e. Start, Stop, Update).
SESSION_ID	Unique Accounting ID to make it easy to match start and stop records.
SESSION_TIME	Duration of the record in seconds.
SESSION_TIME_USEC	Microseconds component of the SESSION_TIME.
SRC_ADDR	Source address of the record.
DST_ADDR	Destination address of the record
PROTO	Protocol of the record. Reference to OSA_PROTOCOL_TYPE table.
SRC_PORT	Source port number.
DST_PORT	Destination port number.
SRC_OCTETS	Number of bytes transmitted into the network by the source. For RADIUS is equivalent to Acct-Input-Octets.
DST_OCTETS	Number of bytes sent out of the network, to the source. For RADIUS is equivalent to Acct-Output-Octets.
SRC_PKTS	Number of packets transmitted into the network by the source. For RADIUS is equivalent to Acct-Input-Packets.
DST_PKTS	Number of packets transmitted out of the network, to the source. For RADIUS is equivalent to Acct-Output-Packets.
SRC_TOS	The Type of Service coding marked by the source.
DST_TOS	The Type of Service coding marked by the destination.
SRC_TTL	The Time To Live field set by the source and modified by the network until the Nortel flow probe recorded it.
DST_TTL	The Time To Live field set by the destination and modified by the network until the Nortel flow probe recorded it.

-continued

Column Name	Description
OSA_CAUSE	A number that indicates the reason the accounting record was generated.
OSA_STATUS	A value used to indicate the status of an accounting record when it was created.
ACCT_DELAY_TIME	Indicates how many seconds the client has been trying to send this record
ACCT_AUTHENTIC	Indicates how the user was authenticated.
ACCT_TERMINATE_CAUSE	Indicates how the session was terminated
ACCT_MULTI_SESSION_ID	Unique Accounting ID to make it easy to link
ACCT_LINK_COUNT	Indicates the count of links which are known to have been in a given multilink session at the time the accounting record is generated.

The summary NAR and activity NAR have a one-to-many relationship. That is, while there can be a single summary NAR for a particular user over a particular call that will contain information about the sum of usage of network resources over the duration of the call, there can be many activity NARs. The activity NARs capture details about the actual activity and applications being used during the call. The summary NAR, therefore, depicts the total expense of the transaction or a set of transactions on a network, whereas, the activity NARs depict expenses of a transaction at any point in time. The summary NAR is generated in the flow aggregation process **60**, as will be described below. In essence, the summary NAR is generated from individual activity NARs correlated in the data collectors **52***a*–**52***g*, as will be described below.

A NAR is a member of a generic data message set that is used to transport data, such as network accounting data, through the accounting process **14**. These system data messages include "Status Event", "Maintenance Event", "Trace Event", "Network Accounting Record". Accounting process **14** messages share a common MSG_HDR structure that is used to discriminate between the four types of accounting process **14** messages. The Message Header (MSG_HDR) includes Message Type, an Message Event and Cause, and Message Length.

Network Accounting Record Data Structures

As will be described below, the NAR is unique within the accounting process **14**. The NAR has a NAR_ID that specifies an accounting process component ID. The component ID specifies the data collector assigned to a particular network device that produced the NAR. The component ID e.g., NAR_SRC_ID **203***a* (FIG. **8B** below) is allocated at the time that the component is produced. The NAR_ID also includes a time stamp at the second and microsecond level so that the accounting process **14** can discriminate between multiple NARs generated by a particular component. A sequence number, e.g., 32 bits is also used to differentiate NARs from the same accounting process component that may have the same time stamp. The sequence number e.g., NAR_SEQ_NUM **203***c* (FIG. **8B**) is preferably a monotonically increasing number starting from, e.g., 1. As long as the component is functioning, and producing NARs, the component provides a new sequence number to a new NAR.

Referring now to FIGS. **8A**–**8C**, a Network Accounting Record (NAR) data structure **200** is shown.

As shown in FIG. **8A**, the NAR data structure **200** includes two basic accounting objects, a Network Accounting Record Identifier **202**, and optionally one or as shown a plurality of Network Accounting Record Attributes **204***a*–**204***n*, generally denoted as **204**. The Network Accounting Record Identifier **202** has a set of object identifiers that uniquely identifies the network accounting record within the accounting process **14**.

The Network Accounting Record Identifier **202** acts as a database key value that makes the NAR **200** unique within the entire accounting process **14**. The Network Accounting Record Identifier **202** allows the NARs to be handled and managed using database functions such as database integrity analysis and reliability analysis. The Network Accounting Record Identifier **202** also gives the accounting process **14** the ability to track the source of NARs and to build mechanisms such that the accounting process **14** can maintain identity of the origination of NARs throughout the system **10**.

The plurality of Network Accounting Record Attributes **204***a*–**204***n* provide metrics for the NAR **200**. The Network Accounting Record Attributes **204***a*–**204***n* capture specific information contained in data from network devices. Differentiating between the entity identifier **202** and the metric **204** allows the accounting process **14** to perform logical and arithmetical operations on metrics **204** while leaving the accounting identifier intact **202**. The accounting identifier **202** can be enhanced unlike the metrics.

The data collectors **52***a*–**52***g* (FIG. **2**) are oriented around the process of filling in the NAR. The metrics are left untouched by the data collector and are passed transparently into the accounting process flow aggregation process **60**. The data collectors **52***a*–**52***g* assign the accounting entity identifiers **202** to the metrics e.g., a source and a destination identifier to the metric. In the example of a router link, the metrics that the router interface provides are in the form of "information in" and "information out" e.g., octets in, octets out, bytes in, bytes out datagrams in, datagrams out, faults in, faults out, and so forth. The data collectors **52***a*–**52***g* determine what "in" and "out" means and assigns the unique identifier that is unambiguous relative to the determined meaning of "in" and "out." Once a data collector **52** has established this convention, the convention is used throughout the system **10**.

Thus, the NAR Identifier **202** provides database constructs to a NAR, whereas, the plurality of Network Accounting Record Attributes **204***a*–**204***n* provide the actual metrics used for network activity reporting and network accounting.

As shown in FIG. **8B**, the Network Accounting Record Identifier **202** (NAR_ID) is a set of objects within the NAR that uniquely identifies the NAR throughout the accounting process **14**. The NAR_ID **202** is designed to support a number of properties of a NAR including flexibility, accountability, reliability and uniqueness. In order to provide these properties, the NAR_ID **202** is divided into objects designed to specifically provide these properties. Flexibility is supported through a NAR_HDR **203** section of the NAR_ID. Accountability is attained in the NAR through explicit identification of the source of the NAR by a component identification NAR_SRC_ID **203***a*. The source time is maintained in a NAR_SRC_TIME **203***b*. Reliability is supported, as described above, through the use of a NAR sequence number (NAR_SEQ_NUM) **203***c*, which is designed to enable traditional database integrity mechanisms.

The NAR_ID **202** is used to provide uniqueness for each NAR. The responsibility for guaranteeing the uniqueness of each NAR is handled by every accounting process component that has the ability to originate/source network accounting records. This responsibility requires that each accounting process component have the ability to unambiguously identify itself in each NAR that it produces. Thus, NAR type identifier, NAR_TYPE, is comprised of the source component identifier, NAR_SRC_ID, the NAR source time, NAR_SRC_TIME, and the NAR sequence number, NAR_SEQ_NUM. These three data objects act as a database key for a particular network activity record, ensuring the uniqueness of the NAR throughout the entire system.

The NAR_SEQ_NUM can have several purposes. One way that the NAR_SEQ_NUM can be used is as a discriminator when two NARs are produced at the same time. A second way that the NAR_SEQ_NUM is used is as a monotonically increasing index to ensure database integrity. Because the NAR_ID is unique, it should be considered as an allocated value. A NAR_ID is allocated at NAR origination.

If a component creates or modifies the contents of an existing NAR, as for example when aggregating two NARs together, the component originates the NAR_ID. This provides an opportunity for the accounting process 14 to have explicit internal integrity mechanisms that can account for any network accounting record that is processed by the accounting process 14.

The NAR Source Identifier NAR_SRC_ID **203*a*** includes a source type **207*a*** and a Source Serial Number **207*b***. The serial number **207*b*** is an administratively allocated value e.g., 24-bits that uniquely identifies the NAR source type throughout the accounting process **14**. The source serial number **207*b*** should be unique within the specific accounting domain.

The (NAR_SEQ_NUM) **203*c*** is a monotonically increasing, e.g., unsigned 32-bit integer that acts as a sequence number for NARs that originate from a particular NAR source. Because the value of the NAR_SEQ_NUM can "wrap around", the combined 64-bit value NAR_SRC_ID and NAR_SEQ_NUM are unique only over a specified time period.

Referring now to FIGS. **9A–9B**, exemplary formats for Network Accounting Record Attributes **204** are shown. There are two variations on a single NAR_ATTRIBUTE format that can be used. As shown in FIG. **9A**, a standard NAR_ATTRIBUTE format **206*a*** includes header fields NAR_ATTR type, NAR_ATTR Code, NAR_ATTR Qualifier, and NAR_ATTR Length and a "value field." In order to conserve the size of accounting information, when the size of the value of the NAR_ATTRIBUTE is a byte i.e., 8-bits, as indicated in the NAR-ATTR Qualifier field, the format **206*b*** of the NAR_ATTRIBUTE can be as shown in FIG. **9B**, including fields NAR_ATTR type, NAR_ATTR Code and an 8-bit NAR_value field.

Each supported object is assigned an NAR_ATTR Code. Through the NAR_ATTR Code, the accounting process **14** can distinguish the semantics of a particular NAR ATTRIBUTE. Although NAR_ATTR Codes are specific to the NAR_ATTR Type, the NAR_ATTR Code assignments can be unique to aid in implementation. Values can be assigned to provide some explicit hierarchical structure. Each NAR_ATTR has an 8-bit NAR_ATTR Qualifier that provides typing information for the NAR_ATTR. The NAR_ATTR Qualifier is used because some supported objects can be represented using several different types. Counters, for instance can be 32-bit as well as 64-bit, in the case of aggregated objects. Network identifiers may use numeric indexes, or strings as labels. The NAR_ATTR field specifies the length of the NAR attribute including the NAR_ATTR header.

There are five types of Network Accounting Record Attributes that are supported in the NAR. The five attributes are Accounting Time Interval (ACCT_TIME) (FIG. **10**); Accounting Entity Identifier (ACCT_ENTITY_ID), (FIGS. **11A–11E**); Accountable Entity Descriptor (ACCT_ENTITY_Desc); Network Activity Metrics (NET_METRICS)(FIG. **12**); and two Transparent Attributes (TRANS_ATTR)(FIGS. **13A–13B**). As necessary, additional NAR_ATTRIBUTES can be supported. For example, a NAR_ATTRIBUTE type could also include Security Attributes for accounting data to protect against unauthorized introduction or modification of accounting information.

Referring now to FIG. **10**, an Accounting Time Interval record includes a value "seconds" and a value "micro second". The values of "seconds" and "micro seconds" together represent a time stamp of network activity for the NAR, as discussed above. When derived from an absolute time value that represents the end of the accounting time interval, the Accounting Time Interval is the duration, as calculated using the Accounting Time Interval as the starting time value. All Network Accounting Records can have an Accounting Time Interval attribute.

Referring now to FIGS. **11A–11E**, Accountable Entity Identifier data structures are shown. The Accountable Entity Identifiers are a collection of entity description attributes that together identify an accountable entity in the accounting process **14**. The accounting entity identification mechanism facilitates flexible NAR aggregation properties of the accounting process **14**. The ACCT_ENTITY_ID is the description of an accounting object within the accounting process **14**. There can be one or more ACCT_ENTITY_IDs in a given NAR, but there must be at least one ACCT_ENTITY_ID in an Network Accounting Record. The actual accountable object is defined by the entire collection of ACCT_ENTITY_IDs that are included in the NAR.

In transaction based accounting, a network accounting record will contain two ACCT_ENTITY_IDs, representing the source and the destination entities that are involved in the network transaction. For traditional flow based accounting, these would normally be the two network addresses that are involved in the flow. Qualifiers are available in the ACCT_ENTITY_ID objects to indicate which ID is the source and which is the destination of the network transaction.

In direct support of flow based accounting data sources, the accounting process **14** supports a specific IP flow descriptor. This is the traditional IP 5-tuple flow description. The accounting process **14** could also support a 6-tuple flow descriptor that includes a type of service (TOS) indicator in the flow designator. This allows for Class of Service distinction in the accounting model.

For network activity data sources that do not have a transaction accounting model, there may only be a single ACCT_ENTITY_ID present in the accounting record. Qualifiers for the ACCT_ENTITY_ID are available to indicate if the single object is the source, destination, or both, for the accounting metrics that will be included. The types of entities include User Identifiers and Network Entity Identifiers. The network identifiers can include IP Address, Flow Description, and Network Object ID. Other types of accounting entities can be provided.

The actual accountable entities for a specific network accounting record are specified in the complete set of ACCT_ENTITY_ID(s) that are present in the NAR. Operations that can be applied to NARs, specifically aggregation, can influence how ACCT_ENTITY_IDs are used in NARs. Each accountable entity identifier that is present adds refinement to the definition of what accountable entity the metrics actually apply to, whereas each ACCT_ ENTITY_DESC further refines the description of the accountable entity.

Referring now to FIG. 11A, a NAR_USERNAME is a specific type of NAR_USERID data structure. A system string type "Username" 222 can represents a real accountable user within the accounting process 14. The NAR_ USERNAME data structure 220 is used to transmit the string. The semantics can be applied when the string "Username" 222 is supplied by RADIUS or from DCHP management systems. The NAR_USERNAME data structure 220 includes a NAR_USERNAME NAR_ATTR Qualifier that provides for Role designation, indicating whether the object referenced is acting as a source, destination, both or undeterminable within the system. The NAR_ATTR Qualifier bits are set when the Role can be determined without ambiguity.

Referring now to FIG. 11B, a NAR_USER_ID data structure 230 is the general type for identifying an accountable user. The accounting process 14 can use any available object type to represent the NAR_USER_ID value 232. The NAR_USER_ID value 232 will be a system established STRING type or a user index as generally supplied by a database system. The semantics of the NAR_USER_ID value 232 are consistent within the accounting process 14, and can be consistent outside of the accounting process 14.

Referring now to FIG. 1C, a NAR_IP_ADDRESS data structure 240 is shown and which is the general network component identifier for an IP enterprise network. NAR_ IP_ADDRESS data structure 240 includes a IPAddress 242 that is usually unique within the accountable domain, and thus can be usable as an accounting process 14 identifier. Within the accounting process 14, the occurrence of this record implies that the address is unique within the accounting realm. NAR_IP_ADDRESS type includes a NAR_ IP_ADDRESS NAR_ATTR Qualifier. The NAR_IP_ ADDRESS NAR_ATTR Qualifier provides for Role designation, indicating whether the object referenced is acting as a source, destination, both or undeterminable within the system. These bits are set when the Role can be determined without ambiguity.

Referring now to FIG. 11D, a NAR_NETWORK_ID type data structure 250 is shown. The NAR_NETWORK_ ID data structure 250 includes a NETWORK_ID value 252 is a general type used for identifying a network component when a network address is inappropriate. The accounting process 14 can use any available object type to represent the NAR_NETWORK_ID, but it is assumed that this value will be an accounting process 14 established STRING type, (e.g., a Media Access Control (MAC) address that is predefined in Network interface cards), object type or a number index that cannot be associated with a network address. The semantics of the NAR_NETWORK_ID is consistent within the accounting process 14, and can be consistent outside the accounting process 14. A NAR_NETWORK_ ID NAR_ATTR Qualifier provides for Role designation, indicating whether the object referenced is acting as a source, destination, both or undeterminable within the system. These bits are set when the Role can be determined without ambiguity.

Referring now to FIG. IE, a NAR_FLOW_DESC data structure 260 is the general type for reporting on flow based network activity. The NAR_FLOW_DESC is a composite data structure 260 including a IP Source Address 262, IP Destination Address 263, Transport Protocol 264, Type of Service 265, Source Port 266 and Destination Port 267 that are populated from transport and network layer of IP packets via flow probe. The NAR_FLOW_DESC NAR_ATTR Qualifier provides for Role designation, indicating whether the object referenced is acting as a source, destination, both or undeterminable within the system. These bits are set when the Role can be determined without ambiguity.

Therefore the Network Accounting Activity Records are fundamentally bindings between an accountable entity and a set of metrics that can be associated with that entity over a specified period of time. The NARs provide flexibility in defining, or specifying, the accountable entity. This level of flexibility is required because in network accounting, an accountable entity could potentially refer to objects that are either physical or logical, singular or members of collections, or geographically or topologically constrained, such as network numbers or autonomous system numbers.

A set of accountable entities includes Username and Network Object Identifiers. There can be additional descriptive information available within network activity reports and within networking components that could be used to further describe accountable entities. These entity attribute descriptors can be used in the accounting process 14 to provide additional flexibility in how network activity information is reported and tallied. Support for entity descriptions can include object support for:

- Flow Descriptors
  - Flow Protocol Descriptors
  - Flow Transport Port Descriptors
- Authentication Descriptors
  - NAS Descriptors
- Aggregate Descriptors
  - Class Identifiers
  - Session Identifiers
  - Multi-Session Identifiers
  - VLAN Identifiers
  - ELAN Identifiers
  - Group Identifiers
- Access Identifiers
  - Source and Destination Ethernet Addresses
  - Ingress and Egress Tunnel Ids
  - Ingress and Egress Port Numbers
    - ATM Virtual Circuit VPI/VCI
    - Calling and Called Station Ids
- Flow Status Descriptors
- Class of Service Identifiers
- Quality of Service Identifiers
- Traffic Path Identifiers
- Accounting Time Interval
- Accountable Network Activity Metrics
  - Source and Destination Datagrams
  - Source and Destination Octets
- Extended Network Activity Attributes
  - Network Flow Control Indications
  - Host Flow Control Indications
  - Traffic Burst Descriptors

Referring now to FIG. 12, a NET_METRIC data structure 270 is shown. A NET_METRIC data structure 270 to support a count is shown in FIG. 14. The NET_METRIC data structure 270 is used to hold basic accounting values that can be tallied within the accounting process 14. The NET_METRIC data structure 270 can support time, octets, datagram, counts and cells, circuits, tunnels and so forth.

Referring now to FIGS. 13A and 13B, two basic transparent objects TRANS_ATTR objects are shown; UNDEFINED 280 and RADIUS 290. New TRANS_ATTR object types can be defined as needed. These are objects that a user may want to send through the accounting process **14**, that are customer specific, or proprietary in nature. The accounting process **14** allows for object transparency, i.e., an object that the system does not act on or modify. Thus, the contents of transparent attributes are undefined with respect to the accounting system. They are passed through, unmodified.

Flow Data Collector

Referring to FIG. **14**, a flow data collector system **300** for supporting the flow data collector ("FDC") **52** (from FIG. **2**) is shown. The flow data collector system **300** includes a processor **302** coupled to a memory **304**. In this embodiment, the FDC is a process stored in the memory **304** and executed by the processor **302**. The FDC **52** includes several NAR processing components or processes. These processes include a NAR constructor **306** for converting data gathered by the equipment interface (EI) **16** (shown in dashed lines) from a network device or technology ("network entity")into NAR format. Recall that each equipment interface *42a–42g* is associated with an flow data collector. Thus, the combination of a equipment interface and a flow data collector support a particular device or technology and collects data from the particular device or technology using a pre-defined format, schedule and protocol specific to that device/technology. The NAR processes further include a correlator **308**, an enhancement process **310** and an aggregator **312** for processing the constructed NARs as appropriate. The details of these processes will be discussed further with reference to FIG. **15** below.

Still referring to FIG. **14**, the memory includes a local store **314** and a flow data collector configuration (file) **318**. The local store **314** stores data received from the equipment interface **16** and processed NARs. The configuration file **318** is provided at startup to configure the flow data collector **52**. The configuration file **318** specifies various configuration parameters **319**, including a time parameter **320** and a policy **322**. The NAR processes **304** populate and process NARs for data received from network devices via the equipment interface **16** in accordance with the policy **322** of the configuration file. NARs being held in the local store **314** are transferred to the flow aggregation process **60** (FIG. **2**, shown here in dashed lines) when the time specified by the time parameter **320** expires.

It can be appreciated from the above description that the flow data collector **52** is a software component of the accounting process and runs on the flow data collector system **300**. The flow data collector system may be any computer system, such as a workstation or host computer, which can communicate with the equipment interface. Alternatively, the FDC may reside in the network device itself. Many known details of the flow data collector system **300** have been omitted from FIG. **17** for the sake of clarity, as the figure is intended to highlight the processes of and memory structures associated with the flow data collector.

Conceptually, as earlier described, each flow data collector of the accounting process architecture is capable of supporting multiple equipment interfaces **16**. At the implementation level, there is a one-to-one correspondence between each flow data collector "process" and a given equipment interface **16**. For example, a single computer system might provide both RADIUS and flow probe support and thus run separate flow data collector processes for the RADIUS EI and the flow probe equipment interface. In such a configuration, where the flow data collector processes are operating independently and loading directly into the flow aggregation processor **60** (FIG. **2**), the computer system itself may be viewed as an flow data collector supporting multiple EIs.

Referring now to FIG. **15**, a data collection process **330** performed by the flow data collector **52** of FIG. **17** is shown. The flow data collector receives **332** data from the equipment interface for an network device. The flow data collector performs an equipment interface specific translation to convert **336** the received data into NAR format as well as populates the NAR header. Once the NAR is populated with the appropriate data, the flow data collector **52** attempts to correlate **338** the newly populated NAR with other NARs. That is, the flow data collector **52** compares the newly populated NAR to NARs currently stored in the local store **314** (from FIG. **14**) to determine if there are multiple instances of the same object. Specifically, correlation is performed by examining the ACCT_ENTITY_ID (from FIGS. **11A–11E**).

The flow data collector uses one clock and one time determinator, so all NARs that the flow data collector is processing or holding are assumed to be in the same time domain. Consequently, the flow data collector need not consider time during correlation. If the flow data collector **52** determines that a NAR ACCT_ENTITY_ID (i.e., the collection of descriptors or objects as described above) in the NAR matches that of another NAR that it is currently holding, the FDC **52** can replace an older (stored) NAR with the new (i.e., most recently populated) NAR and discard the older NAR. For example, the existing or older NAR may be a start record and the new NAR a stop record that includes all the data included in the older NAR, thus superseding the older NAR. Alternatively, if the new NAR is a replica of an existing NAR, the FDC may decide to discard the new NAR. Also, the data collector can determine that the two NARs should be merged or aggregated. Thus, the correlation process may discard the new NAR, replace an older NAR with the new NAR or mark the two matched NARs as candidates for aggregation, a process which is described in detail below.

As part of the correlation process, the flow data collector may enhance **340** the new NAR. That is, the FDC may determine that the NAR cannot be correlated without some amount of enhancement. The FDC **52** enhances the NAR by supplementing the information provided by the original source equipment with information that is not available from that source equipment. The supplemental information is added to the ACCT_ENTITY_ID. Recall that the accounting entity identifier ACCT_ENTITY_ID is a collection of descriptors, so the enhancement process **310** adds to that collection of descriptors. For example, the accounting entity ID ACCT_ENTITY_ID in one NAR might include a source address and a destination address, along with a value indicating how long the flow (for the accounting entity) has been in existence. A subsequently processed NAR record having those same three objects can be correlated. However, if a subsequently processed NAR only has two of the three objects, the flow data collector can enhance the accounting entity ID ACCT_ENTITY_ID for the third (missing) object to permit correlation. Enhancement may involve collecting information from a completely different network device (via a NAR generated by another accounting process component, such as another data collector), or it may be as simple as adding a timestamp to a NAR's accounting entity ID.

As indicated above, the correlation process may determine **342** that two NARs should be "aggregated". Aggregation merges the accounting entity identifiers of the two NARs together. It also merges metrics for NARs that contain metrics, as later described. Aggregation of the accounting entity identifiers is accomplished through an explicit and implicit matching of those accounting entity identifiers. Correlation relies on the explicitly matched fields, that is, the fields or objects actually used to determine that two NARs should be aggregated. The other descriptors or objects in the accounting entity ID that were not used by the correlation process to make a match may be equal or different. Aggregation of the accountable entity ID portion of the NAR keeps the explicitly matched objects, and determines which of the implicitly matched objects (the matching objects that were not a part of the explicit match) to save or discard. Of course, the nonmatching objects are automatically discarded, as all of the metrics that are the result of this aggregation have to apply to the objects in the aggregated accountable entity ID ACCT_ENTITY_ID. The removal of accounting entity ID descriptors actually serves to lower the semantic complexity of the NAR, whereas enhancement does just the opposite.

When the data collection process **330** involves a decision concerning aggregation, the flow data collector **52** applies **344** the aggregation policy **322** (from FIG. **14**) and uses a method defined therein. The method outlines the decision-making process to be followed by the FDC in the case of implicitly matched objects. The aggregation policy will be discussed in further detail with reference to FIG. **18**. Once the flow data collector aggregates the accounting entity ID ACCT_ENTITY_ID portion of the NAR attributes, it can aggregate the NAR metrics. To aggregate the metrics, the flow data collector performs a summation process on numerical metric values and/or a logical operation (e.g, ANDing, ORing, or XORing) on logical metric values. Aggregation of the metrics is specific to each metric field in the NAR.

Once the NAR aggregation is complete **346**, the FDC changes the NAR header (i.e., the NAR_SRC_ID and NAR_SRC_TIME in the NAR_ID) of the newly aggregated NAR to identify the component (in this case, the FDC) that performed the aggregation as the originator of this particular NAR. The FDC stores aggregated NARs for a period of time determined by the configuration profile's event-based counter or timer **320** (from FIG. **14**). When the timer expires **348**, the FDC is ready to transfer NARs processed by the correlator/(enhancement) and possibly the aggregator as well to the FAP.

Prior to commencing transfer, the flow data collector **52** determines **350** if the flow aggregation processor **60** is available to receive NARs. If the flow aggregation processor **60** is unavailable, the flow data collector stores **352** the NARs to be transferred in its local store **314** (FIG. **16**). The flow data collector **52** continues to check **354** the availability of the flow aggregation processor at periodic intervals until the connection between the flow aggregation processor **60** and the flow data collector is re-established. When the periodic status check indicates **350** that the flow aggregation processor is available, the flow data collector loads **356** NARs into the flow aggregation processor **60**. The loading function can be implemented according to one of many strategies, e.g., a database, file, or data streaming strategy. Other strategies could be used. When the flow data collector receives **358** a confirmation or acknowledgment back from the flow aggregation processor that the NARs were loaded, the transfer is deemed successful and the locally stored copies of the transferred NARs are removed **360** from the local store. Thus, the "store and forward" capabilities of the flow data collector provide a measure of fault tolerance at this accounting process level to ensure reliable data transfer. The flow data collector only transfers NARs when it has determined that the flow aggregation processor is available and it considers the NAR transfer successful only upon receipt of an acknowledgment from the flow aggregation processor.

The flow aggregation processor (FAP) **60** (FIG. **2**) aggregates and/or enhances record data across the system **10**. It receives data from multiple flow data collectors (FDCs) that may be aggregating and enhancing close to the source of the information (as described above with reference to FIG. **17**). As NARs are received from multiple FDCs, the data can be further enhanced and/or reduced (i.e. aggregated) to meet the specific needs of an application or output interface based on the aggregation policy of the flow data processor **60** (FAP). The design and operation of the FAP will be described in more detail below.

Flow Aggregation Processor

Referring now to FIG. **16**, one implementation of the FAP **60** is as a database management system, or more specifically, a Structured Query Language (SQL) database management system, like those commercially available from Oracle or Sybase. Although not shown, it will be appreciated that the FAP is installed on a computer system, such as a host computer. Implemented as a database management system, the FAP includes a database server **400** coupled to a database **402**. The FDCs **52** (from FIG. **14**) can use the "push" model to move NARs up to the FAP via SQL calls. The database **402** stores a plurality of tables **404**, including a NAR table **406** (implemented as a persistent cache) and an aggregation store **408**. Also stored in the database are a plurality of SQL commands and procedures (functions) **410** to be executed by the server **400**. The functions include a FAP correlator **412**, a FAP enhancer (enhancement process) **414** and a FAP aggregator **416**. The database also stores a configuration file **420** for storing configuration parameters such as time and policy information. The operation of the FAP will be described below with reference to FIG. **17**.

[illegible]

Referring to FIG. **17**, an overall flow aggregation process **430** performed by the FAP is shown. The FAP receives **432** a NAR from one or more FDCs and loads **434** the received NAR into a persistent store or cache (of database **492** from FIG. **16**). If the FAP is unable to load the NAR, it requests **436** that the transferring FDC resend the NAR. If the load is successful, the FAP sends **438** an acknowledgment back to the sending FDC. The FAP determines **440** if the NAR can be correlated (with or without enhancement). If the FAP determines that the NAR can be correlated, the FAP correlates **442** the NAR with other NARs received from other FDCs. Once the NAR is correlated, it may be enhanced **444** "across the system", in a manner more fully described with reference to FIG. **18**. The NAR may be enhanced **446** to include enhancement information obtained from an outside source (i.e., collected by a data collector for a different equipment interface). Once any potential correlation and enhancement has been performed, the FAP determines **448** if the NAR is a candidate for aggregation. If so, the FAP applies **450** the aggregation policy **420** (from FIG. **16**) and stores **452** the resulting aggregated NAR in the aggregation store until a predetermined time expires or event occurs **454** (as set in the FAP configuration **420**). The FAP ensures **456** the uniqueness and integrity of any NAR by examining NAR header information prior to re-loading **458** such NAR into the persistent store.

The accounting architecture may be implemented to include a second "shadow" FAP process, also coupled to the data collectors and operating in the manner described above with respect to receiving and processing NARS. In the dual/shadowing FAP implementation, the accounting architecture further includes an error detection module (not shown) coupled to both of the first (primary) and second (shadow) FAP processes. The error detection module operates to detect an error relating to the first flow aggregation process and cause the aggregate reports from the second flow aggregation process to be transferred to the accounting module (i.e., flow data distributor 70) in place of the aggregate reports from the first flow aggregation process.

Enhancement

Now referring to FIG. 18, an example of an application of the FAP enhancement process 444 (from FIG. 20) is shown. In the illustrated example, enhancement deterministically identifies the source of a captured network accounting record, flow or a transaction across a network. Enhancement accesses other sources of information on the network in order to enhance a record and make it chargeable to a specific user.

In the example shown in the figure, two NARs of different sources are inevitably going to be aggregated together to produce a third unique NAR. A first source equipment (or source) 500 is a DHCP (Dynamic Host Configuration Protocol) server. A second source equipment (or source) 502 is a flow probe (discussed below). The sources 500, 502 have corresponding flow data collectors, a first FDC (FDC1), 504 and a second FDC (FDC2) 506, respectively, for converting their data into respective NARs NAR1 508 and NAR2 510. As described earlier, each flow data collector assigns an accounting entity identifier 512, 514, and adds time stamp information 516, 518 on the records of the sources to which they correspond. The NAR1 508 includes in its assigned accounting entity identifier 512 an "IP address-to-username" assignment, thus including an IP address 522 and a username 524. The accounting entity identifier 514 for the second source is an IP-to-IP flow and therefore includes a first IP address 526 and a second IP address 528. The NAR2 of the flow probe includes a metric 530 attribute as well.

These two records NAR1, NAR2 are combined through correlation 442 (from FIG. 17) and enhancement 444 (FIG. 17) to generate an enhanced NAR2 532. This enhanced NAR has a modified accountable entity identifier 534 and a metric. The modified accountable entity identifier is the existing accounting entity ID 514, to which the FAP has added the IP-to-user name assignment 512 from the accounting entity ID 512 of the NAR1 508.

Still referring to FIG. 18, the NAR1 508 has an IP-to-username mapping 512 and an accounting interval 516 comprising a start time and a session time to indicate a time interval bounded by start time "T1" and a start time +session time ("T2"), that is, the accounting interval represents a start time and a stop time. The username 524 in the IP address-to-username mapping is supplied by the DHCP server 500. In the FAP, this NAR1 information will either go directly to a correlation function or to the local store (which could either be a database, file or memory), where it can be directly accessed by the correlator function. The NAR2 510 has an accounting entity ID 514, a T3-to-T4 accounting time interval 518 and a metric 530. The accounting entity identifier 514 has two IP addresses 526, 528, one corresponding to a source IP address and the other corresponding to a destination IP address. The NAR2 502 is passed to the correlator 442, which determines that the T1-to-T2 time interval 516 from the IP-to-username address map in the NAR1 508 overlaps or in some way relates to the T3-to-T4 time interval 518 of the NAR2 510. The correlator determines that T1, T2, T3 and T4 are related, and that the IP address 522 in the IP-to-username address mapping 512 is associated with one of the two IP addresses 526, 528 in the NAR2 510. Thus, the FAP enhances the NAR2 510 by inserting information from the accounting entity ID 512 (of NAR1 508) into the accounting entity ID portion of the NAR2 510. The resulting, enhanced NAR2 532 has an enhanced accounting entity ID 534 that includes the T3-to-T4 timestamp (not shown), the IP-to-IP addresses 526-528 and the username 524. Thus, the enhanced NAR2 now has a mapping between the username and the one of the IP addresses 526, 528 that is related to the IP address 522. The metric 530 is unchanged.

It should be noted that the correlator is able to determine that the time intervals are related to each other because the flow data collectors are time synchronized (or closely synchronized, assuming some amount of drift). Thus, if the correlator assumes no drift, then T3-to-T4 must be within the time period of T1-to-T2. The IP-to-username address mapping is an event that has to encompass or cover all of the accounting records that apply to that IP address. Any user assigned to this IP address, started at T1 and ended at T2. Only those records that reference that IP address between T1 and T2 will have this username applied to it. When the two flow data collectors are not strictly synchronized, then the amount by which T3-to-T4 overlaps T1-to-T2 should correspond to the amount of tolerance, i.e., drift, built into the system. The accounting process assumes a drift amount of at least one second for even a strict time synchronization, so T4 can be greater than T2 by one second.

Referring now to FIG. 19, an aggregation of the enhanced NAR2 532 (from FIG. 18) is shown. In this example, the aggregation involves combining NARs with IP-to-username address mappings to workgroups. To accomplish this requires two enhancements, two correlations, and an aggregation phase. As already described above, with reference to FIG. 19, the IP address-to-username information is received by the FAP and is either passed to the correlation or stored in the local store but available to the correlator. When the IP-to-IP address NAR with metrics is received, the correlator and the enhancer work together to add the username mappings to these IP-to-IP address NAR. The username could be provided for one or both of the source and the destination addresses. More than likely, the username is assigned to the source IP address.

Referring again to FIG. 19, another correlation and enhancement process 442, 444 maps the username 524 to a workgroup. The FAP builds up search keys using database principles and relational algebra. Thus, for example, the IP address has a one-to-one mapping with a username. (The one-to-one mapping is assured because of the nature of IP addressing and the way that the DHCP server assigns usernames.) Therefore, there can be only one user for an IP address in a given instance. These terms or values are equivalent keys, so the username can easily be replaced with the IP address. The username 524 that was inserted into the enhanced NAR2 532 can be used as a look-up into a workgroup 540 in one of the database tables 404 (FIG. 16) because the user is actually a member of a workgroup. Therefore, the enhancement function can be used to insert the workgroup label into the enhanced NAR2 (already enhanced for username) to produce a twice-enhanced NAR2 542. If the now twice-enhanced record 542 is to be aggregated, it is held in the aggregation store 408 (FIG. 16) for some time period T until other NARs are received for potential aggregation.

Suppose now that another NAR is loaded into the FAP. This new NAR passes to correlation, which determines that enhancement is need in order to correlate the new NAR with the twice enhanced NAR2 542 of FIG. 19. As a result, the FAP enhances the NAR to include the username 524 and the workgroup 540 to produce a resultant NAR "NAR3" 550, as shown.

Referring to FIG. 20, in addition to the username and the workgroup, the other NAR3 attributes include the T3-T4 accounting interval, the IP-IP address mapping and the metrics. With the enhancement, the correlation process 444 determines that the resultant NAR3 now matches the twice enhanced NAR2 542 held in the aggregation store 408. Having explicitly matched the two NARs, aggregation 448 is performed. Aggregation preserves the explicitly matched data objects that are in the accounting entity identifier, discards any mismatches in the accounting entity identifier and makes a decision whether to keep the implicitly matched objects (i.e., those that seem to be equal but were not used to make the correlation match). It also then combines the relevant metric values together via summation or logical operations (e.g., ORing, XORing, ANDing). Once the aggregation is complete, the FAP holds the resulting aggregated NAR 552. As the FAP receives additional NARs, the aggregator continues to sum and perform these logical operations on these metrics values for some aggregation period. The duration of that aggregation period may be in the order of 60 seconds to a week, or however long the FAP is configured to aggregate these records. The termination of that period can be a time-based or event-based. Once an event that terminates the time period occurs or an aggregation timer expires, the aggregated NARs held in the aggregation store are released for output by the FAP.

Aggregation Adjustment

It can be understood from the foregoing description that aggregation exists at different levels of the accounting process. As shown and described above with reference to FIGS. 15 and 17, both the flow data collector and the FAP are aggregation-capable. Each aggregates in accordance with an overall aggregation policy that defines how aggregation is used to provide the data to meet the needs of a specific application. The aggregation performed by the different levels can also be remotely and independently adjusted, as will now be described.

Aggregation adjustment involves the ability to adjust the level of aggregation to meet specific application data needs. There are two aspects of aggregation adjustment: remote control and variable degree.

Referring to FIG. 21, a graphical representation of aggregation control and adjustment via a data flow decomposition model is depicted. As shown, the accounting system is depicted as a tree 560. The flow data collectors are leaf nodes 562*a*–562*e* and the two illustrated FAP processes are intermediate nodes 564*a*–564*b*. The root 566 is the collective view of all of the processed accounting information. Given a common view of all the data and the particular accounting information requirements of a given application, the root 566 thus embodies a single accounting/aggregation policy 568. The accounting policy is defined such that an accounting schema is a direct derivative of the accounting policy 568.

The accounting policy 568 is viewed as a collection of accounting objects 570, each defined as an accounting entity identifier 572 and a set of metrics (not shown). The accounting entity identifier is an abstract object resulting from construction functions that use the flow data collector data as its original starting point. If an accounting entity ID is in the accounting policy as a part of a collection of accounting objects, it is there because it can be constructed from the FDC data and the collective set of operations that allow for correlation, enhancement and aggregation. Therefore, if an accounting entity ID can be constructed, it can be decomposed.

To implement a given user/application requirement, therefore, the data flow model 566 decomposes each object's accounting entity ID into policy information 572*a*–g, which includes a collection of data 574 that can be supplied by the available flow data collectors and a set of functions or methods 574 needed to correlate, aggregate or enhance that data in order to construct the accounting entity identifier.

Aggregation adjustment takes an accounting policy that is a collection of accounting objects and decomposes those accounting objects into their accounting entity identifiers and then further decomposes the accounting entity identifiers in a recursive fashion to provide the collection of basic data and functions needed to construct those accounting identifiers. This concept builds on the logical directed graphs as seen in many compilers or data flow systems. Knowing the order of the functions, the data requirements and dependencies, the data flow software can build the logical graph from the decomposition and that specifies data requirements and methods that can be distributed to configuration files in the flow data collectors and FAPs to result in adjusting the configuration of those accounting components.

For example, suppose a user wants to receive accounting on an hourly basis from all of the potential sources of information. The flow data collectors 562*a*–562*e* are the components that are available for collecting the raw information to generate the accounting data in accordance with a user-specified accounting policy. The internal FAP processes 564*a*–564*b* further correlate, enhance and aggregate to evolve the data towards the overall accounting data to meet the accounting policy 568 specified by a user. Thus, the user's information requirements are translated into a policy (i.e., collection of _objects), which is received by the accounting system and decomposed into the sets of data requirements and methods for each of the available accounting components 562*a*-562*e*, 564*a*–564*b*, that is, policy information 572*a*–572*g*). Assuming that these components or processes are already configured, these sets represent configuration updates that are distributed to and stored in the configuration files (see FAP configuration file 420 from FIG. 16 and FDC configuration file 318 from FIG. 14) in their respective processes.

Referring now to FIG. 22, a depiction of the configuration update is shown. The decomposition/configuration update process is implemented in software and is based on known data flow technology used in conjunction with an available visualization tool to act as a front-end graphical interface. Using such visualization tools, the updated configuration is simply mapped to the appropriate component.

It should be noted that not all accounting processes have a complete collection of data collectors. For instance, if the accounting process is to perform user-based accounting and the accounting process only has a flow probe, then it will be necessary to request that the user supply a static table of IP-to-username mappings or a source of DHCP user IP address mappings. The source of that "outside" information becomes part of the decomposition strategy.

Information Management

The NAR sequence number (NAR_SEQ_NUM FIG. 8B) allows components that are in the next level to detect if there are missing NARs in a collection of NARs and can be used to give a sense of how often NARs are produced in a given time period. With the time stamps and the sequence numbers, a per second creation rate of NARs throughout the system can be determined. With this information being part of every NAR, the accounting process 14 can determine a sense of the functional capabilities of the intermediate components and detect some aspects of the communication channel between components. Also included in a NAR identifier is a component type identifier 207*a* which specifies what kind of component produced the NAR and its serial number 207*b* as described above in FIG. 8B. The component type identifier allows the accounting process 14 to keep component statistics and characteristics based on component type. It also allows specific processing on the NARs. NAR IDs are allocated in a very specific way through a management system in order to insure that the IDs are actually unique within the accounting process 14.

Referring now to FIG. 23, the sequence numbers (NAR_SEQ_NUM) are a key reliability feature 590 of the accounting process 14. By having the sequence numbers as part of the NARS and knowing that the numbers are monotonically increasing enables the accounting process 14 to track and identify 592 lost traffic or records. It also enables the accounting process to determine 592 the amount of lost traffic. By having the NARs with stored accounting process component IDs (e.g. a data collector assigned to a particular network device that is allocated at the time that the collector is assigned) the information management process 590, can identify 594 the data collector responsible for the flow. The accounting process 14 can call back to the data collector that produced the NARs of a particular flow and request 596 that the missing NARs (i.e., those NARs for which there are missing sequence numbers) be retransmitted.

Flow Probe

As discussed above in reference to FIG. 2, the accounting process supports a flow probe e.g., 12*c* that captures a user's network activity for purposes of IP accounting. The flow probe 12*c* monitors all traffic over a given network link and captures data associated with the different flows in the traffic on that link. It is capable of monitoring IP data flows over a number of technologies (e.g., Ethernet, ATM, FDDI, etc.).

One important feature of the flow probe is its ability to detect and report on successful and unsuccessful connectivity. This capability is useful to billing and chargeback applications. For example, a user may try to connect to a particular switch or reach a particular network, but is rejected. The flow probe 12*c* can identify that transaction as unsuccessful and provides the billing application with information that the billing application can use in determining whether or not the user should be charged for that transaction. The flow-based connectivity model embodied in the flow probe is described generally with reference to FIGS. 23–25, and specifically with reference to FIGS. 27–28.

Referring to FIG. 24, a representation of a network 600 in which an end system "A" 602 is connected to another end system "B" 604 is shown. The terminal systems A 602 and B 604 communicate with one another over a communication path 606. Along that path are multiple intermediate devices 608 (e.g., routers, switches) to support the communication services required for communications between A and B. Although the path from A to B is depicted as a single straight line, it may be appreciated that the actual physical topology of this path most likely is extremely complex. For the purpose of understanding the flow probe's connectivity model, however, it is not necessary to know how the actual network would be configured.

The physical deployment of the flow probe in a network, such as the network 600, is based on two criteria: performance, e.g., a 100 Mb probe must be deployed within a region of the network that operates at 100 Mb, and granularity of the information to be generated. That is, if the performance or the quality of service provide by A is of particular interest, then the flow probe is located as close to A as possible so that the flow probe will see all of the traffic that is seen by A.

The deployment of the flow probe may be in-line or out-of-line of the stream of IP packets of interest. Thus, the flow probe 12 may be deployed in-line, i.e., integrated into either of the components that are actually party to a conversation (like end station A 602, as shown in the figure), one of the devices 608 that are actually supporting the communication or out-of-line, i.e., packets are copied and delivered to a remote position.

Generally, a flow is defined as any communication between communicating entities identified by an IP address, a protocol and a service port. All IP packets (or datagrams) are categorized using the fields present in the packets themselves: source/destination IP addresses, the protocol indicated in the IP header PROTO field, and, in the case of UDP or TCP, by the packet's source and destination port numbers.

In a given network segment monitored by the flow probe, much of the typical IP traffic includes TCP protocol traffic. Because the flow probe is a flow based monitor that is actually tracking the TCP as a flow, it is completely aware of the TCP protocol and that protocol's three-way handshake algorithm (state machine). The TCP flow has indicators to indicate that a connection is being established or a flow is being disconnected. However, these messages are only relevant to the two communicating parties (e.g., A and B in FIG. 27). The end system A may request that it be able to communicate with B and sends a "TCP SYN" indication. Any of the networking devices 608 along the path 606 can reject this SYN request, completely independent of the intended destination (in this example, end system B) and without the knowledge that the end system B is a party to this communication request. There are a variety of problems that can cause an internal network component to reject a request. For example, a router between A and B may find that there is no route available for forwarding a packet towards B or that the routing path is inoperable (and no alternate exits), or the router may find that it doesn't have the resources to handle the packet.

The Internet Control Message Protocol (ICMP) is designed to convey this type of error event information back to the originator of the request. For example, suppose device 608 is a router that is in a "failed" state and cannot process the SYN request that it received from A. The support exists in the Internet protocol, specifically, ICMP, to signal this condition back to A. Originator A has the ability to correlate the error event with the request and inform the requesting application that its request is not going to be supported. Because the network uses a completely independent protocol, i.e. ICMP, to convey the information, it is necessary to correlate these independent protocols (TCP and ICMP) to provide the accounting process with the information it needs to know about a given transaction. Specifically, the accounting process needs to know if the transaction was successful or unsuccessful and the cause of failure if unsuccessful.

As an independent monitor operating outside of the context of the originating entity ("A", in this example), the flow probe is able to produce a complete and accurate record of the transaction by mapping the network control information to the user request information. To do so, flow probe correlates the state information in protocols such as TCP with error event or condition messages provided by other protocols, such as ICMP. In this manner, it is possible to determine if a particular request for a service has actually been denied as a result of some network independent event. The flow probe correlates the dissimilar protocols together and finds a way of representing the network event in its normal reporting of the TCP flow.

The flow probe has specific reporting mechanisms for the specific protocols. The TCP protocol, for instance, has many more metrics associated with its protocol states than UDP based flows. However, because ICMP relevant events or network relevant events are not associated with or have any impact on the state of TCP or UDP or any of the normal protocols, the flow probe provides a mechanism for tagging its state tracking with the error event. The NAR is represented as a start flow indication, a continuing or status record and a stop record. All of the flow probe's internal protocol indications map to start, continuous or stop states. When a network rejection event comes in (e.g., in the form of an ICMP message, or other type of internet control information), regardless of what state the probe is tracking as the current state, it reverts to a stop state and has to expand upon the normal time or transition based stop conditions to include an specific ICMP event as the cause of the closed state. The flow probe NAR includes bit indications for the actual protocol states that it is tracking. For ICMP generated events, the flow probe indicates whether the source or the destination was affected by the events. In order to convey this network rejection or network event back to the parent flow, the NAR allows for specific network rejection logic to be reported either by the source or the destination, and has specific bit indicators in either the source or the destination fields.

There are two key aspects to the connectivity scheme of the flow probe as described thus far. First, the probe determines that an ICMP event has occurred. Second, the probe correlates that event to the "parent" flow, i.e., the same flow as that associated with the failed request, and stores the exact ICMP event into some state associated with that flow so the event can be reported to the accounting system in a NAR. At this point it may be useful to examine the IP packet and ICMP message formats in general, as well as examine certain fields of interest.

Referring to FIG. 25, an exemplary IP packet format **610** is shown. The IP packet format **610** includes an IP packet header **612** and an IP packet data field **614**. The IP packet header **612** includes a PROTOCOL field **616** for indicating the protocol of the message encapsulated therein. The header also includes a source IP address field **618**, a destination IP address field **620** and other known fields (not shown). In the example of FIG. 25, the message contained in the IP packet data field (or payload) is an ICMP message or packet **622**. The ICMP packet is formatted to include an ICMP header **624** and an ICMP data field **626**. In the example, the protocol field **616** would be set to indicate a protocol value corresponding to ICMP.

Referring to FIG. 26, an exemplary ICMP message format **622** for reporting errors is shown. The format includes an ICMP message header **624**. The header **624** includes a type field **630**, which defines the meaning of the message as well as its format, and a code field **632** that further defines the message (error event). The error reporting message types (type values) include: destination unreachable (3); source quench (4); source route failed (5); network unreachable for type of service (11); and parameters problem (12). Each of the types has a number of code values. For a destination unreachable message (TYPE field value is 3), the possible codes (code values) include: network unreachable (0); host unreachable (1); protocol unreachable (2); port unreachable (3); fragmentation needed and DF set (4); source route failed (5); destination network unknown (6); destination host unknown (7); source host isolated (8); communication with destination network administratively prohibited (9); communication with destination host administratively prohibited (10); network unreachable for type of service (11); and host unreachable for type of service (12).

Also included in the ICMP message format is a datagram prefix field **634**, which contains a prefix—header and first 64 bits of data—of the IP datagram that was dropped, that is, the datagram that triggered the error event message. The datagram prefix field **634** corresponds to the ICMP message (packet) payload. The IP datagram or packet header **612**, partially illustrate in FIG. 24, is shown here in its entirety. Assuming that the IP datagram carries a TCP message, the protocol value would correspond to TCP and the portion of the IP datagram's data **636** (first 64-bits) would in fact correspond to a TCP message header **636**, which includes a source port field **638**, destination port field **640** and a sequence number field **642**. The source port is the port number assigned to the process in originating (source) system. The destination port is the port number assigned to the process in the destination system.

It will be understood that TCP is an example protocol. The field **636** could correspond to a portion of packet header from a packet of another protocol type. Also, the error reporting protocol could be a protocol other than ICMP, and the amount of header in field **636** could be more or less than **64** bits, that is, this amount may be adjusted so that the appropriate flow information can be obtained from the header of the message contained in the discarded IP packet, as described below.

Referring to FIG. 27, a packet processing method ("the process") **650** performed by the flow probe is shown. The process captures **652** a new IP packet (datagram) and tests **654** the received packet to determine if it is good (i.e., well-formed). The process **650** examines **656** the protocol field in the IP packet header to determine if the protocol is the ICMP protocol. If the protocol is ICMP and the information type field is set to one of the five error reporting messages described above, the process bypasses the IP packet and ICMP message headers and processes **658** the ICMP message or packet payload (FIG. 26), which corresponds to a portion of IP packet which that was discarded and to which the event message relates. The payload process will be described with reference to FIG. 28 below. Once the payload processing is complete, the processing of the IP packet resumes **659** the processing that would be performed if the IP packet had not been detected as containing an ICMP message of the error reporting variety as discussed above, as will now be described.

Still referring to FIG. 27, if the protocol is not ICMP and/or the information type is not an error report, the IP packet is processed as follows. The probe scans **660** the header to determine the values of the fields which correspond to the "flow key", the fields which define "the flow" for the probe. Each flow probe can be configured for a particular flow key definition. For example, the flow key might be the source/destination IP addresses, the source/destination ports and the protocol. The probe determines **662** if the flow key of the processed packet header matches a flow already stored in the flow probe. A local store in the flow probe is used to hold flow representations including flow key parameters, metrics, state information. The state information will include, in addition to the protocol control-related states (i.e., TCP "FIN"), error event/state change cause and source/destination to which the message is addressed. These flow representations are converted into NARs for accounting process reporting purposes.

Still referring to FIG. 27, if the flow probe cannot match **664** the flow key information to a stored flow, the probe constructs (and stores) **666** a new flow and completes **668** the process. If the probe finds a match, it updates **670** metrics for the matching stored flow (or "parent" flow). It also updates **672** the flow state of the parent and then completes **674** the process. It should be noted that the construction of a new flow triggers **676** the generation of a start NAR and the update of the flow state triggers **678** the generation of an update NAR. The generation of NARs by the flow probe will be discussed later.

Referring to FIG. **28**, processing of the ICMP message payload, i.e., the embedded IP packet, **658** (from FIG. **27**) is shown. The processing of the ICMP message payload processing is recursive in nature. The essential method is the same as used above for an IP packet (FIG. **27**), with a few differences. If the flow probe determines **664** that there is no stored flow corresponding to the flow of the dropped IP packet or datagram (indicated by the ICMP message in the data prefix field or payload **634** of FIG. **26**), the processing is complete **680**. If a stored flow matching the flow key of the dropped datagram is found, the probe updates **672** the flow state to indicate a "rejected" state for the stored flow. It also updates the flow state information to indicate whether it was the stored flow's source or destination that was associated with the ICMP message and the event cause. The state change (to rejected state) triggers **682** the generation of a stop NAR, as is later described. Once the probe has completed the payload processing **658**, it resumes **659** the processing of the original IP packet (as indicated in FIG. **27**).

Thus, the payload processing can be viewed as a packet processing exception, an exception that is invoked when it is determined that an ICMP error reporting message has been received. The ICMP messsage reports a error event and the IP packet associated with that error event. The exception process serves to correlate the flow of the discarded IP packet in the ICMP message with the parent (matching stored) flow, thus mapping the ICMP error (state) information to the parent IP flow.

The flow probe reports on network traffic activity through a flow probe NAR, which reports IP flow traffic activity. The flow probe categorizes network traffic into one of four classes of traffic flow: I) connection oriented (e.g., TCP); ii) new connectionless; iii) request/response connectionless (e.g., UDP, DNS); and iii) connectionless persistent (e.g., NFS, Multicast BackBONE or "MBONE" multicast traffic). To each of these class it applies connection oriented semantics for a uniform approach to status reporting. That is, flow probe treats these dissimilar transaction models as if they were the same. There is one uniform structure for the status reports generated for each of the 4 different transactions. Each status report includes transaction start and stop information, MAC and IP source and destination addresses, the IP options that were seen, the upper layer protocol used, the transaction source and destination byte and packet counts and upper layer protocol specific information. The protocol specific information and the criteria for when the status reports are created, is different for each of the four transaction types.

The connection oriented protocol understood by the flow probe is TCP. Flow probe has complete knowledge of the TCP state machine and thus can generate status reports with each state transition seen within any individual TCP. There is also a provision for generating time interval based status reporting in the TCP connections that the flow probe is tracking. The status report indicates which states were seen, if any packets were retransmitted, if the source or destination had closed, and if the report had been generated by a time condition. In a default mode, the flow probe generates a cumulative status at the time a TCP closes, or times out. This strategy offers the greatest amount of data reduction on transactions.

Any non-TCP traffic is categorized as a connection-less transaction. When configured to generate the most detailed level of reporting for connectionless traffic, the flow probe can report the discovery of a new connection-less transaction; the existence of a request/response pair within the transaction (as exists when the probe has seen a single packet from both the source and the destination for the transaction); the continuation or transaction persistence, and so forth. The transaction persistence status is generated with a timer function. If it has been seen within a configured timer window, a report is generated.

The status report for non-TCP traffic indicates if the report is an initial report, a request/status report or a continuation (or a current transaction) report.

In the default mode, the flow probe generates a status report when it has seen a request/response "volley" within a transaction and every 15 minutes thereafter, if the transaction persists. This offer immediate notification of request/response traffic and a fair amount of data reduction on connection-less transactions.

Thus, the flow probe state tracking includes protocol-specific state information. It provides detailed information on transport specific flow initiation, such as TCP connection establishment, as well as flow continuation and termination event reporting.

Protocol Independent Packet Monitor

Referring to FIG. **29A**, a network **700** includes a monitor **702** that runs a process for detecting packet loss. The monitor **702** will be particularly described using IP SEC authentication headers. The monitor **702** uses sequence numbers that exist in IP SEC authentication headers. The monitor **702** can be used to detect lost packets in any type of protocol that uses sequence numbers in headers of the packets, etc. The monitor **702** is an independent monitor that can be disposed anywhere in the network **700**. The monitor **702** is protocol independent.

The network **700** would include a plurality of such independent monitors **702** each disposed at corresponding single points in the network **70**. Typically, the monitor **702** can be disposed in-line such as in a network device such as a switch, router, access concentrator, and so forth. Alternatively, the monitor can be disposed in an out of line arrangement in which network packets are copied from the device and coupled to the out-of line monitor.

The monitor **702** examines each packet of a network flow that passes through the device associated with the monitor **702**. The monitor **702** receives serialized IP packets. The packets can have the format specified by the Network Working Group, by S. Kent, Request for Comments: 2402, November 1998 "IP Authentication Header" as part of the "Internet Official Protocol Standards", The Internet Society (1998). The IP Authentication header includes a Next Header field that identifies the type of the next payload after the Authentication Header, Payload Length an 8-bit field specifies the length of AH, and a reserved 16-bit field. The IP Authentication header also includes a Security Parameters Index an arbitrary 32-bit value that, in combination with a destination IP address and security protocol, uniquely identifies the Security Association for a datagram and a Sequence Number. The sequence number is an unsigned 32-bit field containing a monotonically increasing counter value (sequence number). It is always present in such datagrams and is provided form the purpose to enable an anti-replay service for a specific security authentication. According to the standard if anti-replay is enabled the transmitted Sequence Number is not allowed to cycle. Thus, the sender's counter and the receiver's counter are reset by establishing a new security authentication and thus a new key prior to the transmission of the 2^{32nd} packet. The datagram also includes Authentication Data, i.e., a variable-length field that contains the Integrity Check Value (ICV) for the datagram.

Referring now to FIG. 29B, a packet loss detector process **704** that runs in the monitor **702** is shown. The packet loss detector process **704** examines **706** header information in the packet, to determine if the packet includes an authentication header. If the packet does not include an authentication header, then the packet loss detector process **704** ignores **24** the packet and exits to wait for the next packet. If the packet includes an authentication header, the packet loss detector process **20** tests **708** to determine if the packet loss detector process **20** had been tracking the flow that is represented by the source and destination IP addresses and the SPID value that is in the authentication header. The packet loss detector will perform a cache look up to determine if the flow is stored in a cache of currently tracked flows. The packet loss detector process **20** tests **708** those values to see if the packet loss detector process **704** is currently tracking that security flow.

If the packet loss detector process **704** is not tracking that security flow, the packet loss detector process **20** will establish **710** a flow cache entry for that flow in a cache that can be maintained in memory (not shown). The packet loss detector process **704** will store the source and destination IP address and the SPID value from of the authentication header. The flow cache also includes all other authentication headers from other security flows that have previously been tracked. The flow cache enables the packet loss detector process **20** to monitor and track many hundreds, thousands, and so forth of different security flows. A cache entry is established for every different flow. Once the cache entry is established, the packet loss detector process **704** updates **712** the sequence number entry in the cache for that security flow. That is, the initial sequence number in the authentication header for the encountered flow is stored. The sequence number can start at any arbitrary value.

If, however, the packet loss detector process **704** deterimined **708** that it is tracking the flow, then the packet loss detector process **704** tests **714** if the sequence number in the current packet is equal to the previous sequence number noted for this flow plus 1. If the sequence number in the current packet is equal to the previous sequence number plus 1, then the packet loss detector process **704** can stop the current evaluation because the packet loss detector process **704** did not detect and the system did not experience any packet loss on that particular association. The packet loss detector process **704** will update **712** the stored sequence number for that flow in the cache.

If the sequence number in the current packet does not equal the previous sequence number noted for this flow plus 1, the packet loss detector process **704** for the IP SEC Authenication packets detected a potentially missed packet.

For some protocols that permit wrap around, the packet loss detector process **704** tests **718** if the sequence number has wrapped around e.g., gone from 32 bits of all ones to 32 bits of all zeros. The IP SEC Authentication packets currently do not permit wrap around, so test **718** would not be necessary for IP SEC Authentication Headers. If for other protocols (or latter versions of the IP SEC Authenication protocol), the packet loss detector process **704** detects a wrap around condition then there has not been any packet loss and the packet is dropped. The packet loss detector process **704** will update **712** the stored sequence number for that flow in the cache. If the sequence number is any other number, i.e., it did not turn over to all zeros, then there may have been packet loss. If there may have been packet loss, the packet loss detector process **704** can determine how many packets have been lost by determining how many sequence numbers are missing.

When packets may traverse more than one packet monitor **10**, the packet loss detector process **704** may produce a packet loss detected indication that does not indicate that the packets were actually dropped. A packet loss drop indication in a multi-monitor embodiment indicates that the lost packets did not come through the particular packet loss detector process **704**. However, the indicated lost packets could be on other segments of the network. That is, it is possible that other parts of the current flow are in other parts of the network. Therefore, the packet loss detector process **704** notes how many packets were actually successfully transmitted, as well as lost, and optionally their sequence numbers. These values can be compared to other values from other monitors **702** to establish whether or not there had been packet loss for the flow through the network.

This indication, could be converted into Network Accounting Records thus would be coupled to a process e.g. the accounting process **14** that reports statistics on that particular flow to provide a summary of how many packets were lost relative to how many packets were actually successfully transmitted on the flow. In the accounting process **14**, the network accounting records are correlated, aggregated, enhanced and so forth to identify network flows. This information can be used to determine the records that correspond to a particular network flow and whether a determined network flow lost any packets.

Capturing Quality of Service

Referring now to FIG. **30**, a process **730** for capturing quality of service in a network system **11**, (FIG. **1**), is shown. The capturing quality of service process **730** allows an administrator to configure **732** the network **11** with a policy that corresponds to a first quality of service. The process **730** also includes an optimization process that assigns or develops **734** the policy, defines the policy being used, and enforces the policy by deploying a policy dictated configuration into various policy enforcement devices in the network **11**. The capturing quality of service process **730** allows the administrator to observe **736** the actual service delivered by the network **11** to a customer on the network **11** to determine if the quality of service provided matches that specified by the policy **740**.

The capturing quality of service process **730** uses an accounting process **738** to collect information from the network. A preferred accounting process is accounting process **14** described above. The accounting process **14** collects data from the network **11** as part of the observation process **736**. The accounting process collects different kinds of metrics from the network, correlates these metrics to specified network flows, via the use of NARS, and maps collected, correlated information i.e., NARs back to the policy that was defined and actually deployed in the network. Because the accounting process **14** performs this observation function, the accounting process can provide an indication **738***a* whether or not the policy **740** is being satisfied.

By deploying the accounting process **14** to observe service quality, the capturing quality of service process **730** can validate performance of service level agreements (not shown). If the capturing quality of service process **730** detects that the policy level specified in a service level agreement is not being enforced, then the policy can be reassessed, redefined, and redeployed **742**. The capturing quality of service process **730** can again observe **737**. Through the observation **736**, the capturing quality of service process **730** can determine whether reassessment and redefining of the deployed policy was successful. Several cycles of this quality of service optimization process could be required.

An important component of quality of service includes determining whether there has been packet loss. The packet detector monitor described in conjunction with FIGS. **29A** and **29B** can be used to access packet loss. The packet detection monitor **702** can be deployed in the network and generate NARs that can be used to determine packet loss as discussed above. This information can be used in the capturing quality of service process **730** to assess whether the policy specified by the service level agreement was provided to the customer. Additionally, so called Differentiated Service "DivServe technology" that a known quality of service solution that has been proposed for the Internet as well as enterprise networks. In contrast to a per-flow orientation of some types of quality of service solutions such as Int-serv and RSVP, DiffServ enabled networks classify packets into one of a small number of aggregated flows or "classes", based on bits set in the type of service (TOS) field of each packet's IP header. This is a quality of service technology for IP networking is designed to lower the statistical probability of packet loss of specific flows. The capturing quality of service process **730** establishes DivServ policy, that is decomposed into a collection of DivServ configurations. The DivServ configurations are deployed to a collection of routers or switches that the customer would have access to in the network **11** as part of the enforcement/deployment process **732**. Because packet loss is a statistical phenomenon, the capturing quality of service process **730** observes **736** a large number of network flows. The capturing quality of service process **730** can observe network traffic because of the use of the accounting process **14** and the resulting NARs at the granularity in which the DivServe policies are actually being deployed. The DivServe policies are generally deployed at the source and destination IP address, protocol and possibly destination port level.

By observing **736** network flows at the same granularity as a DivServe policy enforcement mechanism, if the capturing quality of service process **730** detects packet loss at that granularity, then there will be a direct feedback coupling to determine whether the policy is actually being enforced or not. If the policy is not being enforced, then an administrator will can reassess the policy, redefine the policy, and redeploy **742** new enforcement strategies. The capturing quality of service process **730** again will observe **736**.

As mentioned, because IP network quality of service is a statistical phenomenon, the capturing quality of service process **730** obtains a large number of samples, over a long period of time. Through this optimizing capturing quality of service process **730** and DivServe deployment **734**, the customer will get beneficial policy deployment for this service.

Service Management

Referring now to FIG. **31**, a service management loop **750** includes a service provisioning application **752**, a policy enabled network server **754** and an accounting process **756**. In a typical example, an Internet Service Provider (ISP) and a customer will enter into a service agreement or contract **751** that will specify a level of service for the network. The contract **751** has requirements and conditions that are enforced by the policy enabled network **754**. The service contract **751** is decomposed by the policy server to produce a template that defines the service represented by the agreement **751**. The template is fed to the service provisioning application **752** that actually produces a configuration file **752*a*** that is sent out to the network **10** to configure network for a level of service based upon that contract **751**.

A service management feedback process **750** therefore includes three components, service provisioning **752**, policy server **754** and service accounting **756**. The role of service provisioning **752** is to send requests **752*b*** to the policy server **754** to obtain an appropriate active policy, and obtaining rules and domain information **754*a*** from the policy server. The provisioning system can communicate with appropriate network management systems and element management systems (not shown) to configure the network **10** for an end-to-end service. When the configuration **752*a*** is deployed at the various network devices (not shown) at that point, the service is produced. The level of service is monitored or audited by the accounting system **756** which can be the accounting process **14** described above. The accounting process **14** monitors the level of service by producing appropiate network accounting records. The network accounting records NARs are used by a billing application to adjust billing based on the level of service that was provided as determined by the accounting system **14**. The accounting system **14** also can compare the policies produced by the policy server to the actual levels of service provided to the customer by examining NARs that are produced by the customer's usage of the network.

In addition, levels of service might change, and the system takes changes into account so that the service management can modify the charge or account differently for those changes in levels of service. The service accounting also uses the active policy information from the policy server to deliver billing information to a billing system or to a chargeback system that can may adjustments to billings for the service.

A policy enable network **754** is build on the capabilities of address management, domain name management and so forth. Essentially in a policy enabled network, policy services produce a set of rules and applys those rules to a domain or problem set. The policy server communicates the rules to the accounting process **14** so that the accounting process **14** can determine what kind of records to generate. All of the information is described using data flows.

As an example, a service contract may specify that a company "X" will be given 100% availability of a particular network device e.g., a router (not shown) and its corresponding service. In order to assure that level of service, the policy server **754** sends that requirement in a template to the provisioning service **752** to produce a configuration file **752*a*** to configure the router to give company "X" preferred use for the router. Therefore, every time a packet from company "X's" network comes across the router, the packet will always be transmitted unless there is something wrong with the router. This may occur even if a packet of company "Y" which has a lower service level than company "X" is waiting in the router to be transmitted. The packet from company "Y" will wait because company "Y" is not paying for the quality of service that company X" is paying for.

In that case, the provisioning service configures **752** the policy enforcement mechanism that was put into the router in the network. How the policy was defined to the provisioning equipment is that there is a one-to-one relationship between the policy and what the accounting process **14** will monitor in the network. The accounting process **14** will be aware that company "X" contracted to have 100% availability from the router.

The accounting process **14** will then take every source of information it has available and will construct an accounting record that reflects the level of service actually delivered to company "X." The accounting records produce are relative to the two components, i.e., the router and the customer. The accounting process **14** is flexible and can generate accounting records of any flow abstraction. In this process **750**, the policy server **754** sends a flow based policy to the provisioning server **752**. The provisioning server **752** uses a flow based policy to configure the network. That same flow based policy is passed to the accounting process **14** which can generate network accounting records NARs having metrics that can be used to match the same level of those flows. The output of the accounting process **14** will determine whether the quality of service, availability, etc. that was contracted for in the contract **751** was provided. Therefore the service management process **750** provides the level of service that was delivered at the same semantic level as the actual contract.

Capturing quality of service as audited by the accounting process **14** includes detecting of packet loss, as mentioned above. Each of the components managed by the service management process **750** require information. Therefore, the service provisioning has to provision these various quality levels. The policy server **754** thus, keeps what is essentially enforcement of the levels of quality that are offered by different service types, and the accounting process **756** detects, monitors and audits whether those classes in quality of service are being delivered.

Referring to FIG. **32**, an implementation of the service management provisioning **752** is shown. The service management provisioning **752** extends concepts of device management and network management into a service management layer of functionality. Service management provisioning includes a provisioning core **782**, provisioning modules **784**, and element managers **786**. Service provisioning **752** is user focused rather that network focused as conventional network management. Network management involves communication with network systems and equipment. Service provisioning **752** is orient more towards a user and a user's concepts of services. Service provisioning **752** provides an additional layer of abstraction that relates description of services at a user level to a network's ability to provide those end-to-end services. The architecture **780** of Service provisioning **752** is multi-device **788** at the bottom of the architecture and multi-service **790** at the top of the architecture. The service provisioning **752** is deployed to write commands to the network systems i.e., network elements **788** in order to change configurations of those systems.

Since many end customer services now require that a network operate with multiple, different kinds of network elements in order to provide an end-to-end service, the service provisioning **752** simplifies producing information that is necessary for a service provider to translate a service order from a customer to a network configuration, i.e., all commands necessary for all the different elements in the network in order to create an end-to-end service.

The service provisioning builds on existing systems. That is, in the lower layers there are existing element managers that have a configuration management system to configure at the network layer. The service provisioning adds layering over the conventional network management layer. Service provisioning maps a customer specified end to end service to the network elements that are required to produce that end-to-end service. Mapping of a customer's service orders into the state of the network can have various pieces of workflow necessary to create or completely activate this service order.

OTHER EMBODIMENTS

It is to be understood that while the invention has been described in conjunction with the detailed description thereof, the foregoing description is intended to illustrate and not limit the scope of the invention, which is defined by the scope of the appended claims. Other aspects, advantages, and modifications are within the scope of the following claims.

What is claimed is:

1. A method for tracking network accounting records in a accounting process that collects and correlates information derived from network data comprises:

producing a network accounting record that has an identifier that uniquely identifies the record within the accounting process with the identifier including a sequence number that specifies a sequence number for network accounting records that originate from the source device;

determining when there is a break in the sequence numbers of network accounting records produced from the source device; and

requesting missing network accounting records when there is a break in the sequence.

2. The method of claim **1** wherein producing a network accounting record further comprises:

producing a network source identifier that identifies a source device that creates the network accounting record.

3. The method of claim **2** further comprising determining the data collector that produced the missing network accounting records.

4. The method of claim **3** wherein determining the data collector comprises:

examining the network source identifier in a data flow.

5. The method of claim **4** wherein the data flow is identified by aggregating received network accounting records and correlating the received records to identify a flow.

6. A system comprising:

a data collector collecting data from a network device, and producing network accounting records from the collected data; and

a flow aggregation process, that receives network accounting records, the network accounting records including data identifying the data collector and a sequence number, said flow aggregation process detects missing network accounting records by detecting at least one missing sequence number;

wherein upon detecting a missing sequence number, the flow aggregation process retrieves data identifying the data collector from received records that have been correlated to identify a flow associated with the missing records; and

sends a request to the identified data collector to retransmit the missing record corresponding to the missing sequence number.

* * * * *

US006330226B1

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

(10) **Patent No.: US 6,330,226 B1**
(45) **Date of Patent: Dec. 11, 2001**

(54) **TCP ADMISSION CONTROL**

(75) Inventors: **Alan Stanley John Chapman**, Kanata (CA); **Hsiang-Tsung Kung**, Lexington, MA (US)

(73) Assignee: **Nortel Networks Limited**, St. Laurent (CA)

(*) 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.: **09/014,110**

(22) Filed: **Jan. 27, 1998**

(51) **Int. Cl.**^{7} **B65H 9/08**
(52) **U.S. Cl.** **370/232**; 370/233; 370/234
(58) **Field of Search** 370/229, 241, 370/230, 231, 232, 233

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,349,578	*	9/1994	Tatsuki et al.	370/244
5,361,372	*	11/1994	Rege et al.	395/800
5,410,536	*	4/1995	Shah et al.	370/216
5,444,706	*	8/1995	Osaki	370/230
5,553,057	*	9/1996	Nakayama	370/241
5,729,530	*	3/1998	Kawaguchi et al.	370/236
5,812,525	*	9/1998	Teraslinna	370/229
6,041,038	*	3/2000	Aimoto	370/229

FOREIGN PATENT DOCUMENTS

0 473 188	3/1992	(EP).
99/66676	12/1999	(WO).

* cited by examiner

Primary Examiner—Hassan Kizou
Assistant Examiner—Thien Tran
(74) *Attorney, Agent, or Firm*—Allan P. Millard

(57) **ABSTRACT**

Congestion at a network node can be aggravated by having too many TCP connections. A simple method of avoiding the bad effects of too many TCP connections is to limit the number of connections. Limiting the number of connections is achieved by an admission control which delays or even discards the connection set-up packets. TCP traffic flows are monitored to generate packet loss characteristics and when a certain condition is met, a connection request queue is disabled.

20 Claims, 6 Drawing Sheets

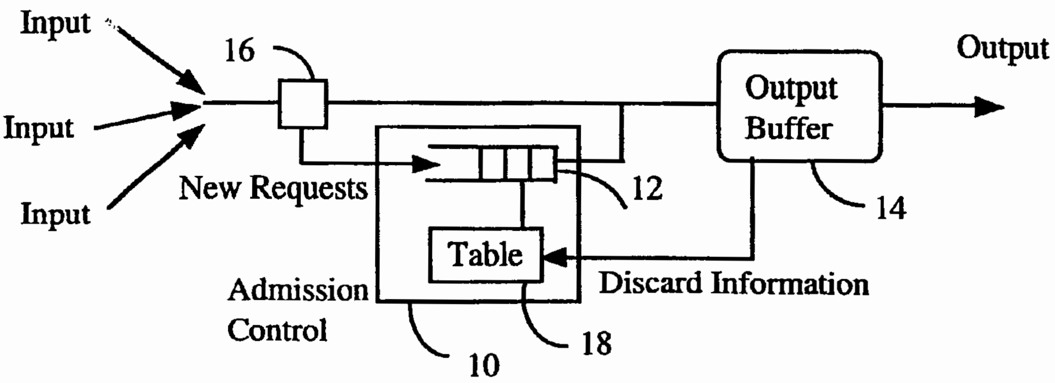
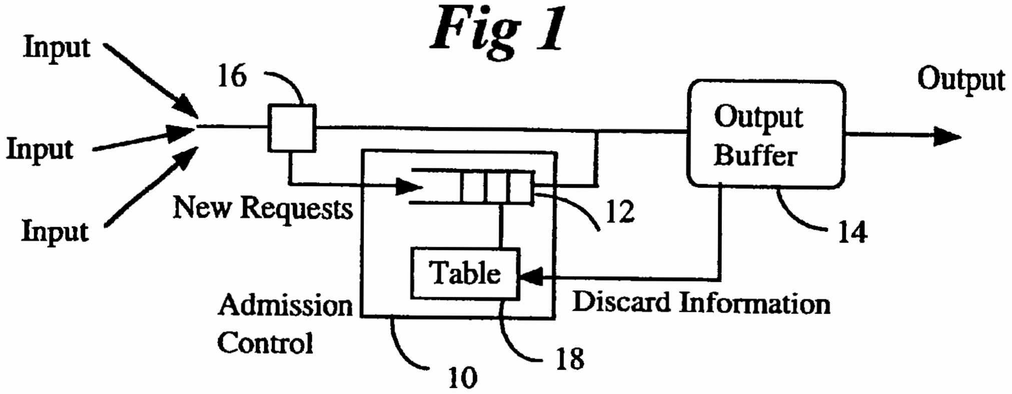


Fig 1



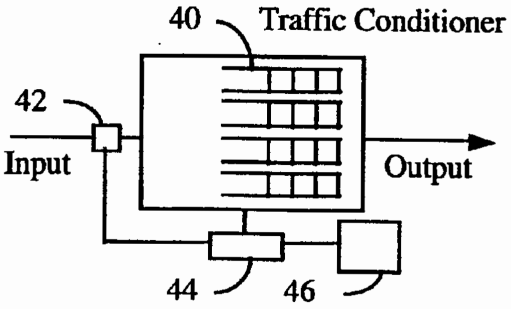


Fig 3

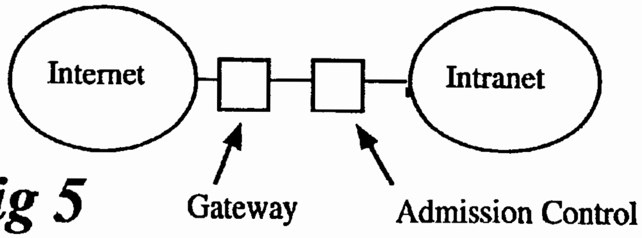


Fig 5

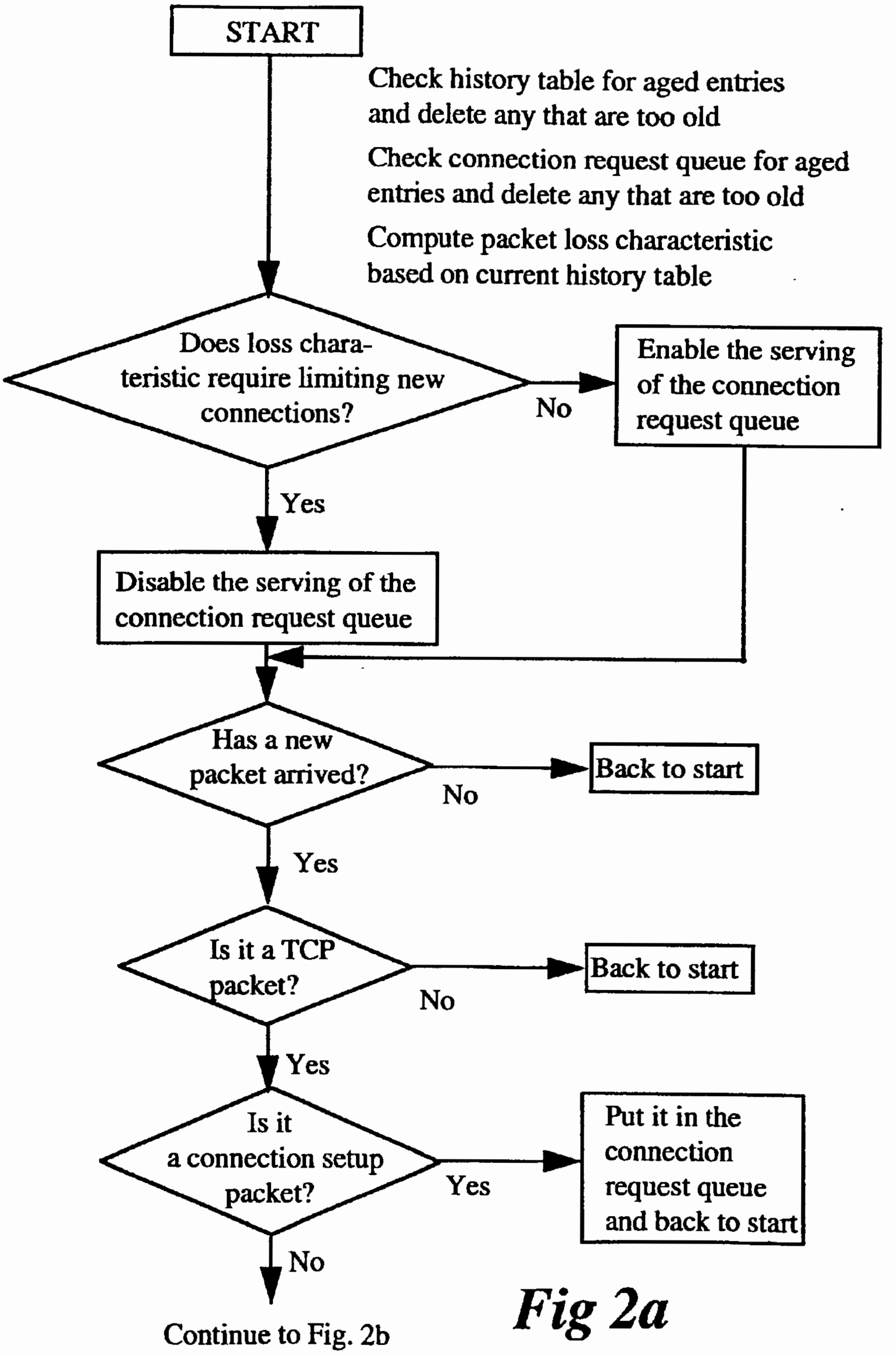

START
Check history table for aged entries and delete any that are too old
Check connection request queue for aged entries and delete any that are too old
Compute packet loss characteristic based on current history table
Does loss characteristic require limiting new connections?
No
Enable the serving of the connection request queue
Yes
Disable the serving of the connection request queue
Has a new packet arrived?
No
Back to start
Yes
Is it a TCP packet?
No
Back to start
Yes
Is it a connection setup packet?
Yes
Put it in the connection request queue and back to start
No
Continue to Fig. 2b

Fig 2a

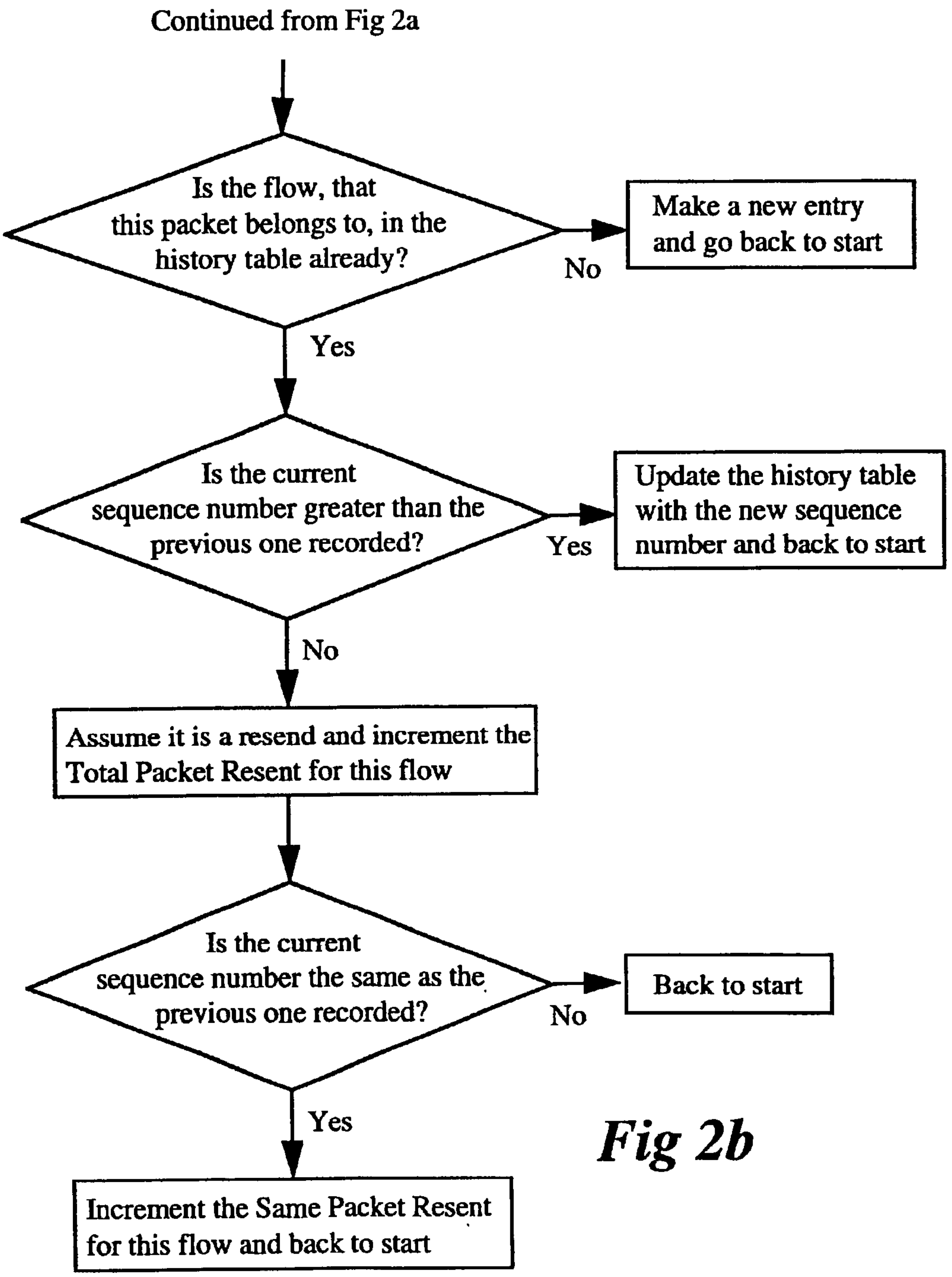

Continued from Fig 2a
Is the flow, that this packet belongs to, in the history table already?
No
Make a new entry and go back to start
Yes
Is the current sequence number greater than the previous one recorded?
Yes
Update the history table with the new sequence number and back to start
No
Assume it is a resend and increment the Total Packet Resent for this flow
Is the current sequence number the same as the previous one recorded?
No
Back to start
Yes
Increment the Same Packet Resent for this flow and back to start

Fig 2b

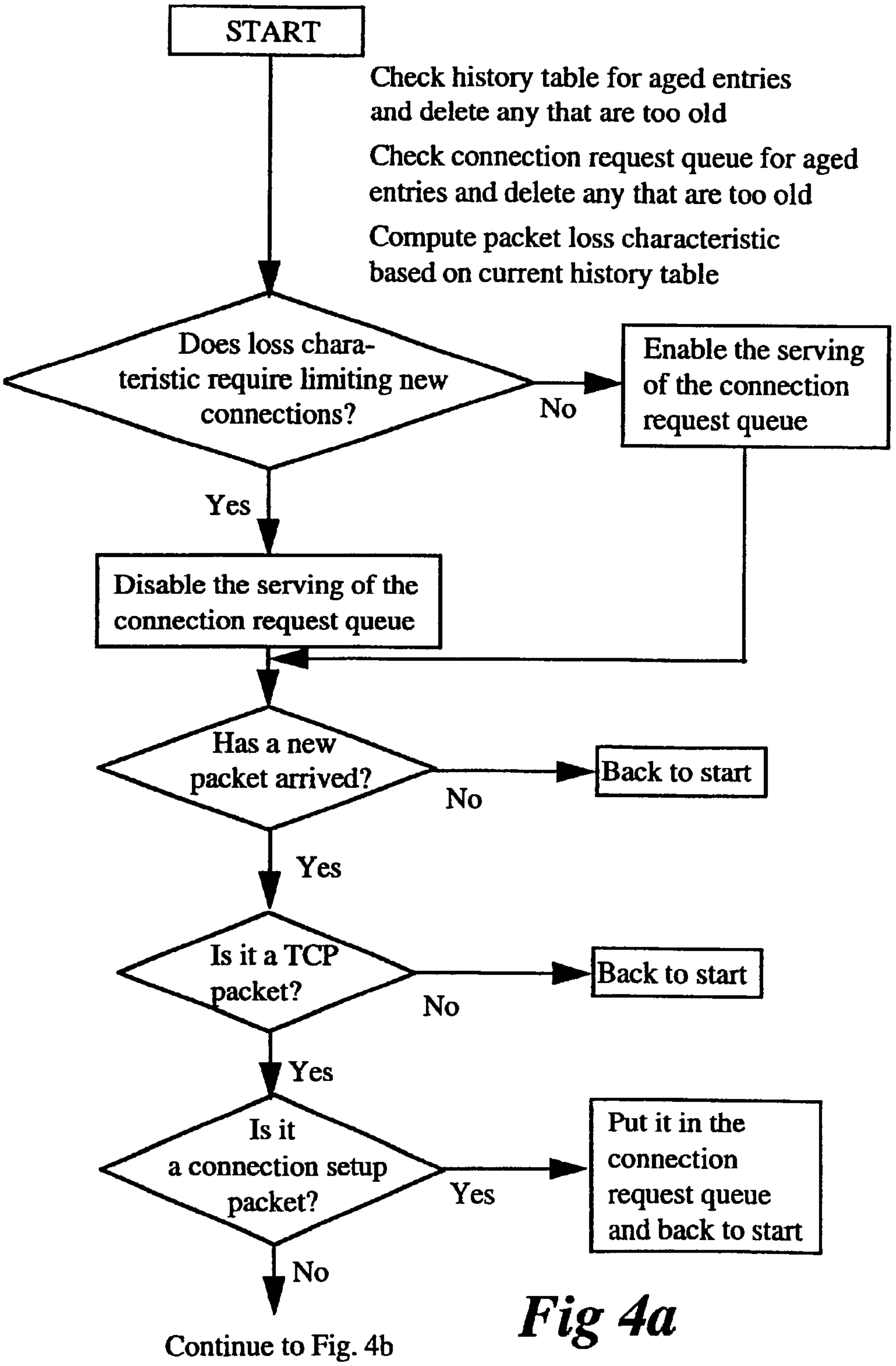

START
Check history table for aged entries and delete any that are too old
Check connection request queue for aged entries and delete any that are too old
Compute packet loss characteristic based on current history table
Does loss characteristic require limiting new connections?
No
Enable the serving of the connection request queue
Yes
Disable the serving of the connection request queue
Has a new packet arrived?
No
Back to start
Yes
Is it a TCP packet?
No
Back to start
Yes
Is it a connection setup packet?
Yes
Put it in the connection request queue and back to start
No
Continue to Fig. 4b

Fig 4a

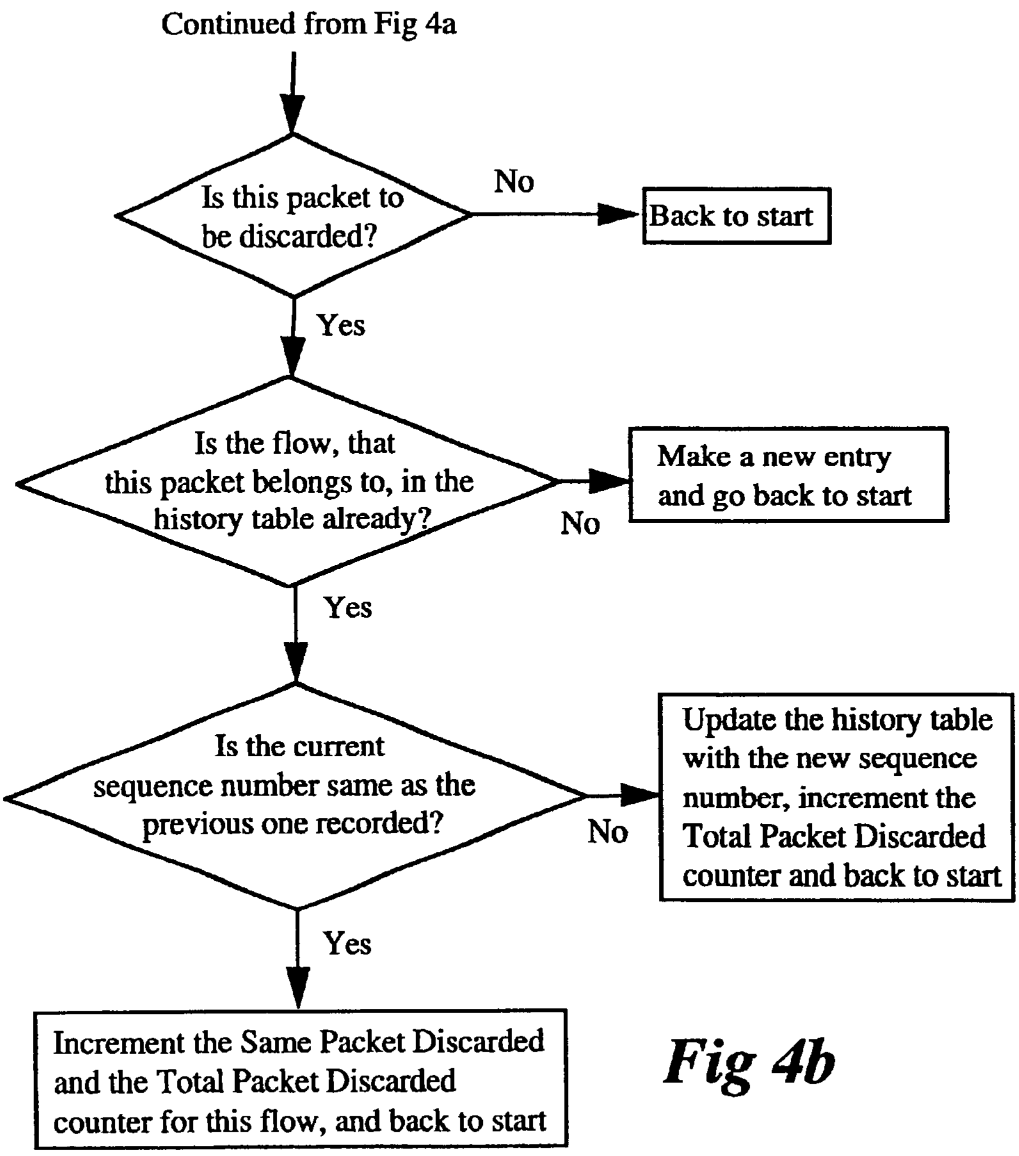

Continued from Fig 4a
Is this packet to be discarded?
No
Back to start
Yes
Is the flow, that this packet belongs to, in the history table already?
No
Make a new entry and go back to start
Yes
Is the current sequence number same as the previous one recorded?
No
Update the history table with the new sequence number, increment the Total Packet Discarded counter and back to start
Yes
Increment the Same Packet Discarded and the Total Packet Discarded counter for this flow, and back to start

Fig 4b

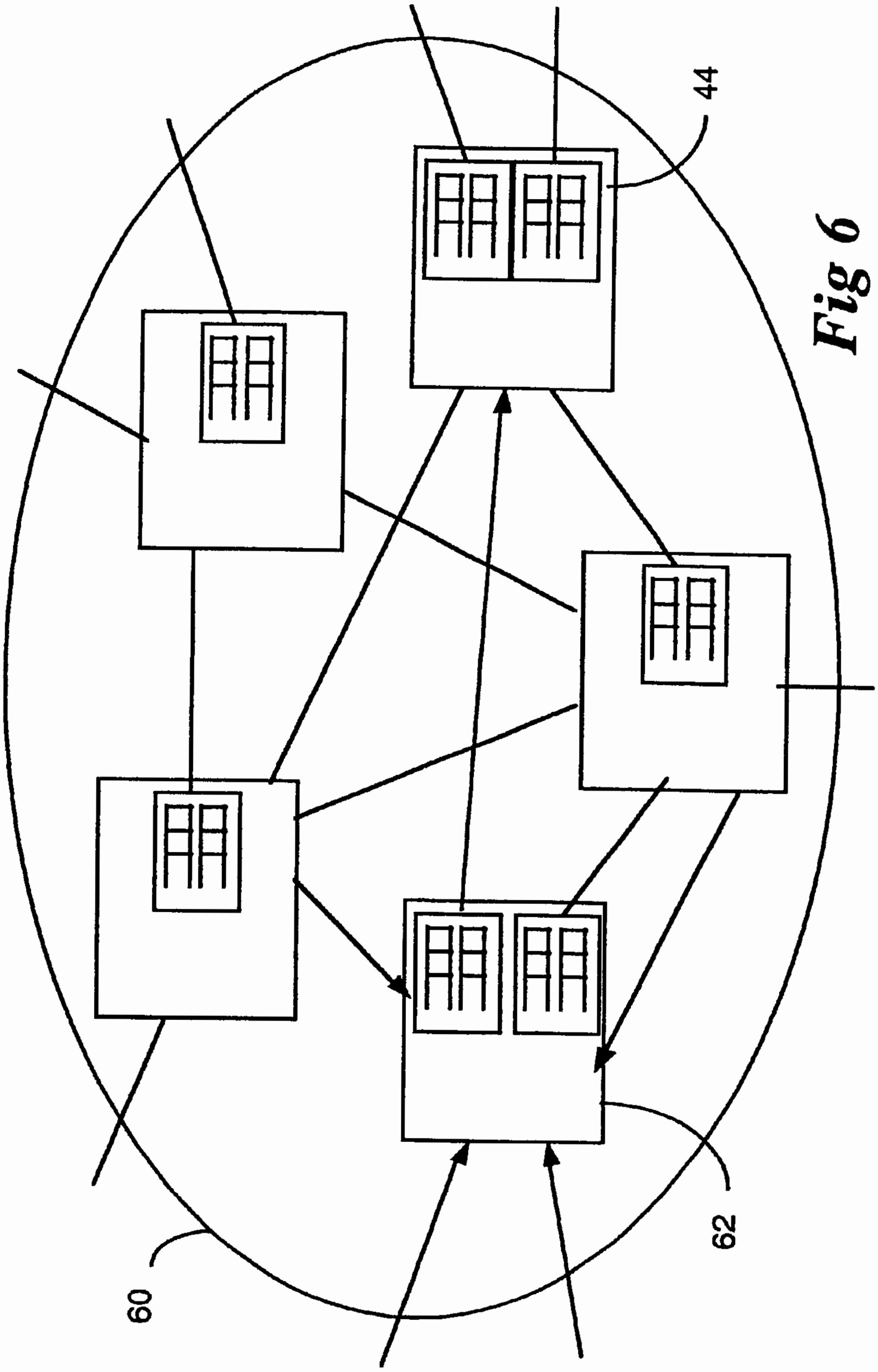


Fig 6

TCP ADMISSION CONTROL

FIELD OF THE INVENTION

The invention relates generally to traffic congestion management of a data network. In particular, it is directed to a technique by which congestion in the data network is controlled by limiting new TCP connection setups based on packet loss characteristics of the data network.

BACKGROUND OF THE INVENTION

The current data networks are handling not only enormous volume of traffic but more and more diversified multi media traffic, causing the data network to become congested more often. When congestion causes an excessive number of packets to be dropped, it can easily impact many traffic flows, and cause many timeouts. By guaranteeing a certain number of traffic flows a minimum bandwidth and treating the remainder as best effort, it is possible to avoid spreading high packet loss over so many flows and to reduce the number of aborted flows. Pending U.S. patent application Ser. No. 08/772,256 filed on Dec. 23, 1996 and Ser. No. 08/818,612 filed on Mar. 14, 1997 by the present inventors describe dynamic traffic conditioning techniques which make use of this concept. The dynamic traffic conditioning techniques described therein allow the network to discover the nature of the service for each traffic flow, classify it dynamically, and exercise traffic conditioning by means of such techniques as admission control and scheduling when delivering the traffic downstream to support the service appropriately.

Congestion at a network node can be aggravated by having too many TCP connections. TCP will adjust to try to share bandwidth among all connections but when the available buffer space is insufficient, time-outs will occur and as the congestion increases there will be an exponentially growing number of packets resent. The effect of having too many connections is that much of the bandwidth in the upstream network is wasted carrying packets that will be discarded at the congested node because there is not enough buffer there.

A simple method of avoiding the bad effects of too many TCP connections is to limit the number of connections or to discard one or more packets from one or more existing connections. Limiting the number of connections is achieved by an admission control which delays or even discards the connection set-up packets. In the case of discarding packets, which packets and from which connection to discard packets are decided by preset algorithms or policies. By invoking this control to limit the number of connections, each packet is inspected to see if it is a connection set-up packet, e.g., TCP SYN packet. This control packet is used to initiate a TCP connection and no traffic can flow until it is acknowledged by the other end of the proposed connection.

In one example, a decision to invoke the admission control, i.e. deciding when to limit the TCP traffic, can be made as follows:

Keep track of all TCP connections and thus keep count of the total number. Apply a calculation to see how many connections the available buffer can support and limit new connections. This is not a good way for a general implementation because it requires keeping state information on all TCP flows and being provided with information on the configured buffer size.

A better solution is when buffers get full and packet loss gets above some configured threshold, an admission control algorithm will apply some policy to reduce connections or the amount of traffic to keep the loss below the threshold. The reduction can be by discarding traffic from existing connections or, preferably, by preventing new connections from being set up.

The invention performs the admission control algorithm to achieve this effect.

OBJECTS OF INVENTION

It is therefore an object of the invention to provide a method of managing a data network for congestion.

It is a further object of the invention to provide a method of continuously monitoring the TCP traffic flows for congestion in a data network.

It is another object of the invention to provide a method of managing the data network by performing admission control for TCP traffic.

It is yet an object of the invention to provide a method of managing the data network by exercising the connection admission control for a new TCP connection request based on the packet loss characteristic.

SUMMARY OF THE INVENTION

Briefly stated, the invention resides in a packet data network for multimedia traffic having one or more nodes in which network one or more packets are discarded to control congestion. According to one aspect, a method of performing admission control to connection oriented traffic flows comprises steps of monitoring packets of all the traffic flows, deriving a packet loss characteristic of the traffic flows and disabling the serving of a new connection request when the packet loss characteristic matches a predefined pattern.

In another aspect, a method of performing admission control to TCP traffic flows comprises steps of storing all TCP connection setup packets in a connection request queue, monitoring packets of all active TCP traffic flows according to their port numbers and sequence numbers, and recording the count of either resent or discarded packets for any TCP traffic flows. The method further includes steps of building a history table containing the history of the sequence numbers, port numbers, and the count of either resent or discarded packets, computing a packet loss characteristic using the contents of the history table, and deciding enabling or disabling the connection request queue based on the packet loss characteristic with respect to a predefined pattern.

In a further aspect, the invention is directed to a TCP admission control apparatus for controlling congestion of a data network. The apparatus comprises a TCP output buffer for buffering and inspecting all the TCP packets of an incoming traffic flow, and a connection request queue for storing new connection requests. The apparatus further includes a history table for storing traffic information with respect to the TCP packets inspected above to derive a packet loss characteristic, and a queue controller for enabling or disabling the connection request queue upon detecting the matching of the packet loss characteristic with a predefined pattern.

BRIEF DESCRIPTION OF DRAWINGS

FIG. 1 is a schematic diagram of the admission control according to an embodiment of the invention.

FIGS. 2*a* and 2*b* are a flow chart for the case where TCP admission control is applied in a traffic link.

FIG. 3 illustrates the relationship of admission control with the traffic conditioner.

FIGS. 4*a* and 4*b* are a flow chart for the case where TCP admission control is applied in a router.

FIGS. 5 and 6 show possible locations of admission control of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS OF THE INVENTION

Referring to FIG. 1, the TCP admission control apparatus **10**, according to one embodiment of the invention, includes a connection request queue **12**. It is located at or near the output buffer **14** of a node of a data network. It should be noted that an admission control apparatus can be a separate device or can be made integral with or to reside in any node or link equipment. It should also be understood that TCP traffic flows as a whole can be processed by an apparatus or separate apparatus can be provided for each traffic flow or a group of traffic flows in one class. Every packet of an input stream is inspected and TCP packets are identified at the output buffer **1** using, for example, source and destination IP addresses, source and destination port numbers and protocol. All new connection requests are read at a connection reader **16** and are stored at the connection request queue **12**. The connection request queue **12** is a FIFO. If admission control is not invoked then the new connection requests will be served immediately by enabling the connection request queue. If admission control is switched on then they will be delayed.

The admission control detects the packets that are being discarded and looks for multiple successive packets from the same flow or multiple instances of the same packet, the latter being the result of packet resends due to packet loss or discard. The admission control derives some pattern of packet discards by using a discard measure. For convenience, this measure is called packet loss characteristic in this specification. It is possible that other parameters can be used to indicate the state of congestion in a data network. If certain criteria are met or the packet loss characteristic matches a predefined pattern, admission control is invoked and any new connection requests (connection set-up packets) will be delayed by disabling the connection request queue or packets belonging to one or more existing connections will be discarded until the problem clears. If a connection set-up packet is delayed too long (e.g., one second), it will be discarded from the queue.

When the packet loss characteristic shows that new connections can be accepted the servicing of the connection request queue is enabled. Waiting connection requests can be served immediately or can be released at a controlled pace according to a predefined algorithm.

The admission control apparatus therefore includes a small history table **18** and information about discarded packets is entered into it. When a packet is discarded, the flow identity (source and destination IP plus TCP socket number) is extracted and compared with current table entries. If the flow already has an entry then the history is updated. If the flow does not have an entry and there is room for a new entry, the new entry is made. If there is no room for a new entry the information is discarded.

The admission control can be performed on a traffic link or at a router.

In the case where the admission control is performed on the traffic link, the history table contains, for each active flow (or as many flows as can be handled), the following entries:

The first entry is a count of resent packets for that flow (Total Packet Resent).

The second entry is a count of how many times the currently recorded packet (that is the currently stored sequence number) has been resent (Same Packet Resent).

The third entry is the time that the most recent update was made for that flow. After some period of inactivity the flow is taken out of the table.

This information is used to look for patterns of discard that indicate congestion problems. It is assumed that if the sequence number on an arriving packet is lower than or equal to the stored value, then it must be a resend. The total number of resends as a fraction of the total number of packets is a measure of downstream congestion. In this embodiment, this measure is used as the packet loss characteristic.

Seeing the same packet resent multiple times will suggest that the connection is experiencing time-out or at least a very high loss rate. It is not usual for a packet to be discarded multiple times. Normally the TCP protocol will adjust its window to fit the available bandwidth and will only lose one packet before reducing that window. Although TCP relies on packet loss to constantly test for available bandwidth, a packet that is discarded once will almost certainly be forwarded when it is retransmitted. Multiple instances of the same packet will suggest that the TCP source is experiencing time-out.

There will be many variations on what information is stored and what algorithm is used to assess whether new connections should be enabled.

It is not necessary to keep information on all flows since a sampled history is sufficient to detect problem conditions.

Entries in the history table are removed after a period of time. Also, whenever admission control is invoked, the history table is cleaned out and starts fresh to get a good picture of the new loss characteristic. The history table would be purged, in any case, at regular intervals to keep the history reflecting current loss characteristics. The interval would be configurable depending on line rates and expected number of flows, etc.

FIGS. 2*a* and 2*b* are a flow chart for the case where TCP admission control is applied in a traffic link rather than in a router.

As mentioned earlier, the applicant's pending applications describe traffic conditioners and FIG. 3 shows one of such conditioners. In the Figure, a traffic conditioner **40** includes a plurality of queues **42**, at least one for each class of TCP traffic. Every packet of an input stream is inspected and identified at **44** using, for example, IP addresses, ports, etc. A controller **46** characterises the flow (using rate, duration, etc.) and assigns it a class. The controller refers to a database **48** and uses output scheduling to allocate bandwidth among classes. It can implement an admission control policy of the present invention for a class before delivering an output stream toward downstream nodes or to peripherals. In this case it is necessary to work out whether a packet has been discarded, by looking for a second copy of it passing through the link.

In another embodiment, the admission control is performed in the router where the discarded packets can be inspected directly as the discard decision is made at the buffer of the router.

In this case the history table contains, for each active flow (or as many flows as can be handled), the following entries:

The first entry is a count of discarded packets for that flow (Total Packet Discarded).

The second entry is a count of how many times the currently recorded packet (that is the currently stored sequence number) has been discarded (Same Packet Discarded).

The third entry is the time that the most recent update was made for that flow. After some period of inactivity the flow is taken out of the table.

This information is used to look for patterns of discard that indicate congestion problems. The total number of discards as a fraction of the total number of packets is a measure of buffer congestion.

Seeing the same packet resent multiple times will suggest that the connection is experiencing time-out or at least a very high loss rate.

There will be many variations on what information is stored and what algorithm is used to assess whether new connections should be enabled.

In another embodiment, if the admission control is performed at the router, packets from one or more existing connections can be discarded to control congestion at its buffer. The discarding action can be taken together with action of limiting the set-up of new connections, latter having been described above.

FIGS. 4*a* and 4*b* are a flow chart for the case where TCP admission control is applied in a router rather than in a traffic link.

Like the traffic conditioning of the pending applications, the admission control can take place at various places in the data network and can be biased toward certain kinds of TCP traffic. For example, as gateways are often a bottleneck and bulk flows can decrease response times for interactive users, an admission control can be located at a place shown in FIG. 5 which will alleviate this problem. In FIG. 6, traffic conditioners are located at a plurality of IP switches which form a data network **60**.

What is claimed is:

1. In a packet data network for multimedia traffic having one or more nodes in which network one or more packets are discarded to control congestion; a method of performing admission control to TCP traffic flows comprising steps of;

storing all TCP connection setup packets in a connection request queue;

monitoring packets of all active TCP traffic flows according to their port numbers and sequence numbers;

recording the count of either resent or discarded packets for any TCP traffic flows;

building a history table containing the history of the sequence numbers, port numbers, and the count of either resent or discarded packets;

computing a packet loss characteristic using the contents of the history table; and

deciding enabling or disabling the connection request queue based on the packet loss characteristic with respect to a predefined pattern.

2. The method of performing admission control to TCP traffic flows according to claim **1** wherein the step of computing a packet loss characteristic comprises step of:

deriving the total number of either resends or discards as a fraction of the total number of TCP packets of the TCP traffic flow.

3. The method performing admission control to TCP traffic flow according to claim **2**, comprising the further step of:

deciding to disable the connection request queue when the fraction reaches a preset threshold.

4. The method of performing admission control to TCP traffic flows according to claim **1**, comprising a further step of:

enabling the connection request queue at a controlled pace.

5. A TCP admission control apparatus for controlling congestion of a data network, comprising:

a TCP output buffer for inspecting all the TCP packets of an incoming traffic stream according to their port numbers and sequence numbers;

a connection request queue for storing new connection requests;

a history table for recording the sequence numbers, port numbers and a count of either recent or discarded packets in order to compute a packet loss characteristic; and

a queue controller for enabling or disabling the connection request upon detecting the matching of the packet loss characteristic with a predefined pattern.

6. The TCP admission control apparatus according to claim **5** wherein the history table contains the total number of packets of the TCP traffic flow.

7. The method of performing admission control to TCP traffic flows according to claim **1** wherein the step of recording further comprises recording the time that the most recent update was made for a specified TCP traffic flow.

8. The method of performing admission control to TCP traffic flows according to claim **7** wherein the specified TCP traffic flow is removed from the history table after a predefined period of inactivity.

9. The method of performing admission control to TCP traffic flows according to claim **1** wherein the method is performed in a router.

10. The method of performing admission control to TCP traffic flows according to claim **1** wherein the method is performed in a controller integral to a traffic conditioner.

11. The method of performing admission control to TCP traffic flows according to claim **1** further comprising the step of clearing all entries of the history table whenever the connection request queue is re-enabled.

12. The method of performing admission control to TCP traffic flows according to claim **1** further comprising the step of purging all entries in the history table periodically from time to time or after a certain preset period.

13. The TCP admission control apparatus according to claim **5** wherein the history table records the time that the most recent update was made for a specified TCP traffic flow.

14. The TCP admission control apparatus according to claim **13** wherein the specified TCP traffic flow is removed from the history table after a predefined period of inactivity.

15. The TCP admission control apparatus according to claim **5** wherein the apparatus is a router.

16. The TCP admission control apparatus according to claim **5** wherein the history table clears all entries whenever the connection request queue is re-enabled.

17. The TCP admission control apparatus according to claim **5** wherein the history table all entries periodically from time to time or after a certain preset period.

18. The TCP admission control apparatus according to claim **5** wherein the packet loss characteristic is computed by deriving the total number of either resends or discards as a fraction of the total number of TCP packets of the TCP traffic flow.

19. The TCP admission control apparatus according to claim **18** wherein the queue controller disables the connection request queue when the fraction reaches a preset threshold.

20. The TCP admission control apparatus according to claim **5** wherein the connection request queue is enabled at a controlled pace.

* * * * *

US006651099B1

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

(10) **Patent No.: US 6,651,099 B1**
(45) **Date of Patent: Nov. 18, 2003**

(54) **METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK**

(75) Inventors: **Russell S. Dietz**, San Jose, CA (US); **Joseph R. Maixner**, Aptos, CA (US); **Andrew A. Koppenhaver**, Littleton, CO (US); **William H. Bares**, Germantown, TN (US); **Haig A. Sarkissian**, San Antonio, TX (US); **James F. Torgerson**, Andover, MN (US)

(73) Assignee: **Hi/fn, Inc.**, Los Gatos, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 589 days.

(21) Appl. No.: **09/608,237**

(22) Filed: **Jun. 30, 2000**

Related U.S. Application Data

(60) Provisional application No. 60/141,903, filed on Jun. 30, 1999.

(51) **Int. Cl.**^{7} **G06F 13/00**

(52) **U.S. Cl.** **709/224**; 370/389

(58) **Field of Search** 709/200, 201, 709/220, 223, 224, 231, 232, 236, 238, 239, 240, 246; 370/389, 392, 395.32

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,736,320	A	4/1988	Bristol	364/300
4,891,639	A	1/1990	Nakamura	340/825.5
5,101,402	A	3/1992	Chui et al.	370/17
5,247,517	A	9/1993	Ross et al.	370/85.5
5,247,693	A	9/1993	Bristol	395/800
5,249,292	A	9/1993	Chiappa	395/650
5,315,580	A	5/1994	Phaal	370/13
5,339,268	A	8/1994	Machida	365/49
5,351,243	A	9/1994	Kalkunte et al.	370/92
5,365,514	A	11/1994	Hershey et al.	370/17
5,375,070	A	12/1994	Hershey et al.	364/550
5,394,394	A	2/1995	Crowther et al.	370/60

(List continued on next page.)

OTHER PUBLICATIONS

"Technical Note: the Narus System," Downloaded Apr. 29, 1999 from www.narus.com, Narus Corporation, Redwood City California.

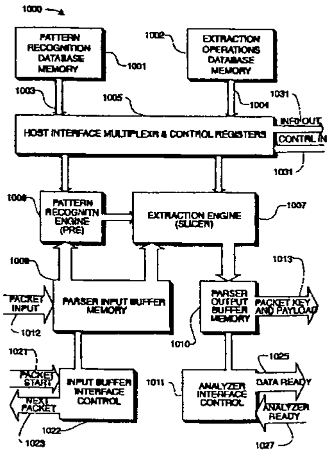
Primary Examiner—Moustafa M. Meky
(74) *Attorney, Agent, or Firm*—Dov Rosenfeld; Inventek

(57) **ABSTRACT**

A monitor for and a method of examining packets passing through a connection point on a computer network. Each packets conforms to one or more protocols. The method includes receiving a packet from a packet acquisition device and performing one or more parsing/extraction operations on the packet to create a parser record comprising a function of selected portions of the packet. The parsing/extraction operations depend on one or more of the protocols to which the packet conforms. The method further includes looking up a flow-entry database containing flow-entries for previously encountered conversational flows. The lookup uses the selected packet portions and determining if the packet is of an existing flow. If the packet is of an existing flow, the method classifies the packet as belonging to the found existing flow, and if the packet is of a new flow, the method stores a new flow-entry for the new flow in the flow-entry database, including identifying information for future packets to be identified with the new flow-entry. For the packet of an existing flow, the method updates the flow-entry of the existing flow. Such updating may include storing one or more statistical measures. Any stage of a flow, state is maintained, and the method performs any state processing for an identified state to further the process of identifying the flow. The method thus examines each and every packet passing through the connection point in real time until the application program associated with the conversational flow is determined.

10 Claims, 18 Drawing Sheets

Statistic Flow
every packet [illegible]
state of flow
[illegible]


1000
PATTERN RECOGNITION DATABASE MEMORY
1001
1002
EXTRACTION OPERATIONS DATABASE MEMORY
1003
1005
1004
1031
INFO OUT
CONTRL IN
HOST INTERFACE MULTIPLEXR & CONTROL REGISTERS
1031
1006
PATTERN RECOGNITN ENGINE (PRE)
EXTRACTION ENGINE (SLICER)
1007
1008
1013
PACKET INPUT
PARSER INPUT BUFFER MEMORY
PARSER OUTPUT BUFFER MEMORY
PACKET KEY AND PAYLOAD
1012
1010
1021
1025
PACKET START
INPUT BUFFER INTERFACE CONTROL
1011
ANALYZER INTERFACE CONTROL
DATA READY
NEXT PACKET
ANALYZER READY
1022
1023
1027

[illegible]

U.S. PATENT DOCUMENTS

5,414,650	A		5/1995	Hekhuis	364/715.02
5,414,704	A		5/1995	Spinney	370/60
5,430,709	A		7/1995	Galloway	370/13
5,432,776	A		7/1995	Harper	370/17
5,493,689	A		2/1996	Waclawsky et al.	395/821
5,500,855	A		3/1996	Hershey et al.	370/17
5,511,213	A		4/1996	Correa	395/800
5,511,215	A		4/1996	Terasaka et al.	395/800
5,568,471	A		10/1996	Hershey et al.	370/17
5,574,875	A		11/1996	Stansfield et al.	395/403
5,586,266	A		12/1996	Hershey et al.	395/200.11
5,606,668	A		2/1997	Shwed	395/200.11
5,608,662	A		3/1997	Large et al.	364/724.01
5,634,009	A		5/1997	Iddon et al.	395/200.11
5,651,002	A		7/1997	Van Seters et al.	370/392
5,684,954	A		11/1997	Kaiserswerth et al.	395/200.2
5,703,877	A		12/1997	Nuber et al.	370/395
5,732,213	A		3/1998	Gessel et al.	395/200.11
5,740,355	A		4/1998	Watanabe et al.	395/183.21
5,761,424	A		6/1998	Adams et al.	395/200.47
5,764,638	A		6/1998	Ketchum	370/401
5,781,735	A		7/1998	Southard	395/200.54
5,784,298	A		7/1998	Hershey et al.	364/557
5,787,253	A		7/1998	McCreery et al.	395/200.61
5,802,054	A		9/1998	Bellenger	370/351
5,805,808	A		9/1998	Hansani et al.	395/200.2
5,812,529	A		9/1998	Czarnik et al.	370/245
5,819,028	A		10/1998	Manghirmalani et al.	395/185.1
5,825,774	A		10/1998	Ready et al.	370/401
5,835,726	A		11/1998	Shwed et al.	395/200.59
5,838,919	A		11/1998	Schwaller et al.	395/200.54
5,841,895	A		11/1998	Huffman	382/155
5,850,386	A		12/1998	Anderson et al.	370/241
5,850,388	A		12/1998	Anderson et al.	370/252
5,862,335	A		1/1999	Welch, Jr. et al.	395/200.54
5,878,420	A		3/1999	de la Salle	707/10
5,893,155	A		4/1999	Cheriton	711/144
5,903,754	A		5/1999	Pearson	395/680
5,917,821	A		6/1999	Gobuyan et al.	370/392
6,014,380	A		1/2000	Hendel et al.	370/392
6,118,760	A	*	9/2000	Zaumen et al.	370/229
6,243,667	B1	*	6/2001	Kerr et al.	703/27
6,452,915	B1	*	9/2002	Jorgensen	370/338
6,453,360	B1	*	9/2002	Muller et al.	709/250
6,466,985	B1	*	10/2002	Goyal et al.	709/238
6,483,804	B1	*	11/2002	Muller et al.	370/230
6,570,875	B1	*	5/2003	Hegde	370/389

* cited by examiner

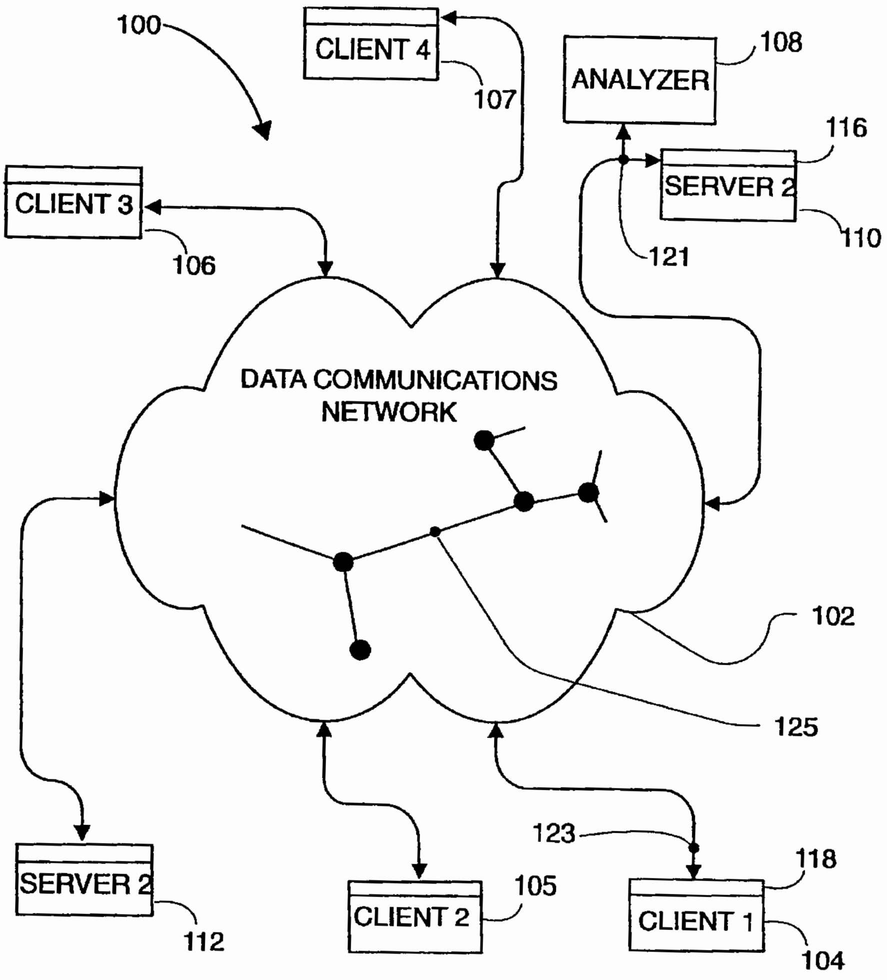

100
CLIENT 4
107
ANALYZER
108
116
SERVER 2
110
121
CLIENT 3
106
DATA COMMUNICATIONS NETWORK
102
125
123
118
SERVER 2
112
CLIENT 2
105
CLIENT 1
104

FIG. 1

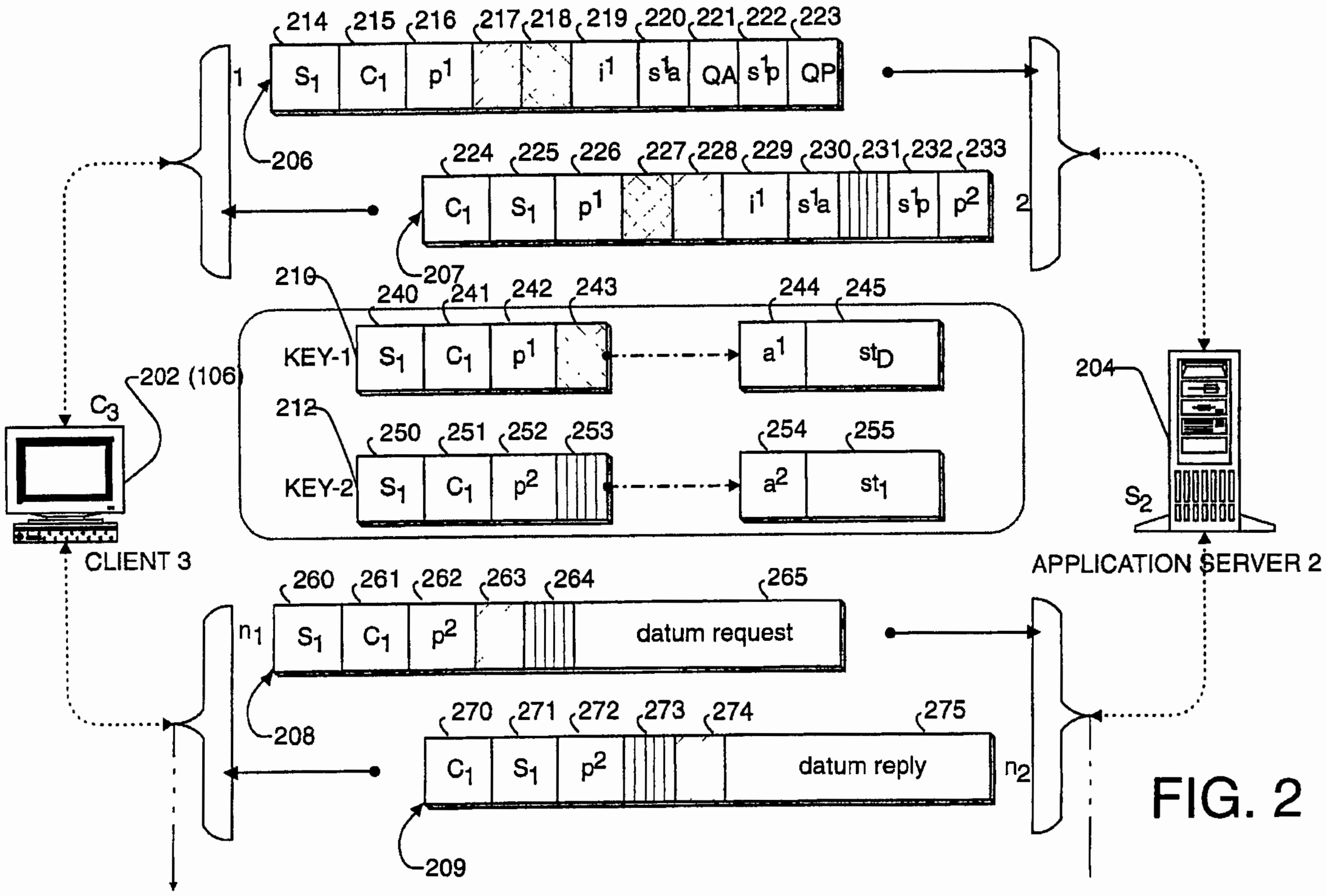

214
215
216
217
218
219
220
221
222
223
1
S1
C1
p1
i1
s1a
QA
s1p
QP
206
224
225
226
227
228
229
230
231
232
233
C1
S1
p1
i1
s1a
s1p
p2
2
207
210
240
241
242
243
244
245
KEY-1
S1
C1
p1
a1
stD
212
250
251
252
253
254
255
KEY-2
S1
C1
p2
a2
st1
202 (106)
C3
CLIENT 3
204
S2
APPLICATION SERVER 2
260
261
262
263
264
265
n1
S1
C1
p2
datum request
208
270
271
272
273
274
275
C1
S1
p2
datum reply
n2
209

FIG. 2

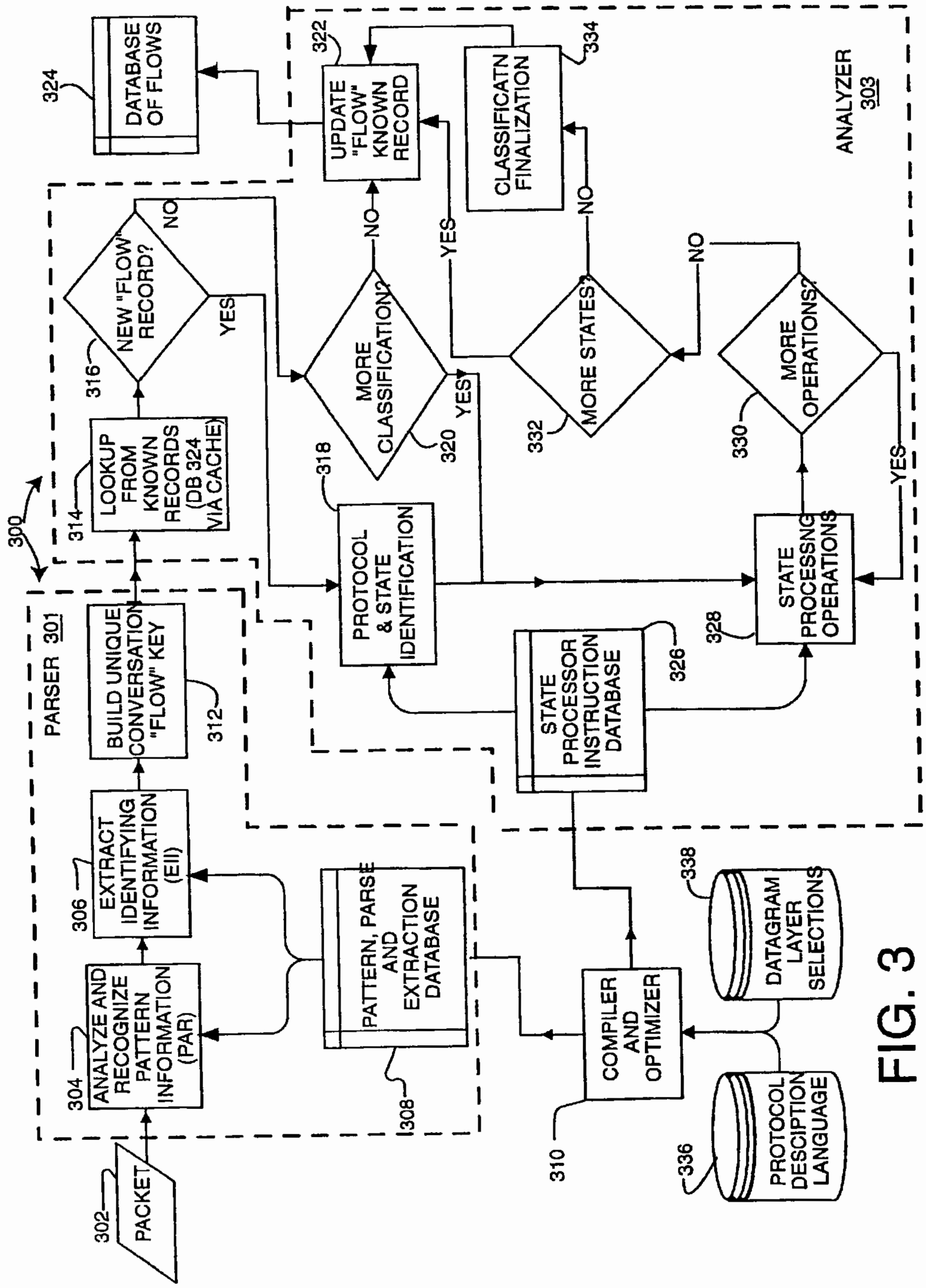

300
PARSER 301
ANALYZER 303
302
PACKET
304
ANALYZE AND RECOGNIZE PATTERN INFORMATION (PAR)
306
EXTRACT IDENTIFYING INFORMATION (EII)
312
BUILD UNIQUE CONVERSATION "FLOW" KEY
308
PATTERN, PARSE AND EXTRACTION DATABASE
314
LOOKUP FROM KNOWN RECORDS (DB 324 VIA CACHE)
316
NEW "FLOW" RECORD?
YES
NO
318
PROTOCOL & STATE IDENTIFICATION
320
MORE CLASSIFICATION?
YES
NO
322
UPDATE "FLOW" KNOWN RECORD
324
DATABASE OF FLOWS
326
STATE PROCESSOR INSTRUCTION DATABASE
328
STATE PROCESSING OPERATIONS
330
MORE OPERATIONS?
YES
NO
332
MORE STATES?
YES
NO
334
CLASSIFICATN FINALIZATION
310
COMPILER AND OPTIMIZER
336
PROTOCOL DESCIPTION LANGUAGE
338
DATAGRAM LAYER SELECTIONS

FIG. 3

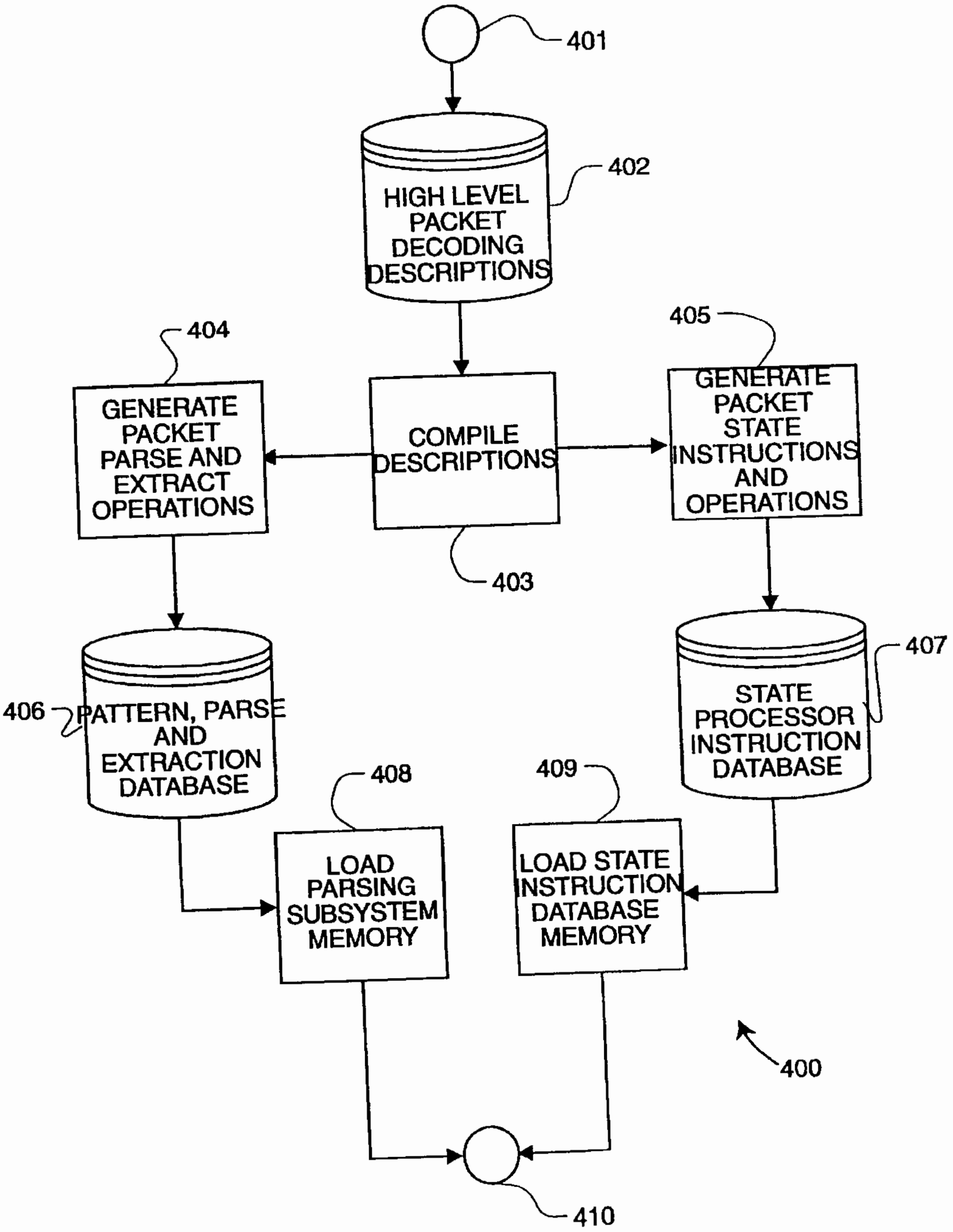

401
HIGH LEVEL PACKET DECODING DESCRIPTIONS
402
COMPILE DESCRIPTIONS
403
GENERATE PACKET PARSE AND EXTRACT OPERATIONS
404
GENERATE PACKET STATE INSTRUCTIONS AND OPERATIONS
405
PATTERN, PARSE AND EXTRACTION DATABASE
406
STATE PROCESSOR INSTRUCTION DATABASE
407
LOAD PARSING SUBSYSTEM MEMORY
408
LOAD STATE INSTRUCTION DATABASE MEMORY
409
400
410

FIG. 4

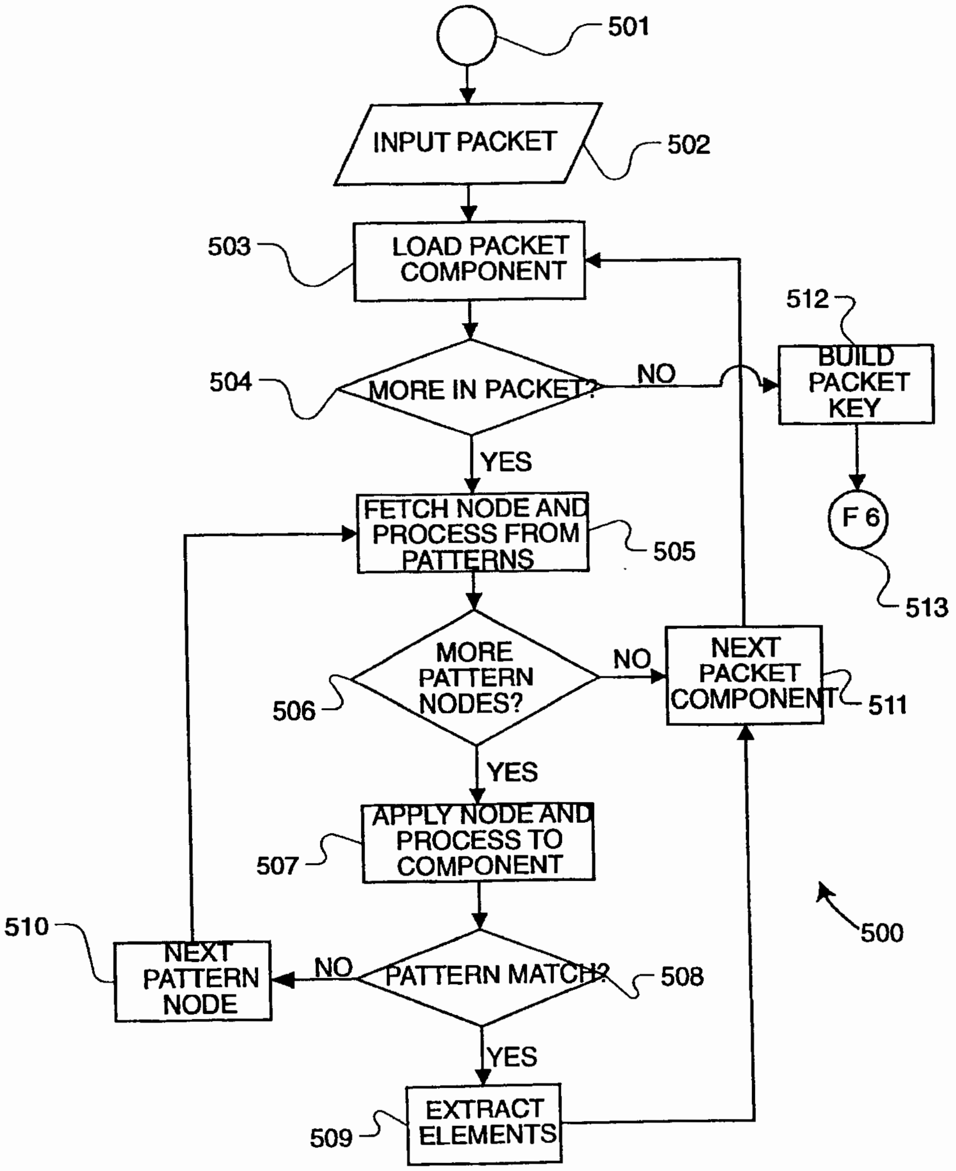

501
INPUT PACKET
502
503
LOAD PACKET COMPONENT
512
BUILD PACKET KEY
504
MORE IN PACKET?
NO
YES
F 6
513
FETCH NODE AND PROCESS FROM PATTERNS
505
MORE PATTERN NODES?
NO
NEXT PACKET COMPONENT
511
506
YES
APPLY NODE AND PROCESS TO COMPONENT
507
500
510
NEXT PATTERN NODE
NO
PATTERN MATCH?
508
YES
EXTRACT ELEMENTS
509

FIG. 5

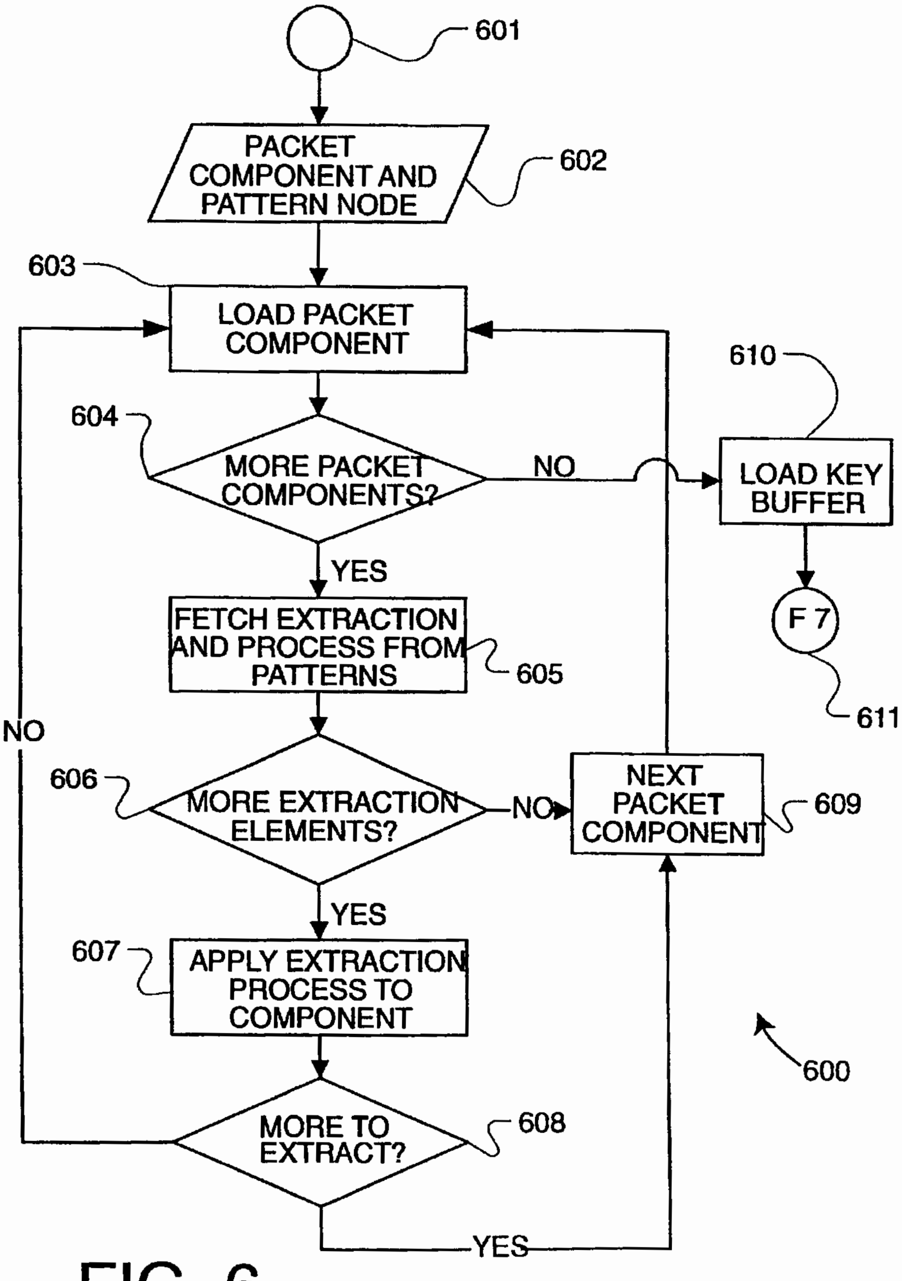

601
PACKET COMPONENT AND PATTERN NODE
602
603
LOAD PACKET COMPONENT
604
MORE PACKET COMPONENTS?
NO
610
LOAD KEY BUFFER
F 7
611
YES
FETCH EXTRACTION AND PROCESS FROM PATTERNS
605
NO
606
MORE EXTRACTION ELEMENTS?
NO
NEXT PACKET COMPONENT
609
YES
607
APPLY EXTRACTION PROCESS TO COMPONENT
600
MORE TO EXTRACT?
608
YES

FIG. 6

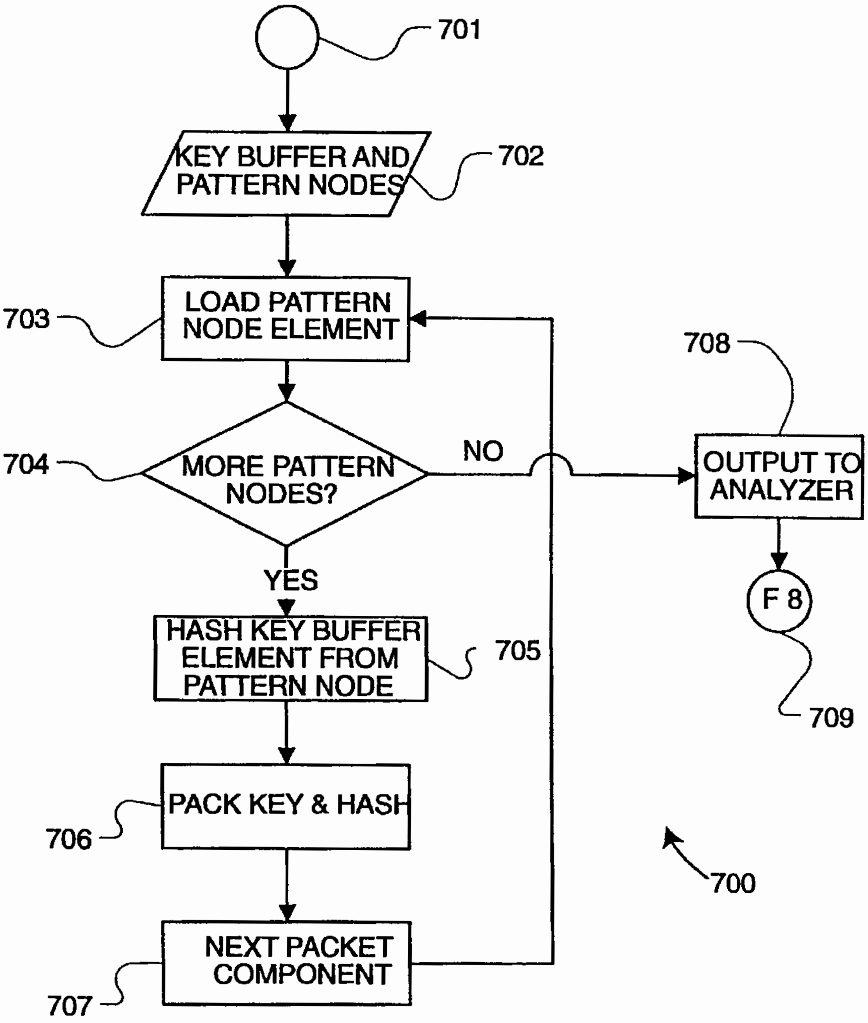

701
KEY BUFFER AND PATTERN NODES
702
LOAD PATTERN NODE ELEMENT
703
708
MORE PATTERN NODES?
704
NO
OUTPUT TO ANALYZER
YES
HASH KEY BUFFER ELEMENT FROM PATTERN NODE
705
F 8
709
PACK KEY & HASH
706
700
NEXT PACKET COMPONENT
707

FIG. 7

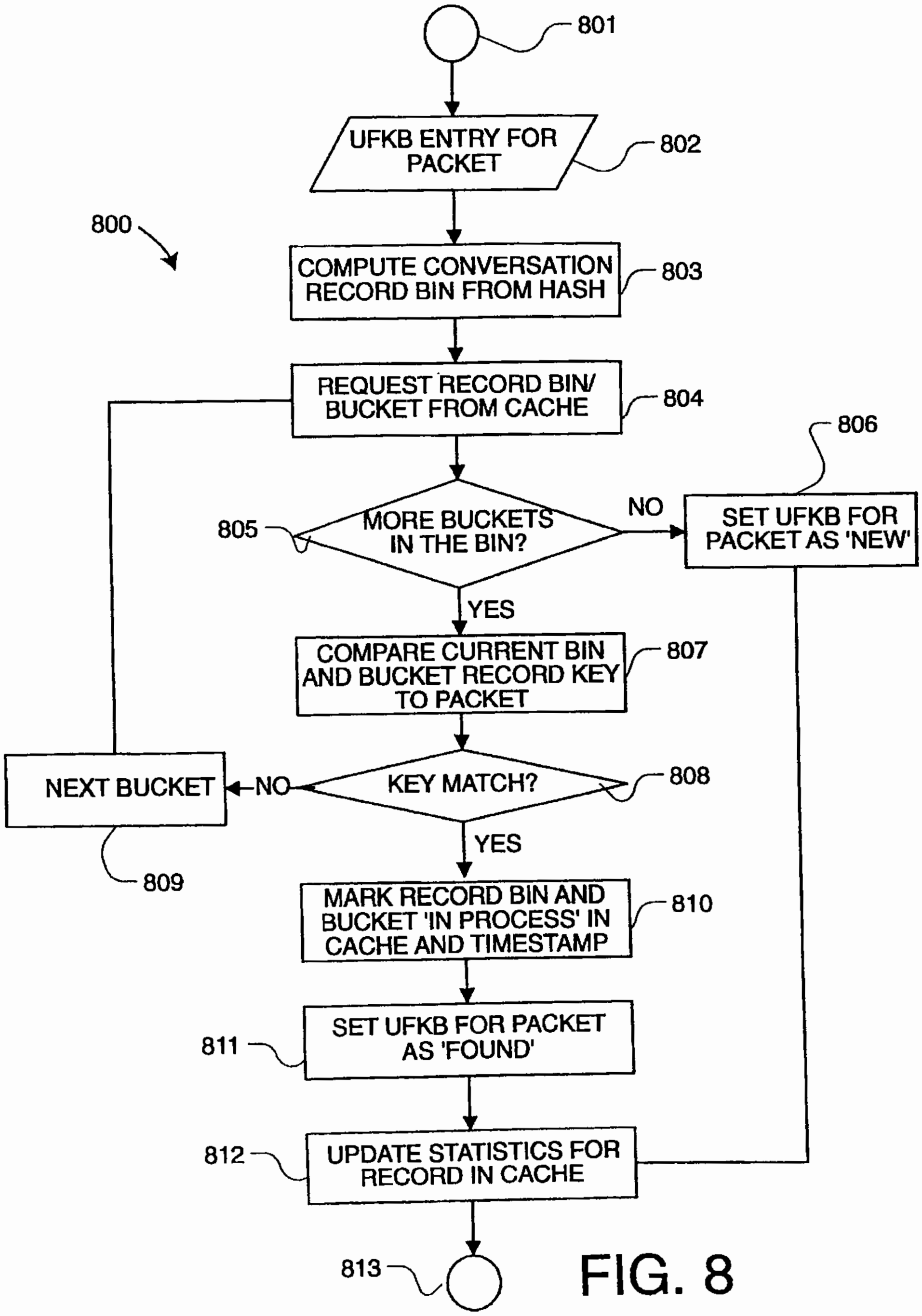

801
UFKB ENTRY FOR PACKET
802
800
COMPUTE CONVERSATION RECORD BIN FROM HASH
803
REQUEST RECORD BIN/ BUCKET FROM CACHE
804
806
805
MORE BUCKETS IN THE BIN?
NO
SET UFKB FOR PACKET AS 'NEW'
YES
COMPARE CURRENT BIN AND BUCKET RECORD KEY TO PACKET
807
NEXT BUCKET
NO
KEY MATCH?
808
809
YES
MARK RECORD BIN AND BUCKET 'IN PROCESS' IN CACHE AND TIMESTAMP
810
811
SET UFKB FOR PACKET AS 'FOUND'
812
UPDATE STATISTICS FOR RECORD IN CACHE
813

FIG. 8

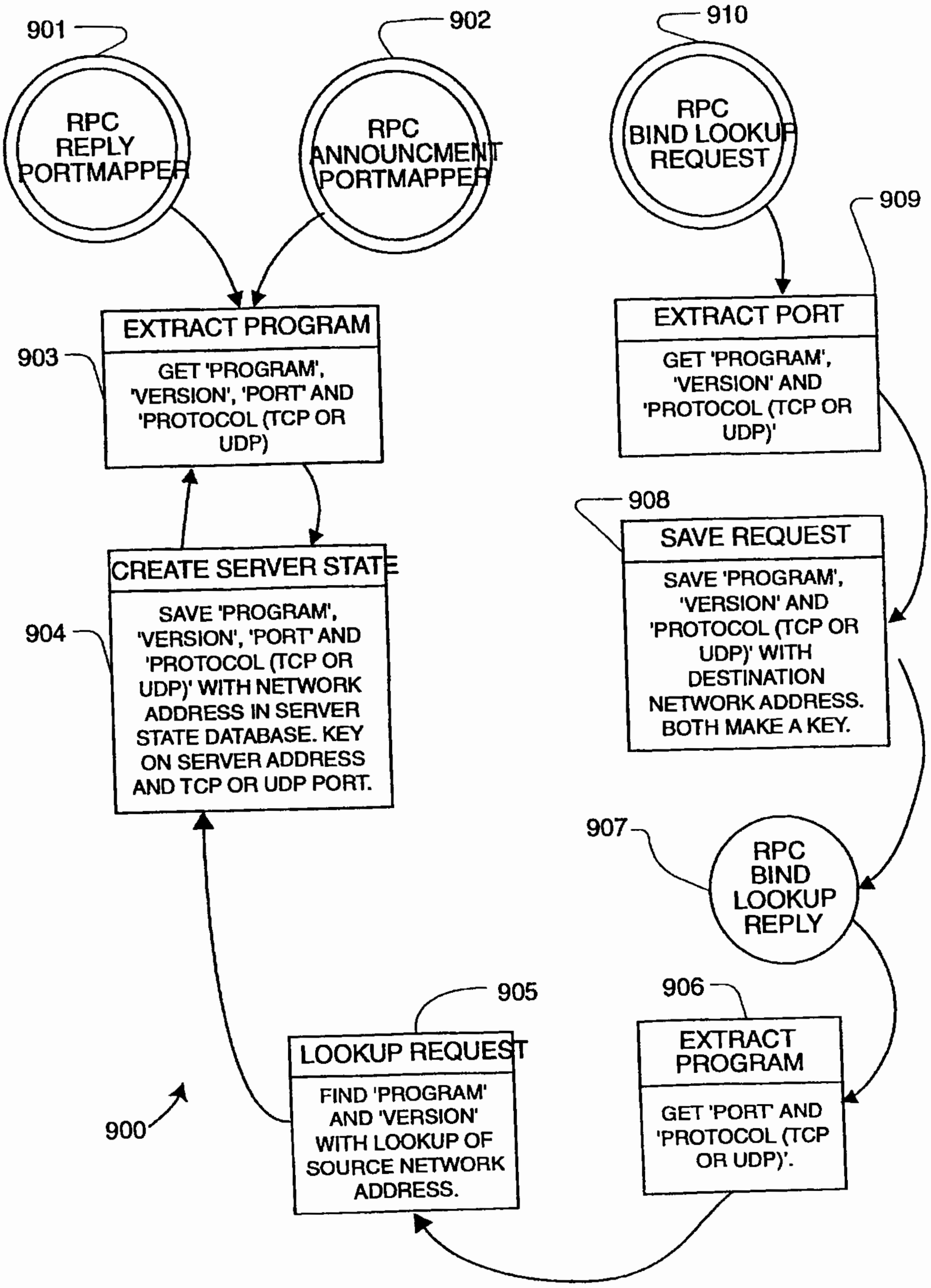

901
RPC REPLY PORTMAPPER
902
RPC ANNOUNCMENT PORTMAPPER
910
RPC BIND LOOKUP REQUEST
909
903
EXTRACT PROGRAM
GET 'PROGRAM', 'VERSION', 'PORT' AND 'PROTOCOL (TCP OR UDP)
EXTRACT PORT
GET 'PROGRAM', 'VERSION' AND 'PROTOCOL (TCP OR UDP)'
908
SAVE REQUEST
SAVE 'PROGRAM', 'VERSION' AND 'PROTOCOL (TCP OR UDP)' WITH DESTINATION NETWORK ADDRESS. BOTH MAKE A KEY.
904
CREATE SERVER STATE
SAVE 'PROGRAM', 'VERSION', 'PORT' AND 'PROTOCOL (TCP OR UDP)' WITH NETWORK ADDRESS IN SERVER STATE DATABASE. KEY ON SERVER ADDRESS AND TCP OR UDP PORT.
907
RPC BIND LOOKUP REPLY
905
906
LOOKUP REQUEST
FIND 'PROGRAM' AND 'VERSION' WITH LOOKUP OF SOURCE NETWORK ADDRESS.
EXTRACT PROGRAM
GET 'PORT' AND 'PROTOCOL (TCP OR UDP)'.
900

FIG. 9

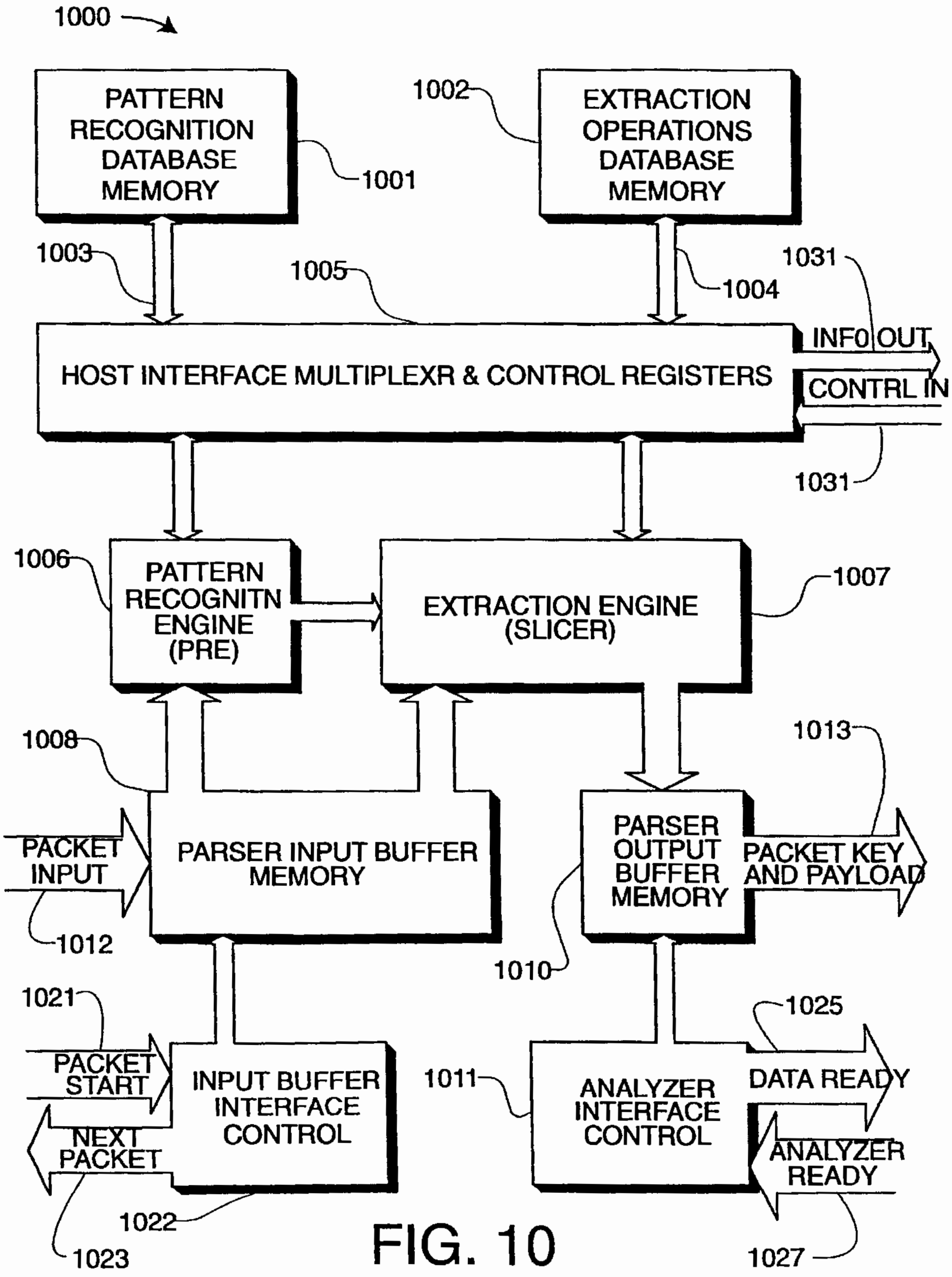

1000
PATTERN RECOGNITION DATABASE MEMORY
1001
1002
EXTRACTION OPERATIONS DATABASE MEMORY
1003
1005
1004
1031
HOST INTERFACE MULTIPLEXR & CONTROL REGISTERS
INFO OUT
CONTRL IN
1031
1006
PATTERN RECOGNITN ENGINE (PRE)
EXTRACTION ENGINE (SLICER)
1007
1008
1013
PACKET INPUT
PARSER INPUT BUFFER MEMORY
PARSER OUTPUT BUFFER MEMORY
PACKET KEY AND PAYLOAD
1012
1010
1021
1025
PACKET START
INPUT BUFFER INTERFACE CONTROL
1011
ANALYZER INTERFACE CONTROL
DATA READY
NEXT PACKET
ANALYZER READY
1022
1023
1027

FIG. 10

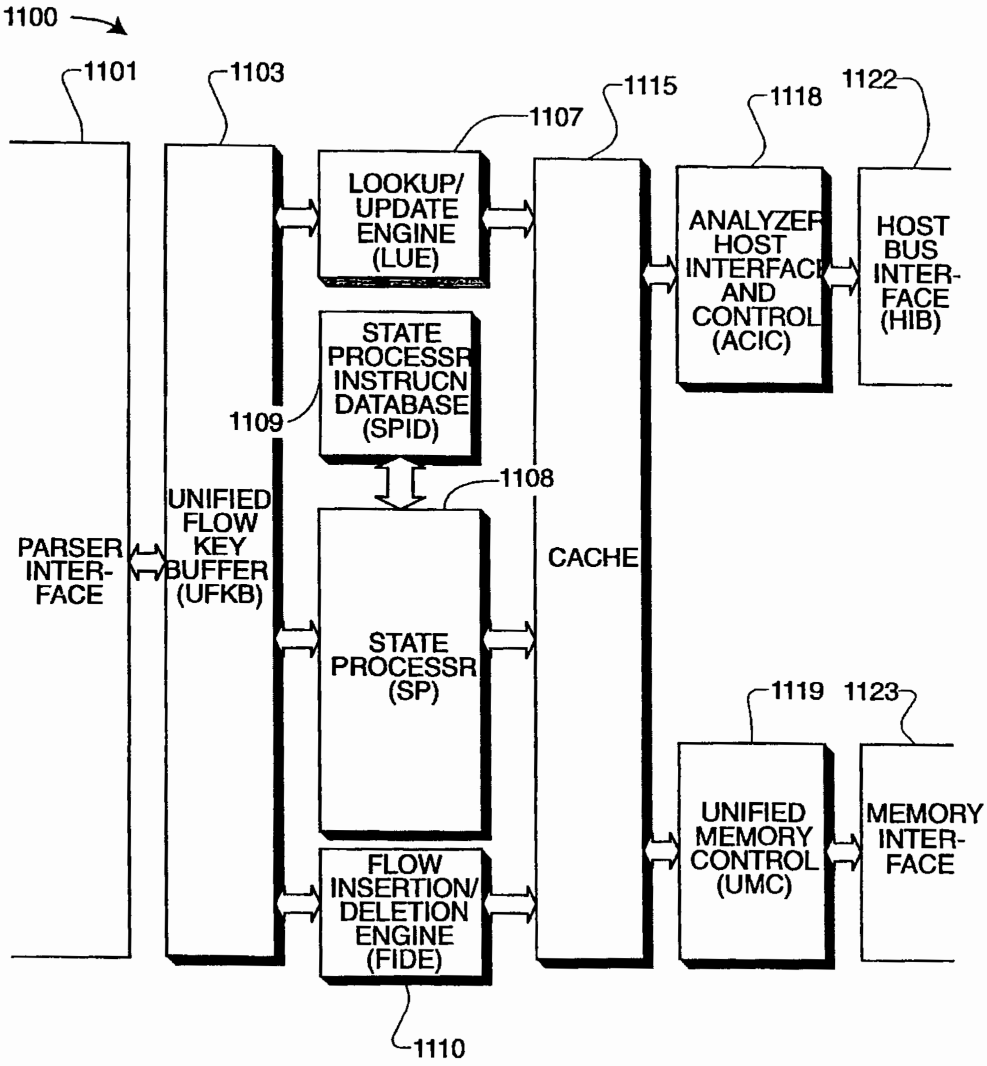

1100
1101
1103
1107
1115
1118
1122
LOOKUP/ UPDATE ENGINE (LUE)
ANALYZER HOST INTERFACE AND CONTROL (ACIC)
HOST BUS INTER-FACE (HIB)
STATE PROCESSR INSTRUCN DATABASE (SPID)
1109
1108
PARSER INTER-FACE
UNIFIED FLOW KEY BUFFER (UFKB)
CACHE
STATE PROCESSR (SP)
1119
1123
UNIFIED MEMORY CONTROL (UMC)
MEMORY INTER-FACE
FLOW INSERTION/ DELETION ENGINE (FIDE)
1110

FIG. 11

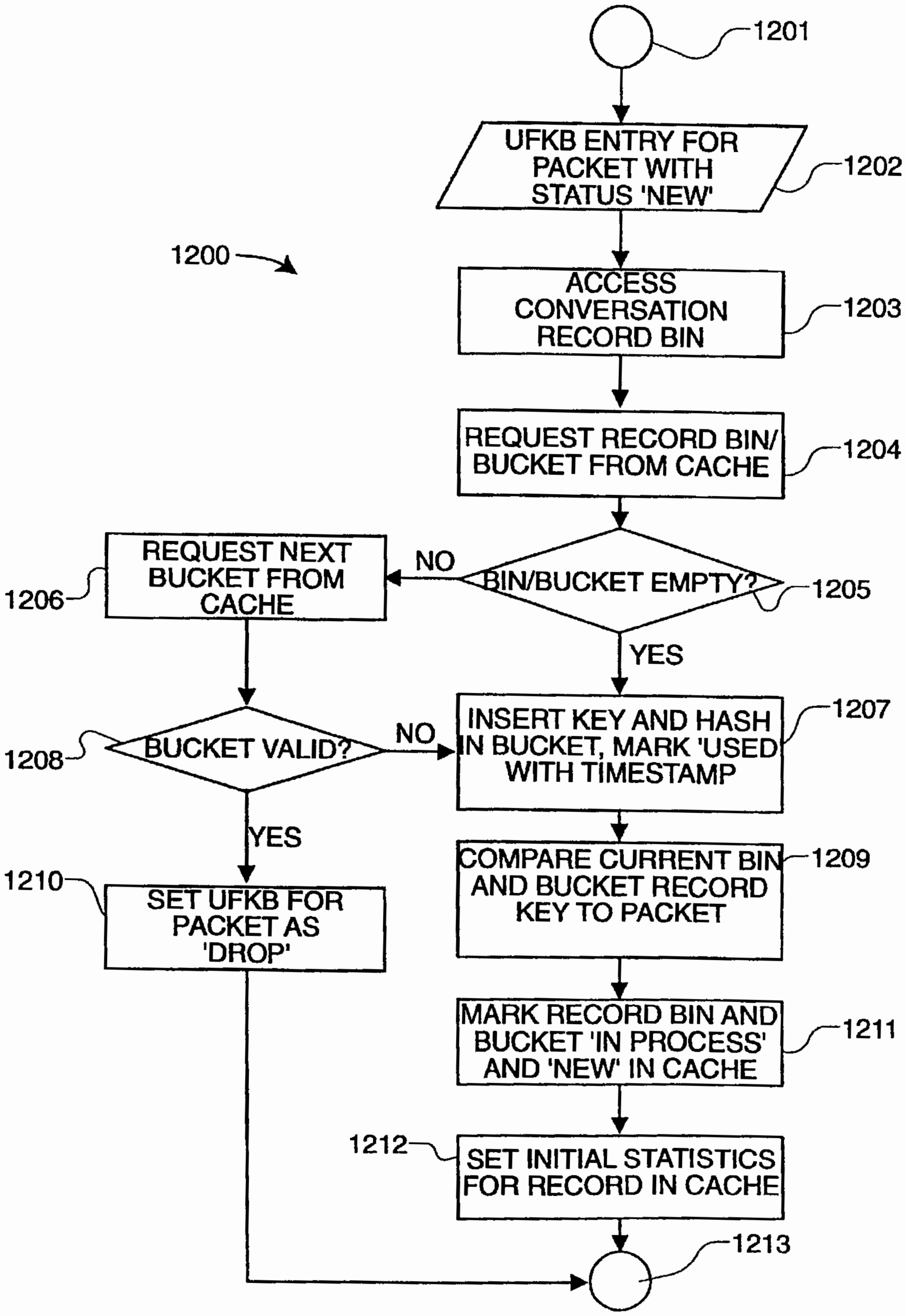

1201
UFKB ENTRY FOR PACKET WITH STATUS 'NEW'
1202
1200
ACCESS CONVERSATION RECORD BIN
1203
REQUEST RECORD BIN/ BUCKET FROM CACHE
1204
BIN/BUCKET EMPTY?
1205
NO
REQUEST NEXT BUCKET FROM CACHE
1206
YES
BUCKET VALID?
1208
NO
INSERT KEY AND HASH IN BUCKET, MARK 'USED WITH TIMESTAMP
1207
YES
SET UFKB FOR PACKET AS 'DROP'
1210
COMPARE CURRENT BIN AND BUCKET RECORD KEY TO PACKET
1209
MARK RECORD BIN AND BUCKET 'IN PROCESS' AND 'NEW' IN CACHE
1211
1212
SET INITIAL STATISTICS FOR RECORD IN CACHE
1213

FIG. 12

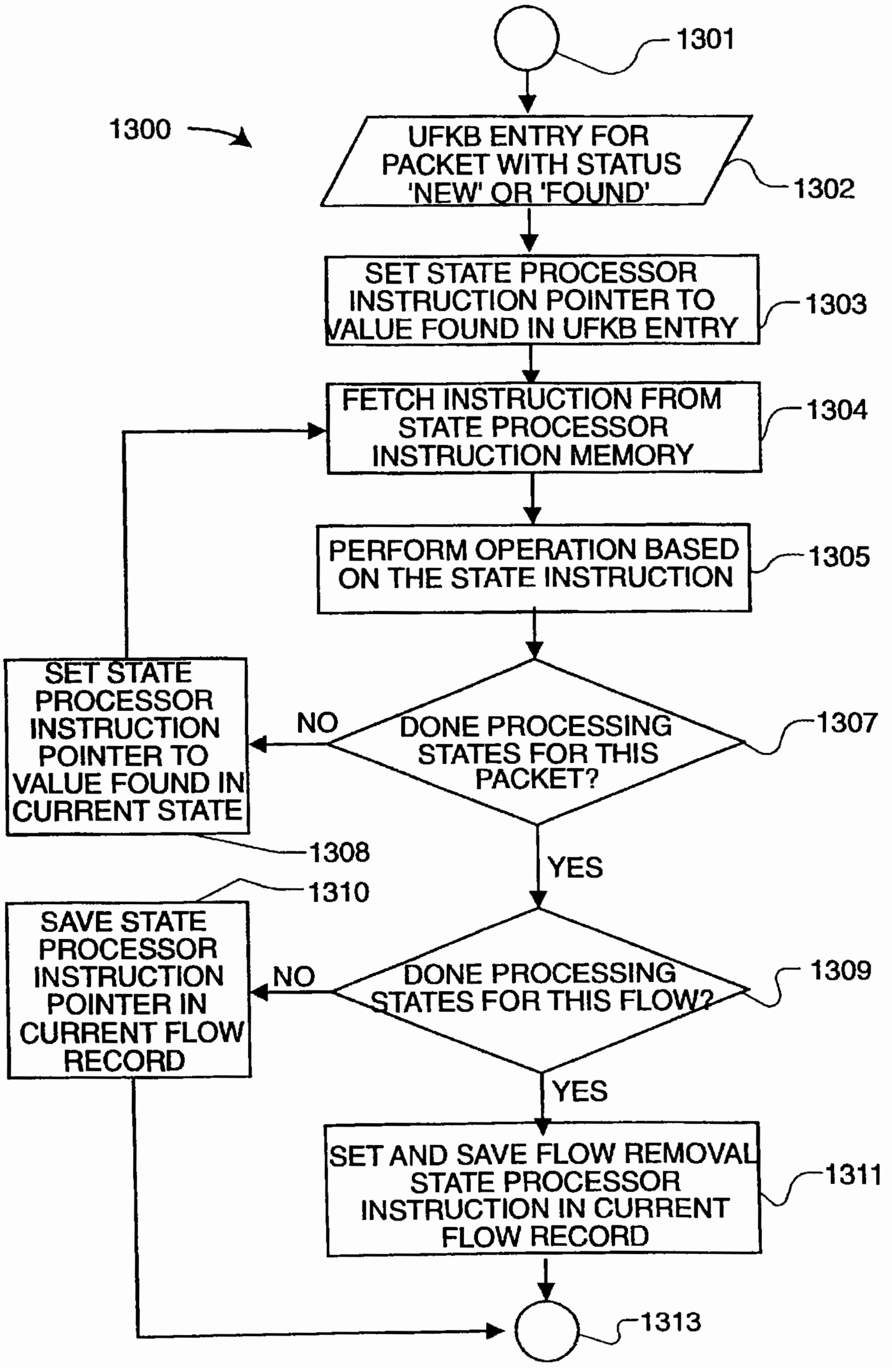

1301
1300
UFKB ENTRY FOR PACKET WITH STATUS 'NEW' OR 'FOUND'
1302
SET STATE PROCESSOR INSTRUCTION POINTER TO VALUE FOUND IN UFKB ENTRY
1303
FETCH INSTRUCTION FROM STATE PROCESSOR INSTRUCTION MEMORY
1304
PERFORM OPERATION BASED ON THE STATE INSTRUCTION
1305
SET STATE PROCESSOR INSTRUCTION POINTER TO VALUE FOUND IN CURRENT STATE
NO
DONE PROCESSING STATES FOR THIS PACKET?
1307
1308
YES
1310
SAVE STATE PROCESSOR INSTRUCTION POINTER IN CURRENT FLOW RECORD
NO
DONE PROCESSING STATES FOR THIS FLOW?
1309
YES
SET AND SAVE FLOW REMOVAL STATE PROCESSOR INSTRUCTION IN CURRENT FLOW RECORD
1311
1313

FIG. 13

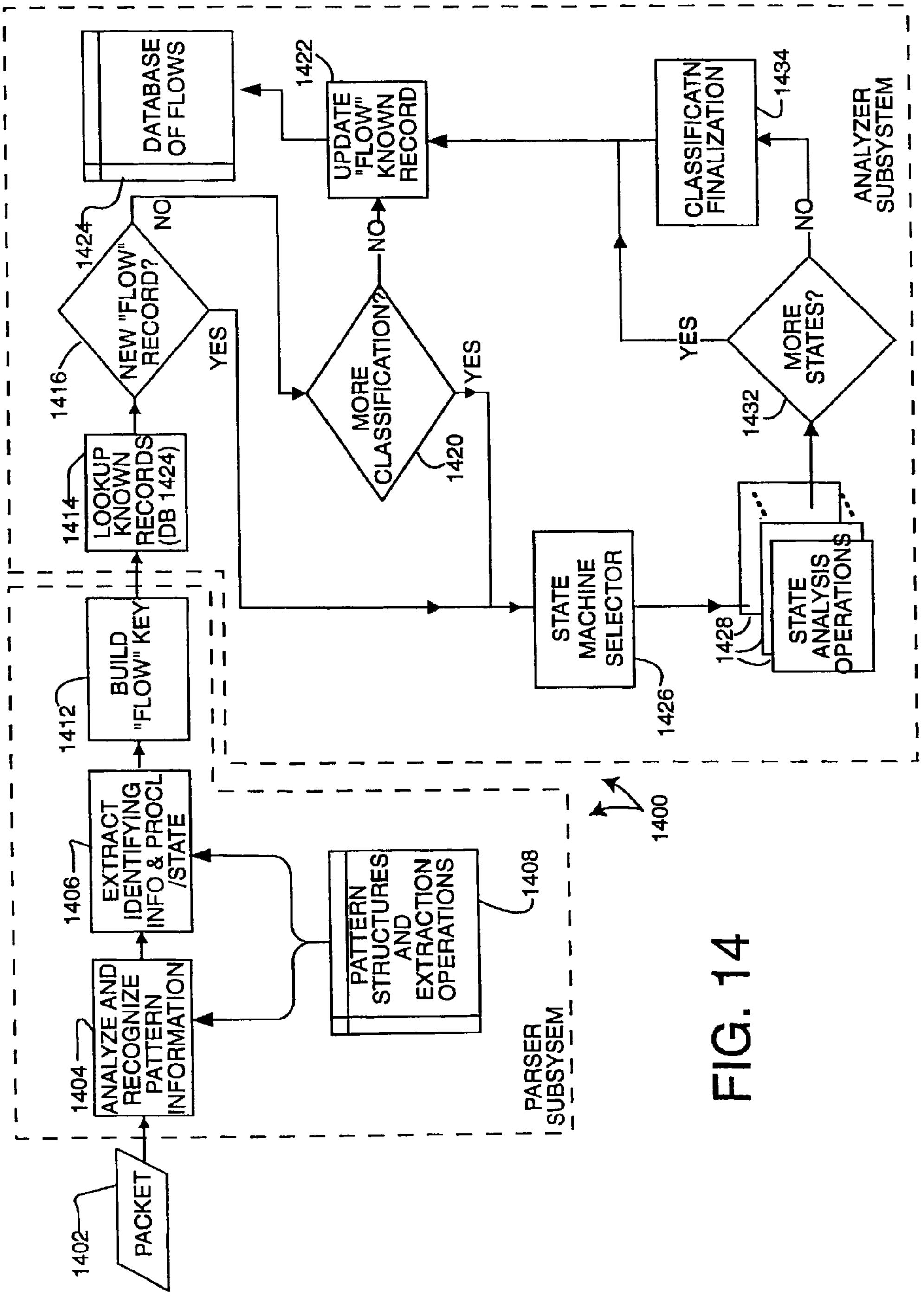

PACKET
1402
ANALYZE AND RECOGNIZE PATTERN INFORMATION
1404
EXTRACT IDENTIFYING INFO & PROCL /STATE
1406
PATTERN STRUCTURES AND EXTRACTION OPERATIONS
1408
PARSER SUBSYSEM
BUILD "FLOW" KEY
1412
LOOKUP KNOWN RECORDS (DB 1424)
1414
NEW "FLOW" RECORD?
1416
YES
NO
DATABASE OF FLOWS
1424
MORE CLASSIFICATION?
1420
YES
NO
UPDATE "FLOW" KNOWN RECORD
1422
STATE MACHINE SELECTOR
1426
STATE ANALYSIS OPERATIONS
1428
MORE STATES?
1432
YES
NO
CLASSIFICATN FINALIZATION
1434
ANALYZER SUBSYSTEM
1400

FIG. 14

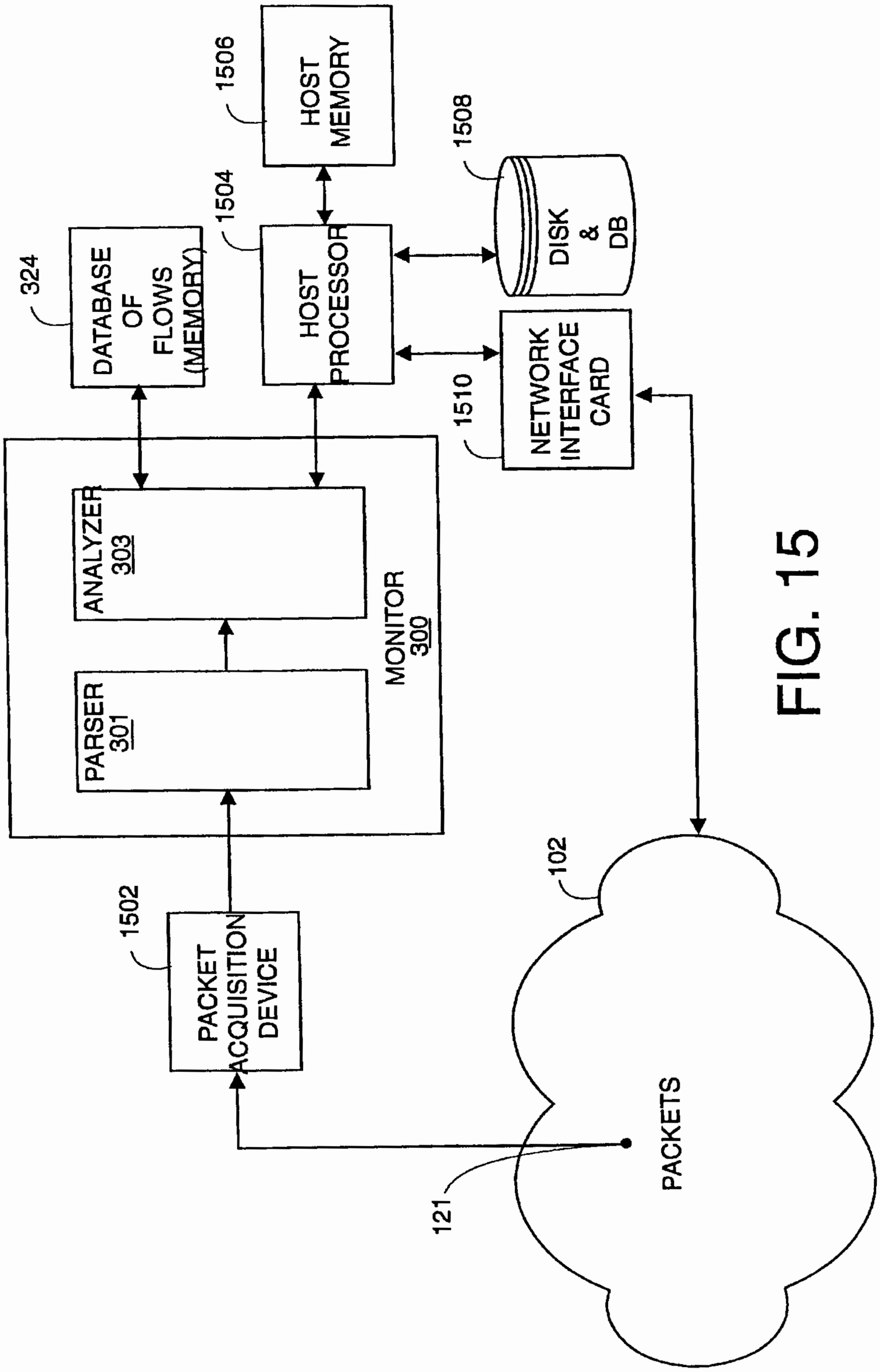

324
DATABASE OF FLOWS (MEMORY)
1504
HOST PROCESSOR
1506
HOST MEMORY
1508
DISK & DB
1510
NETWORK INTERFACE CARD
ANALYZER 303
PARSER 301
MONITOR 300
1502
PACKET ACQUISITION DEVICE
102
PACKETS
121

FIG. 15

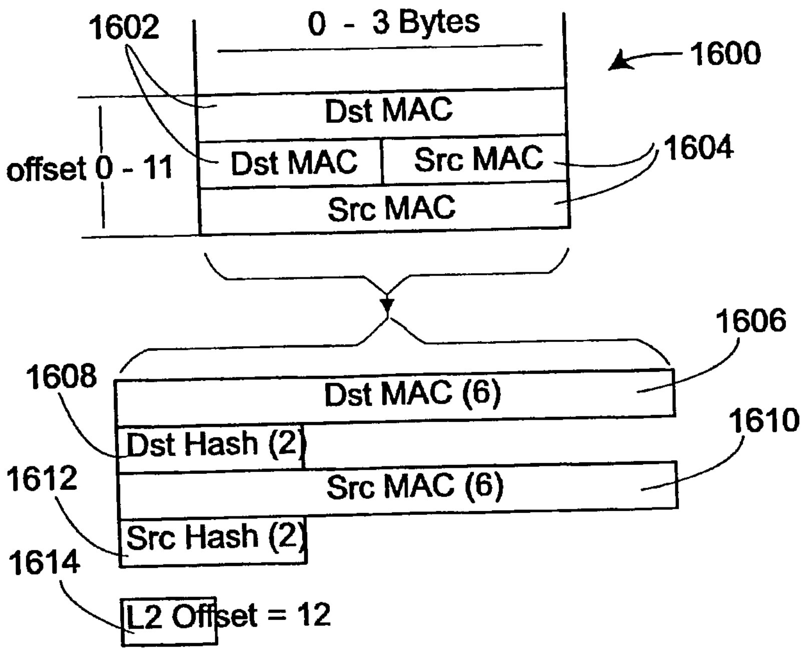

1602
0 - 3 Bytes
1600
Dst MAC
offset 0 - 11
Dst MAC
Src MAC
1604
Src MAC
1606
1608
Dst MAC (6)
Dst Hash (2)
1610
1612
Src MAC (6)
Src Hash (2)
1614
L2 Offset = 12

FIG. 16

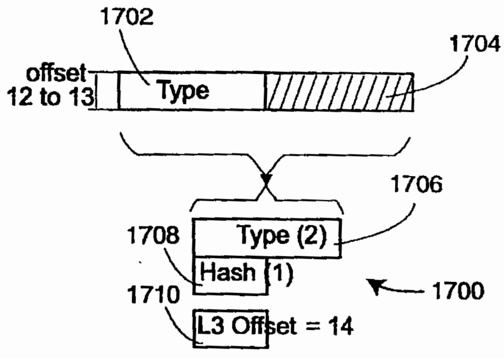


FIG. 17A

1712

IDP	= 0x0600*
IP	= 0x0800*
CHAOSNET	= 0x0804
ARP	= 0x0806
VIP	= 0x0BAD*
VLOOP	= 0x0BAE
VECHO	= 0x0BAF
NETBIOS-3COM	= 0x3C00 - 0x3C0D#
DEC-MOP	= 0x6001
DEC-RC	= 0x6002
DEC-DRP	= 0x6003*
DEC-LAT	= 0x6004
DEC-DIAG	= 0x6005
DEC-LAVC	= 0x6007
RARP	= 0x8035
ATALK	= 0x809B*
VLOOP	= 0x80C4
VECHO	= 0x80C5
SNA-TH	= 0x80D5*
ATALKARP	= 0x80F3
IPX	= 0x8137*
SNMP	= 0x814C#
IPv6	= 0x86DD*
LOOPBACK	= 0x9000
Apple	= 0x080007

* L3 Decoding
# L5 Decoding

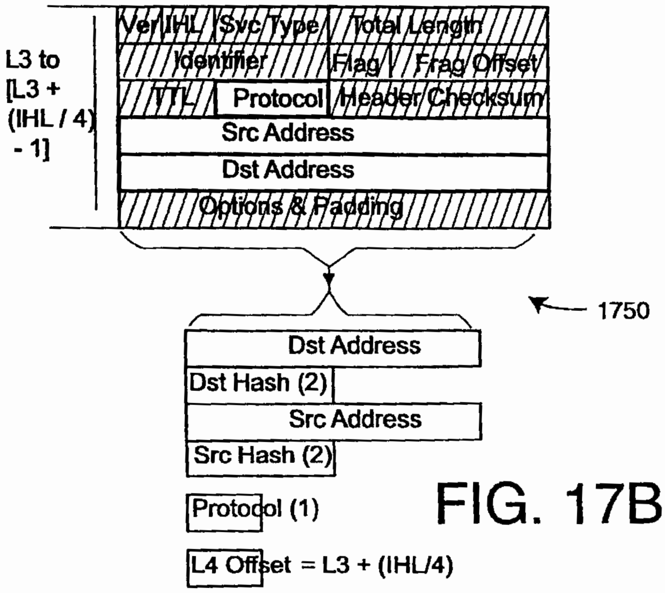


FIG. 17B

1752

ICMP	= 1
IGMP	= 2
GGP	= 3
TCP	= 6*
EGP	= 8
IGRP	= 9
PUP	= 12
CHAOS	= 16
UDP	= 17*
IDP	= 22#
ISO-TP4	= 29
DDP	= 37#
ISO-IP	= 80
VIP	= 83#
EIGRP	= 88
OSPF	= 89

* L4 Decoding
L3 Re-Decoding

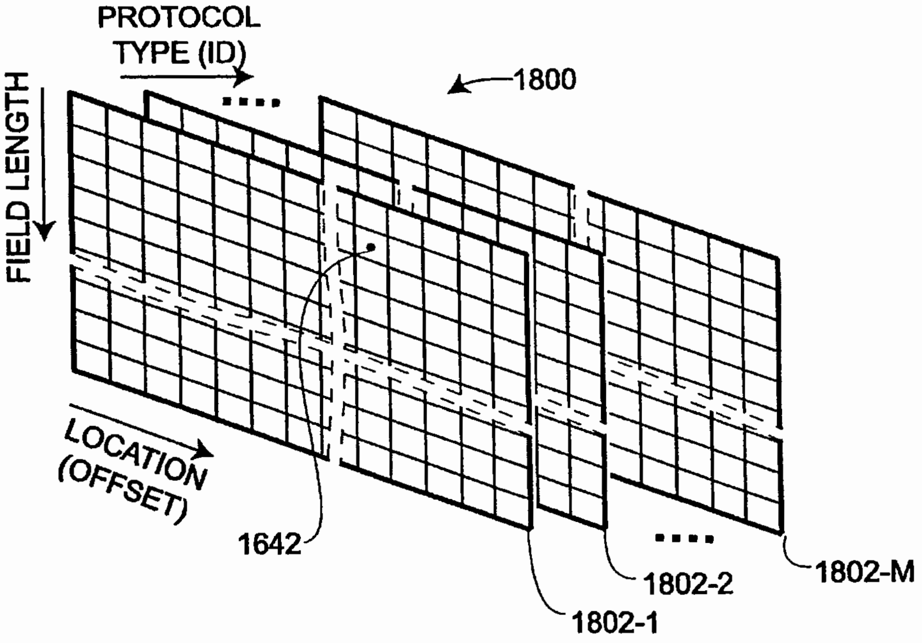


FIG. 18A

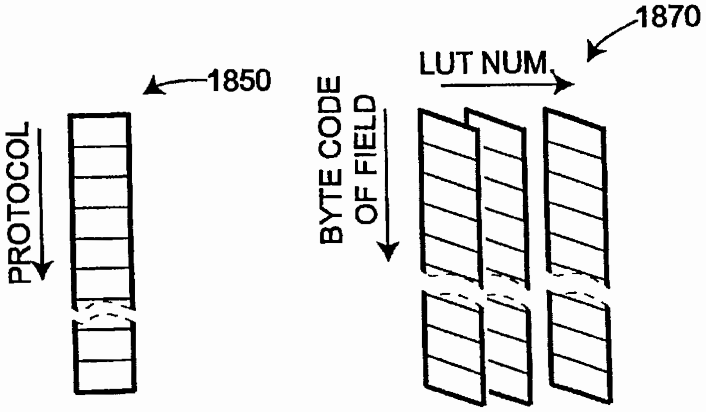


FIG. 18B

METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK

CROSS-REFERENCE TO RELATED APPLICATION

This application claims the benefit of U.S. Provisional Patent Application Ser. No.: 60/141,903 for METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK to inventors Dietz, et al., filed Jun. 30, 1999, the contents of which are incorporated herein by reference.

This application is related to the following U.S. patent applications, each filed concurrently with the present application, and each assigned to Apptitude, Inc., the assignee of the present invention:

U.S. patent application Ser. No. 09/609,179 for PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE, to inventors Koppenhaver, et al., filed Jun. 30, 2000, still pending, and incorporated herein by reference. U.S. patent application Ser. No. 09/608,126 for RE-USING INFORMATION FROM DATA TRANSACTIONS FOR MAINTAINING STATISTICS IN NETWORK MONITORING, to inventors Dietz, et al., filed Jun. 30, 2000, still pending, and incorporated herein by reference. U.S. patent application Ser. No. 09/608,266 for ASSOCIATIVE CACHE STRUCTURE FOR LOOKUPS AND UPDATES OF FLOW RECORDS IN A NETWORK MONITOR, to inventors Sarkissian, et al., filed Jun. 30, 2000, still penting, and incorporated herein by reference. U.S. patent application Ser. No. 09/608,267 for STATE PROCESSOR FOR PATTERN MATCHING IN A NETWORK MONITOR DEVICE, to inventors Sarkissian, et al., filed Jun. 30, 2000, still pending, and incorporated herein by reference.

FIELD OF INVENTION

The present invention relates to computer networks, specifically to the real-time elucidation of packets communicated within a data network, including classification according to protocol and application program.

BACKGROUND TO THE PRESENT INVENTION

There has long been a need for network activity monitors. This need has become especially acute, however, given the recent popularity of the Internet and other internets—an "internet" being any plurality of interconnected networks which forms a larger, single network. With the growth of networks used as a collection of clients obtaining services from one or more servers on the network, it is increasingly important to be able to monitor the use of those services and to rate them accordingly. Such objective information, for example, as which services (i.e., application programs) are being used, who is using them, how often they have been accessed, and for how long, is very useful in the maintenance and continued operation of these networks. It is especially important that selected users be able to access a network remotely in order to generate reports on network use in real time. Similarly, a need exists for a real-time network monitor that can provide alarms notifying selected users of problems that may occur with the network or site.

One prior art monitoring method uses log files. In this method, selected network activities may be analyzed retrospectively by reviewing log files, which are maintained by network servers and gateways. Log file monitors must access this data and analyze ("mine") its contents to determine statistics about the server or gateway. Several problems exist with this method, however. First, log file information does not provide a map of real-time usage; and secondly, log file mining does not supply complete information. This method relies on logs maintained by numerous network devices and servers, which requires that the information be subjected to refining and correlation. Also, sometimes information is simply not available to any gateway or server in order to make a log file entry.

One such case, for example, would be information concerning NetMeeting® (Microsoft Corporation, Redmond, Washington) sessions in which two computers connect directly on the network and the data is never seen by a server or a gateway.

Another disadvantage of creating log files is that the process requires data logging features of network elements to be enabled, placing a substantial load on the device, which results in a subsequent decline in network performance. Additionally, log files can grow rapidly, there is no standard means of storage for them, and they require a significant amount of maintenance.

Though Netflow® (Cisco Systems, Inc., San Jose, Calif.), RMON2, and other network monitors are available for the real-time monitoring of networks, they lack visibility into application content and are typically limited to providing network layer level information.

Pattern-matching parser techniques wherein a packet is parsed and pattern filters are applied are also known, but these too are limited in how deep into the protocol stack they can examine packets.

Some prior art packet monitors classify packets into connection flows. The term "connection flow" is commonly used to describe all the packets involved with a single connection. A conversational flow, on the other hand, is the sequence of packets that are exchanged in any direction as a result of an activity—for instance, the running of an application on a server as requested by a client. It is desirable to be able to identify and classify conversational flows rather than only connection flows. The reason for this is that some conversational flows involve more than one connection, and some even involve more than one exchange of packets between a client and server. This is particularly true when using client/server protocols such as RPC, DCOMP, and SAP, which enable a service to be set up or defined prior to any use of that service.

An example of such a case is the SAP (Service Advertising Protocol), a NetWare (Novell Systems, Provo, Utah) protocol used to identify the services and addresses of servers attached to a network. In the initial exchange, a client might send a SAP request to a server for print service. The server would then send a SAP reply that identifies a particular address—for example, SAP#5—as the print service on that server. Such responses might be used to update a table in a router, for instance, known as a Server Information Table. A client who has inadvertently seen this reply or who has access to the table (via the router that has the Service Information Table) would know that SAP#5 for this particular server is a print service. Therefore, in order to print data on the server, such a client would not need to make a request for a print service, but would simply send data to be printed specifying SAP#5. Like the previous exchange, the transmission of data to be printed also involves an exchange between a client and a server, but requires a second connection and is therefore independent of the initial exchange.

In order to eliminate the possibility of disjointed conversational exchanges, it is desirable for a network packet monitor to be able to "virtually concatenate"—that is, to link—the first exchange with the second. If the clients were the same, the two packet exchanges would then be correctly identified as being part of the same conversational flow.

Other protocols that may lead to disjointed flows, include RPC (Remote Procedure Call); DCOM (Distributed Component Object Model), formerly called Network OLE (Microsoft Corporation, Redmond, Wash.); and CORBA (Common Object Request Broker Architecture). RPC is a programming interface from Sun Microsystems (Palo Alto, Calif.) that allows one program to use the services of another program in a lo remote machine. DCOM, Microsoft's counterpart to CORBA, defines the remote procedure call that allows those objects—objects are self-contained software modules—to be run remotely over the network. And CORBA, a standard from the Object Management Group (OMG) for communicating between distributed objects, provides a way to execute programs (objects) written in different programming languages running on different platforms regardless of where they reside in a network.

What is needed, therefore, is a network monitor that makes it possible to continuously analyze all user sessions on a heavily trafficked network. Such a monitor should enable non-intrusive, remote detection, characterization, analysis, and capture of all information passing through any point on the network (i.e., of all packets and packet streams passing through any location in the network). Not only should all the packets be detected and analyzed, but for each of these packets the network monitor should determine the protocol (e.g., http, ftp, H.323, VPN, etc.), the application/use within the protocol (e.g., voice, video, data, real-time data, etc.), and an end user's pattern of use within each application or the application context (e.g., options selected, service delivered, duration, time of day, data requested, etc.). Also, the network monitor should not be reliant upon server resident information such as log files. Rather, it should allow a user such as a network administrator or an Internet service provider (ISP) the means to measure and analyze network activity objectively; to customize the type of data that is collected and analyzed; to undertake real time analysis; and to receive timely notification of network problems.

Considering the previous SAP example again, because one features of the invention is to correctly identify the second exchange as being associated with a print service on that server, such exchange would even be recognized if the clients were not the same. What distinguishes this invention from prior art network monitors is that it has the ability to recognize disjointed flows as belonging to the same conversational flow.

The data value in monitoring network communications has been recognized by many inventors. Chiu, et al., describe a method for collecting information at the session level in a computer network in U.S. Pat. No. 5,101,402, titled "APPARATUS AND METHOD FOR REAL-TIME MONITORING OF NETWORK SESSIONS AND A LOCAL AREA NETWORK" (the "402 patent"). The 402 patent specifies fixed locations for particular types of packets to extract information to identify session of a packet. For example, if a DECnet packet appears, the 402 patent looks at six specific fields (at 6 locations) in the packet in order to identify the session of the packet. If, on the other hand, an IP packet appears, a different set of six different locations is specified for an IP packet. With the proliferation of protocols, clearly the specifying of all the possible places to look to determine the session becomes more and more difficult. Likewise, adding a new protocol or application is difficult. In the present invention, the locations examined and the information extracted from any packet are adaptively determined from information in the packet for the particular type of packet. There is no fixed definition of what to look for and where to look in order to form an identifying signature. A monitor implementation of the present invention, for example, adapts to handle differently IEEE 802.3 packet from the older Ethernet Type 2 (or Version 2) DIX (Digital-Intel-Xerox) packet.

The 402 patent system is able to recognize up to the session layer. In the present invention, the number of levels examined varies for any particular protocol. Furthermore, the present invention is capable of examining up to whatever level is sufficient to uniquely identify to a required level, even all the way to the application level (in the OSI model).

Other prior art systems also are known. Phael describes a network activity monitor that processes only randomly selected packets in U.S. Pat. No. 5,315,580, titled "NETWORK MONITORING DEVICE AND SYSTEM." Nakamura teaches a network monitoring system in U.S. Pat. No. 4,891,639, titled "MONITORING SYSTEM OF NETWORK." Ross, et al., teach a method and apparatus for analyzing and monitoring network activity in U.S. Pat. No. 5,247,517, titled "METHOD AND APPARATUS FOR ANALYSIS NETWORKS," McCreery, et al., describe an Internet activity monitor that decodes packet data at the Internet protocol level layer in U.S. Pat. No. 5,787,253, titled "APPARATUS AND METHOD OF ANALYZING INTERNET ACTIVITY." The McCreery method decodes IP-packets. It goes through the decoding operations for each packet, and therefore uses the processing overhead for both recognized and unrecognized flows. In a monitor implementation of the present invention, a signature is built for every flow such that future packets of the flow are easily recognized. When a new packet in the flow arrives, the recognition process can commence from where it last left off, and a new signature built to recognize new packets of the flow.

SUMMARY

In its various embodiments the present invention provides a network monitor that can accomplish one or more of the following objects and advantages:

Recognize and classify all packets that are exchanges between a client and server into respective client/server applications.

Recognize and classify at all protocol layer levels conversational flows that pass in either direction at a point in a network.

Determine the connection and flow progress between clients and servers according to the individual packets exchanged over a network.

Be used to help tune the performance of a network according to the current mix of client/server applications requiring network resources.

Maintain statistics relevant to the mix of client/server applications using network resources.

Report on the occurrences of specific sequences of packets used by particular applications for client/server network conversational flows.

Other aspects of embodiments of the invention are:

Properly analyzing each of the packets exchanged between a client and a server and maintaining information relevant to the current state of each of these conversational flows. p1 Providing a flexible processing system that can be tailored or adapted as new applications enter the client/server market.

Maintaining statistics relevant to the conversational flows in a client/sever network as classified by an individual application.

Reporting a specific identifier, which may be used by other network-oriented devices to identify the series of packets with a specific application for a specific client/server network conversational flow.

In general, the embodiments of the present invention overcome the problems and disadvantages of the art.

As described herein, one embodiment analyzes each of the packets passing through any point in the network in either direction, in order to derive the actual application used to communicate between a client and a server. Note that there could be several simultaneous and overlapping applications executing over the network that are independent and asynchronous.

A monitor embodiment of the invention successfully classifies each of the individual packets as they are seen on the network. The contents of the packets are parsed and selected parts are assembled into a signature (also called a key) that may then be used identify further packets of the same conversational flow, for example to further analyze the flow and ultimately to recognize the application program. Thus the key is a function of the selected parts, and in the preferred embodiment, the function is a concatenation of the selected parts. The preferred embodiment forms and remembers the state of any conversational flow, which is determined by the relationship between individual packets and the entire conversational flow over the network. By remembering the state of a flow in this way, the embodiment determines the context of the conversational flow, including the application program it relates to and parameters such as the time, length of the conversational flow, data rate, etc.

The monitor is flexible to adapt to future applications developed for client/server networks. New protocols and protocol combinations may be incorporated by compiling files written in a high-level protocol description language.

The monitor embodiment of the present invention is preferably implemented in application-specific integrated circuits (ASIC) or field programmable gate arrays (FPGA). In one embodiment, the monitor comprises a parser subsystem that forms a signature from a packet. The monitor further comprises an analyzer subsystem that receives the signature from the parser subsystem.

A packet acquisition device such as a media access controller (MAC) or a segmentation and reassemble module is used to provide packets to the parser subsystem of the monitor.

In a hardware implementation, the parsing subsystem comprises two sub-parts, the pattern analysis and recognition engine (PRE), and an extraction engine (slicer). The PRE interprets each packet, and in particular, interprets individual fields in each packet according to a pattern database.

The different protocols that can exist in different layers may be thought of as nodes of one or more trees of linked nodes. The packet type is the root of a tree. Each protocol is either a parent node or a terminal node. A parent node links a protocol to other protocols (child protocols) that can be at higher layer levels. For example, An Ethernet packet (the root node) may be an Ethertype packet—also called an Ethernet Type/Version 2 and a DIX (DIGITAL-Intel-Xerox packet)—or an IEEE 802.3 packet. Continuing with the IEEE 802.3-type packet, one of the children nodes may be the IP protocol, and one of the children of the IP protocol may be the TCP protocol.

The pattern database includes a description of the different headers of packets and their contents, and how these relate to the different nodes in a tree. The PRE traverses the tree as far as it can. If a node does not include a link to a deeper level, pattern matching is declared complete. Note that protocols can be the children of several parents. If a unique node was generated for each of the possible parent/child trees, the pattern database might become excessively large. Instead, child nodes are shared among multiple parents, thus compacting the pattern database.

Finally the PRE can be used on its own when only protocol recognition is required.

For each protocol recognized, the slicer extracts important packet elements from the packet. These form a signature (i.e., key) for the packet. The slicer also preferably generates a hash for rapidly identifying a flow that may have this signature from a database of known flows.

The flow signature of the packet, the hash and at least some of the payload are passed to an analyzer subsystem. In a hardware embodiment, the analyzer subsystem includes a unified flow key buffer (UFKB) for receiving parts of packets from the parser subsystem and for storing signatures in process, a lookup/update engine (LUE) to lookup a database of flow records for previously encountered conversational flows to determine whether a signature is from an existing flow, a state processor (SP) for performing state processing, a flow insertion and deletion engine (FIDE) for inserting new flows into the database of flows, a memory for storing the database of flows, and a cache for speeding up access to the memory containing the flow database. The LUE, SP, and FIDE are all coupled to the UFKB, and to the cache.

The unified flow key buffer thus contains the flow signature of the packet, the hash and at least some of the payload for analysis in the analyzer subsystem. Many operations can be performed to further elucidate the identity of the application program content of the packet involved in the client/server conversational flow while a packet signature exists in the unified flow signature buffer. In the particular hardware embodiment of the analyzer subsystem several flows may be processed in parallel, and multiple flow signatures from all the packets being analyzed in parallel may be held in the one UFKB.

The first step in the packet analysis process of a packet from the parser subsystem is to lookup the instance in the current database of known packet flow signatures. A lookup/update engine (LUE) accomplishes this task using first the hash, and then the flow signature. The search is carried out in the cache and if there is no flow with a matching signature in the cache, the lookup engine attempts to retrieve the flow from the flow database in the memory. The flow-entry for previously encountered flows preferably includes state information, which is used in the state processor to execute any operations defined for the state, and to determine the next state. A typical state operation may be to search for one or more known reference strings in the payload of the packet stored in the UFKB.

Once the lookup processing by the LUE has been completed a flag stating whether it is found or is new is set within the unified flow signature buffer structure for this packet flow signature. For an existing flow, the flow-entry is updated by a calculator component of the LUE that adds values to counters in the flow-entry database used to store one or more statistical measures of the flow. The counters are used for determining network usage metrics on the flow.

After the packet flow signature has been looked up and contents of the current flow signature are in the database, a state processor can begin analyzing the packet payload to further elucidate the identity of the application program component of this packet. The exact operation of the state processor and functions performed by it will vary depending on the current packet sequence in the stream of a conversational flow. The state processor moves to the next logical operation stored from the previous packet seen with this same flow signature. If any processing is required on this packet, the state processor will execute instructions from a database of state instruction for this state until there are either no more left or the instruction signifies processing.

In the preferred embodiment, the state processor functions are programmable to provide for analyzing new application programs, and new sequences of packets and states that can arise from using such application.

If during the lookup process for this particular packet flow signature, the flow is required to be inserted into the active database, a flow insertion and deletion engine (FIDE) is initiated. The state processor also may create new flow signatures and thus may instruct the flow insertion and deletion engine to add a new flow to the database as a new item.

In the preferred hardware embodiment, each of the LUE, state processor, and FIDE operate independently from the other two engines.

BRIEF DESCRIPTION OF THE DRAWINGS

Although the present invention is better understood by referring to the detailed preferred embodiments, these should not be taken to limit the present invention to any specific embodiment because such embodiments are provided only for the purposes of explanation. The embodiments, in turn, are explained with the aid of the following figures.

FIG. **1** is a functional block diagram of a network embodiment of the present invention in which a monitor is connected to analyze packets passing at a connection point.

FIG. **2** is a diagram representing an example of some of the packets and their formats that might be exchanged in starting, as an illustrative example, a conversational flow between a client and server on a network being monitored and analyzed. A pair of flow signatures particular to this example and to embodiments of the present invention is also illustrated. This represents some of the possible flow signatures that can be generated and used in the process of analyzing packets and of recognizing the particular server applications that produce the discrete application packet exchanges.

FIG. **3** is a functional block diagram of a process embodiment of the present invention that can operate as the packet monitor shown in FIG. **1**. This process may be implemented in software or hardware.

FIG. **4** is a flowchart of a high-level protocol language compiling and optimization process, which in one embodiment may be used to generate data for monitoring packets according to versions of the present invention.

FIG. **5** is a flowchart of a packet parsing process used as part of the parser in an embodiment of the inventive packet monitor.

FIG. **6** is a flowchart of a packet element extraction process that is used as part of the parser in an embodiment of the inventive packet monitor.

FIG. **7** is a flowchart of a flow-signature building process that is used as part of the parser in the inventive packet monitor.

FIG. **8** is a flowchart of a monitor lookup and update process that is used as part of the analyzer in an embodiment of the inventive packet monitor.

FIG. **9** is a flowchart of an exemplary Sun Microsystems Remote Procedure Call application than may be recognized by the inventive packet monitor.

FIG. **10** is a functional block diagram of a hardware parser subsystem including the pattern recognizer and extractor that can form part of the parser module in an embodiment of the inventive packet monitor.

FIG. **11** is a functional block diagram of a hardware analyzer including a state processor that can form part of an embodiment of the inventive packet monitor.

FIG. **12** is a functional block diagram of a flow insertion and deletion engine process that can form part of the analyzer in an embodiment of the inventive packet monitor.

FIG. **13** is a flowchart of a state processing process that can form part of the analyzer in an embodiment of the inventive packet monitor.

FIG. **14** is a simple functional block diagram of a process embodiment of the present invention that can operate as the packet monitor shown in FIG. **1**. This process may be implemented in software.

FIG. **15** is a functional block diagram of how the packet monitor of FIG. **3** (and FIGS. **10** and **11**) may operate on a network with a processor such as a microprocessor.

FIG. **16** is an example of the top (MAC) layer of an Ethernet packet and some of the elements that may be extracted to form a signature according to one aspect of the invention.

FIG. **17A** is an example of the header of an Ethertype type of Ethernet packet of FIG. **16** and some of the elements that may be extracted to form a signature according to one aspect of the invention.

FIG. **17B** is an example of an IP packet, for example, of the Ethertype packet shown in FIGS. **16** and **17A**, and some of the elements that may be extracted to form a signature according to one aspect of the invention.

FIG. **18A** is a three dimensional structure that can be used to store elements of the pattern, parse and extraction database used by the parser subsystem in accordance to one embodiment of the invention.

FIG. **18B** is an alternate form of storing elements of the pattern, parse and extraction database used by the parser subsystem in accordance to another embodiment of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Note that this document includes hardware diagrams and descriptions that may include signal names. In most cases, the names are sufficiently descriptive, in other cases however the signal names are not needed to understand the operation and practice of the invention.

Operation in a Network

FIG. **1** represents a system embodiment of the present invention that is referred to herein by the general reference numeral **100**. The system **100** has a computer network **102** that communicates packets (e.g., IP datagrams) between various computers, for example between the clients **104–107** and servers **110** and **112**. The network is shown schematically as a cloud with several network nodes and links shown in the interior of the cloud. A monitor **108** examines the packets passing in either direction past its connection point **121** and, according to one aspect of the invention, can elucidate what application programs are associated with each packet. The monitor **108** is shown examining packets (i.e., datagrams) between the network interface **116** of the server **110** and the network. The monitor can also be placed at other points in the network, such as connection point **123** between the network **102** and the interface **118** of the client **104**, or some other location, as indicated schematically by connection point **125** somewhere in network **102**. Not shown is a network packet acquisition device at the location **123** on the network for converting the physical information on the network into packets for input into monitor **108**. Such packet acquisition devices are common.

Various protocols may be employed by the network to establish and maintain the required communication, e.g., TCP/IP, etc. Any network activity—for example an application program run by the client **104** (CLIENT 1) communicating with another running on the server **110** (SERVER 2)—will produce an exchange of a sequence of packets over network **102** that is characteristic of the respective programs and of the network protocols. Such characteristics may not be completely revealing at the individual packet level. It may require the analyzing of many packets by the monitor **108** to have enough information needed to recognize particular application programs. The packets may need to be parsed then analyzed in the context of various protocols, for example, the transport through the application session layer protocols for packets of a type conforming to the ISO layered network model.

Communication protocols are layered, which is also referred to as a protocol stack. The ISO (International Standardization Organization) has defined a general model that provides a framework for design of communication protocol layers. This model, shown in tables form below, serves as a basic reference for understanding the functionality of existing communication protocols.

ISO MODEL

Layer	Functionality	Example
7	Application	Telnet, NFS, Novell NCP, HTTP, H.323
6	Presentation	XDR
5	Session	RPC, NETBIOS, SNMP, etc.
4	Transport	TCP, Novel SPX, UDP, etc.
3	Network	IP, Novell IPX, VIP, AppleTalk, etc.
2	Data Link	Network Interface Card (Hardware Interface). MAC layer
1	Physical	Ethernet, Token Ring, Frame Relay, ATM, T1 (Hardware Connection)

Different communication protocols employ different levels of the ISO model or may use a layered model that is similar to but which does not exactly conform to the ISO model. A protocol in a certain layer may not be visible to protocols employed at other layers. For example, an application (Level 7) may not be able to identify the source computer for a communication attempt (Levels 2–3).

In some communication arts, the term "frame" generally refers to encapsulated data at OSI layer 2, including a destination address, control bits for flow control, the data or payload, and CRC (cyclic redundancy check) data for error checking. The term "packet" generally refers to encapsulated data at OSI layer 3. In the TCP/IP world, the term "datagram" is also used. In this specification, the term "packet" is intended to encompass packets, datagrams, frames, and cells. In general, a packet format or frame format refers to how data is encapsulated with various fields and headers for transmission across a network. For example, a data packet typically includes an address destination field, a length field, an error correcting code (ECC) field, or cyclic redundancy check (CRC) field, as well as headers and footers to identify the beginning and end of the packet. The terms "packet format" and "frame format," also referred to as "cell format,"0 are generally synonymous.

Monitor **108** looks at every packet passing the connection point **121** for analysis. However, not every packet carries the same information useful for recognizing all levels of the protocol. For example, in a conversational flow associated with a particular application, the application will cause the server to send a type-A packet, but so will another. If, though, the particular application program always follows a type-A packet with the sending of a type-B packet, and the other application program does not, then in order to recognize packets of that application's conversational flow, the monitor can be available to recognize packets that match the type-B packet to associate with the type-A packet. If such is recognized after a type-A packet, then the particular application program's conversational flow has started to reveal itself to the monitor **108**.

Further packets may need to be examined before the conversational flow can be identified as being associated with the application program. Typically, monitor **108** is simultaneously also in partial completion of identifying other packet exchanges that are parts of conversational flows associated with other applications. One aspect of monitor **108** is its ability to maintain the state of a flow. The state of a flow is an indication of all previous events in the flow that lead to recognition of the content of all the protocol levels, e.g., the ISO model protocol levels. Another aspect of the invention is forming a signature of extracted characteristic portions of the packet that can be used to rapidly identify packets belonging to the same flow.

In real-world uses of the monitor **108**, the number of packets on the network **102** passing by the monitor **108**'s connection point can exceed a million per second. Consequently, the monitor has very little time available to analyze and type each packet and identify and maintain the state of the flows passing through the connection point. The monitor **108** therefore masks out all the unimportant parts of each packet that will not contribute to its classification. However, the parts to mask-out will change with each packet depending on which flow it belongs to and depending on the state of the flow.

The recognition of the packet type, and ultimately of the associated application programs according to the packets that their executions produce, is a multi-step process within the monitor **108**. At a first level, for example, several application programs will all produce a first kind of packet. A first "signature" is produced from selected parts of a packet that will allow monitor **108** to identify efficiently any packets that belong to the same flow. In some cases, that packet type may be sufficiently unique to enable the monitor to identify the application that generated such a packet in the conversational flow. The signature can then be used to efficiently identify all future packets generated in traffic related to that application.

In other cases, that first packet only starts the process of analyzing the conversational flow, and more packets are necessary to identify the associated application program. In such a case, a subsequent packet of a second type—but that potentially belongs to the same conversational flow—is recognized by using the signature. At such a second level, then, only a few of those application programs will have conversational flows that can produce such a second packet type. At this level in the process of classification, all application programs that are not in the set of those that lead to such a sequence of packet types may be excluded in the process of classifying the conversational flow that includes these two packets. Based on the known patterns for the protocol and for the possible applications, a signature is produced that allows recognition of any future packets that may follow in the conversational flow.

It may be that the application is now recognized, or recognition may need to proceed to a third level of analysis using the second level signature. For each packet, therefore, the monitor parses the packet and generates a signature to determine if this signature identified a previously encountered flow, or shall be used to recognize future packets belonging to the same conversational flow. In real time, the packet is further analyzed in the context of the sequence of previously encountered packets (the state), and of the possible future sequences such a past sequence may generate in conversational flows associated with different applications. A new signature for recognizing future packets may also be generated. This process of analysis continues until the applications are identified. The last generated signature may then be used to efficiently recognize future packets associated with the same conversational flow. Such an arrangement makes it possible for the monitor 108 to cope with millions of packets per second that must be inspected.

Another aspect of the invention is adding Eavesdropping. In alternative embodiments of the present invention capable of eavesdropping, once the monitor 108 has recognized the executing application programs passing through some point in the network 102 (for example, because of execution of the applications by the client 105 or server 110), the monitor sends a message to some general purpose processor on the network that can input the same packets from the same location on the network, and the processor then loads its own executable copy of the application program and uses it to read the content being exchanged over the network. In other words, once the monitor 108 has accomplished recognition of the application program, eavesdropping can commence.

The Network Monitor

FIG. 3 shows a network packet monitor 300, in an embodiment of the present invention that can be implemented with computer hardware and/or software. The system 300 is similar to monitor 108 in FIG. 1. A packet 302 is examined, e.g., from a packet acquisition device at the location 121 in network 102 (FIG. 1), and the packet evaluated, for example in an attempt to determine its characteristics, e.g., all the protocol information in a multilevel model, including what server application produced the packet.

The packet acquisition device is a common interface that converts the physical signals and then decodes them into bits, and into packets, in accordance with the particular network (Ethernet, frame relay, ATM, etc.). The acquisition device indicates to the monitor 108 the type of network of the acquired packet or packets.

Aspects shown here include: (1) the initialization of the monitor to generate what operations need to occur on packets of different types—accomplished by compiler and optimizer 310, (2) the processing—parsing and extraction of selected portions—of packets to generate an identifying signature—accomplished by parser subsystem 301, and (3) the analysis of the packets—accomplished by analyzer 303.

The purpose of compiler and optimizer 310 is to provide protocol specific information to parser subsystem 301 and to analyzer subsystem 303. The initialization occurs prior to operation of the monitor, and only needs to re-occur when new protocols are to be added.

A flow is a stream of packets being exchanged between any two addresses in the network. For each protocol there are known to be several fields, such as the destination (recipient), the source (the sender), and so forth, and these and other fields are used in monitor 300 to identify the flow. There are other fields not important for identifying the flow, such as checksums, and those parts are not used for identification.

Parser subsystem 301 examines the packets using pattern recognition process 304 that parses the packet and determines the protocol types and associated headers for each protocol layer that exists in the packet 302. An extraction process 306 in parser subsystem 301 extracts characteristic portions (signature information) from the packet 302. Both the pattern information for parsing and the related extraction operations, e.g., extraction masks, are supplied from a parsing-pattern-structures and extraction-operations database (parsing/extractions database) 308 filled by the compiler and optimizer 310.

The protocol description language (PDL) files 336 describes both patterns and states of all protocols that an occur at any layer, including how to interpret header information, how to determine from the packet header information the protocols at the next layer, and what information to extract for the purpose of identifying a flow, and ultimately, applications and services. The layer selections database 338 describes the particular layering handled by the monitor. That is, what protocols run on top of what protocols at any layer level. Thus 336 and 338 combined describe how one would decode, analyze, and understand the information in packets, and, furthermore, how the information is layered. This information is input into compiler and optimizer 310.

When compiler and optimizer 310 executes, it generates two sets of internal data structures. The first is the set of parsing/extraction operations 308. The pattern structures include parsing information and describe what will be recognized in the headers of packets; the extraction operations are what elements of a packet are to be extracted from the packets based on the patterns that get matched. Thus, database 308 of parsing/extraction operations includes information describing how to determine a set of one or more protocol dependent extraction operations from data in the packet that indicate a protocol used in the packet.

The other internal data structure that is built by compiler 310 is the set of state patterns and processes 326. These are the different states and state transitions that occur in different conversational flows, and the state operations that need to be performed (e.g., patterns that need to be examined and new signatures that need to be built) during any state of a conversational flow to further the task of analyzing the conversational flow.

Thus, compiling the PDL files and layer selections provides monitor 300 with the information it needs to begin processing packets. In an alternate embodiment, the contents of one or more of databases 308 and 326 may be manually or otherwise generated. Note that in some embodiments the layering selections information is inherent rather than explicitly described. For example, since a PDL file for a protocol includes the child protocols, the parent protocols also may be determined.

In the preferred embodiment, the packet 302 from the acquisition device is input into a packet buffer. The pattern recognition process 304 is carried out by a pattern analysis and recognition (PAR) engine that analyzes and recognizes patterns in the packets. In particular, the PAR locates the next protocol field in the header and determines the length of the header, and may perform certain other tasks for certain types of protocol headers. An example of this is type and length comparison to distinguish an IEEE 802.3 (Ethernet) packet from the older type 2 (or Version 2) Ethernet packet, also called a DIGITAL-Intel-Xerox (DIX) packet. The PAR also uses the pattern structures and extraction operations database **308** to identify the next protocol and parameters associated with that protocol that enables analysis of the next protocol layer. Once a pattern or a set of patterns has been identified, it/they will be associated with a set of none or more extraction operations. These extraction operations (in the form of commands and associated parameters) are passed to the extraction process **306** implemented by an extracting and information identifying (EII) engine that extracts selected parts of the packet, including identifying information from the packet as required for recognizing this packet as part of a flow. The extracted information is put in sequence and then processed in block **312** to build a unique flow signature (also called a "key") for this flow. A flow signature depends on the protocols used in the packet. For some protocols, the extracted components may include source and destination addresses. For example, Ethernet frames have end-point addresses that are useful in building a better flow signature. Thus, the signature typically includes the client and server address pairs. The signature is used to recognize further packets that are or may be part of this flow.

In the preferred embodiment, the building of the flow key includes generating a hash of the signature using a hash function. The purpose if using such a hash is conventional—to spread flow-entries identified by the signature across a database for efficient searching. The hash generated is preferably based on a hashing algorithm and such hash generation is known to those in the art.

In one embodiment, the parser passes data from the packet—a parser record—that includes the signature (i.e., selected portions of the packet), the hash, and the packet itself to allow for any state processing that requires further data from the packet. An improved embodiment of the parser subsystem might generate a parser record that has some predefined structure and that includes the signature, the hash, some flags related to some of the fields in the parser record, and parts of the packet's payload that the parser subsystem has determined might be required for further processing, e.g., for state processing.

Note that alternate embodiments may use some function other than concatenation of the selected portions of the packet to make the identifying signature. For example, some "digest function" of the concatenated selected portions may be used.

The parser record is passed onto lookup process **314** which looks in an internal data store of records of known flows that the system has already encountered, and decides (in **316**) whether or not this particular packet belongs to a known flow as indicated by the presence of a flow-entry matching this flow in a database of known flows **324**. A record in database **324** is associated with each encountered flow.

The parser record enters a buffer called the unified flow key buffer (UFKB). The UFKB stores the data on flows in a data structure that is similar to the parser record, but that includes a field that can be modified. In particular, one or the UFKB record fields stores the packet sequence number, and another is filled with state information in the form of a program counter for a state processor that implements state processing **328**.

The determination (**316**) of whether a record with the same signature already exists is carried out by a lookup engine (LUE) that obtains new UFKB records and uses the hash in the UFKB record to lookup if there is a matching known flow. In the particular embodiment, the database of known flows **324** is in an external memory. A cache is associated with the database **324**. A lookup by the LUE for a known record is carried out by accessing the cache using the hash, and if the entry is not already present in the cache, the entry is looked up (again using the hash) in the external memory.

The flow-entry database **324** stores flow-entries that include the unique flow-signature, state information, and extracted information from the packet for updating flows, and one or more statistical about the flow. Each entry completely describes a flow. Database **324** is organized into bins that contain a number, denoted N, of flow-entries (also called flow-entries, each a bucket), with N being 4 in the preferred embodiment. Buckets (i.e., flow-entries) are accessed via the hash of the packet from the parser subsystem **301** (i.e., the hash in the UFKB record). The hash spreads the flows across the database to allow for fast lookups of entries, allowing shallower buckets. The designer selects the bucket depth N based on the amount of memory attached to the monitor, and the number of bits of the hash data value used. For example, in one embodiment, each flow-entry is 128 bytes long, so for 128K flow-entries, 16 Mbytes are required. Using a is 16-bit hash gives two flow-entries per bucket. Empirically, this has been shown to be more than adequate for the vast majority of cases. Note that another embodiment uses flow-entries that are 256 bytes long.

Herein, whenever an access to database **324** is described, it is to be understood that the access is via the cache, unless otherwise stated or clear from the context.

If there is no flow-entry found matching the signature, i.e., the signature is for a new flow, then a protocol and state identification process **318** further determines the state and protocol. That is, process **318** determines the protocols and where in the state sequence for a flow for this protocol's this packet belongs. Identification process **318** uses the extracted information and makes reference to the database **326** of state patterns and processes. Process **318** is then followed by any state operations that need to be executed on this packet by a state processor **328**.

If the packet is found to have a matching flow-entry in the database **324** (e.g., in the cache), then a process **320** determines, from the looked-up flow-entry, if more classification by state processing of the flow signature is necessary. If not, a process **322** updates the flow-entry in the flow-entry database **324** (e.g., via the cache). Updating includes updating one or more statistical measures stored in the flow-entry. In our embodiment, the statistical measures are stored in counters in the flow-entry.

If state processing is required, state process **328** is commenced. State processor **328** carries out any state operations specified for the state of the flow and updates the state to the next state according to a set of state instructions obtained form the state pattern and processes database **326**.

The state processor **328** analyzes both new and existing flows in order to analyze all levels of the protocol stack, ultimately classifying the flows by application (level 7 in the ISO model). It does this by proceeding from state-to-state based on predefined state transition rules and state operations as specified in state processor instruction database 326. A state transition rule is a rule typically containing a test followed by the next-state to proceed to if the test result is true. An operation is an operation to be performed while the state processor is in a particular state—for example, in order to evaluate a quantity needed to apply the state transition rule. The state processor goes through each rule and each state process until the test is true, or there are no more tests to perform.

In general, the set of state operations may be none or more operations on a packet, and carrying out the operation or operations may leave one in a state that causes exiting the system prior to completing the identification, but possibly knowing more about what state and state processes are needed to execute next, i.e., when a next packet of this flow is encountered. As an example, a state process (set of state operations) at a particular state may build a new signature for future recognition packets of the next state.

By maintaining the state of the flows and knowing that new flows may be set up using the information from previously encountered flows, the network traffic monitor **300** provides for (a) single-packet protocol recognition of flows, and (b) multiple-packet protocol recognition of flows. Monitor **300** can even recognize the application program from one or more disjointed sub-flows that occur in server announcement type flows. What may seem to prior art monitors to be some unassociated flow, may be recognized by the inventive monitor using the flow signature to be a sub-flow associated with a previously encountered sub-flow.

Thus, state processor **328** applies the first state operation to the packet for this particular flow-entry. A process **330** decides if more operations need to be performed for this state. If so, the analyzer continues looping between block **330** and **328** applying additional state operations to this particular packet until all those operations are completed—that is, there are no more operations for this packet in this state. A process **332** decides if there are further states to be analyzed for this type of flow according to the state of the flow and the protocol, in order to fully characterize the flow. If not, the conversational flow has now been fully characterized and a process **334** finalizes the classification of the conversational flow for the flow.

In the particular embodiment, the state processor **328** starts the state processing by using the last protocol recognized by the parser as an offset into a jump table (ump vector). The jump table finds the state processor instructions to use for that protocol in the state patterns and processes database **326**. Most instructions test something in the unified flow key buffer, or the flow-entry in the database of known flows **324**, if the entry exists. The state processor may have to test bits, do comparisons, add, or subtract to perform the test. For example, a common operation carried out by the state processor is searching for one or more patterns in the payload part of the UFKB.

Thus, in **332** in the classification, the analyzer decides whether the flow is at an end state. If not at an end state, the flow-entry is updated (or created if a new flow) for this flow-entry in process **322**.

Furthermore, if the flow is known and if in **332** it is determined that there are further states to be processed using later packets, the flow-entry is updated in process **322**.

The flow-entry also is updated after classification finalization so that any further packets belonging to this flow will be readily identified from their signature as belonging to this fully analyzed conversational flow.

After updating, database **324** therefore includes the set of all the conversational flows that have occurred.

Thus, the embodiment of present invention shown in FIG. **3** automatically maintains flow-entries, which in one aspect includes storing states. The monitor of FIG. **3** also generates characteristic parts of packets—the signatures—that can be used to recognize flows. The flow-entries may be identified and accessed by their signatures. Once a packet is identified to be from a known flow, the state of the flow is known and this knowledge enables state transition analysis to be performed in real time for each different protocol and application. In a complex analysis, state transitions are traversed as more and more packets are examined. Future packets that are part of the same conversational flow have their state analysis continued from a previously achieved state. When enough packets related to an application of interest have been processed, a final recognition state is ultimately reached, i.e., a set of states has been traversed by state analysis to completely characterize the conversational flow. The signature for that final state enables each new incoming packet of the same conversational flow to be individually recognized in real time.

In this manner, one of the great advantages of the present invention is realized. Once a particular set of state transitions has been traversed for the first time and ends in a final state, a short-cut recognition pattern—a signature—can be generated that will key on every new incoming packet that relates to the conversational flow. Checking a signature involves a simple operation, allowing high packet rates to be successfully monitored on the network.

In improved embodiments, several state analyzers are run in parallel so that a large number of protocols and applications may be checked for. Every known protocol and application will have at least one unique set of state transitions, and can therefore be uniquely identified by watching such transitions.

When each new conversational flow starts, signatures that recognize the flow are automatically generated on-the-fly, and as further packets in the conversational flow are encountered, signatures are updated and the states of the set of state transitions for any potential application are further traversed according to the state transition rules for the flow. The new states for the flow—those associated with a set of state transitions for one or more potential applications—are added to the records of previously encountered states for easy recognition and retrieval when a new packet in the flow is encountered.

Detailed Operation

FIG. **4** diagrams an initialization system **400** that includes the compilation process. That is, part of the initialization generates the pattern structures and extraction operations database **308** and the state instruction database **328**. Such initialization can occur off-line or from a central location.

The different protocols that can exist in different layers may be thought of as nodes of one or more trees of linked nodes. The packet type is the root of a tree (called level **0**). Each protocol is either a parent node or a terminal node. A parent node links a protocol to other protocols (child protocols) that can be at higher layer levels. Thus a protocol may have zero or more children. Ethernet packets, for example, have several variants, each having a basic format that remains substantially the same. An Ethernet packet (the root or level **0** node) may be an Ethertype packet—also called an Ethernet Type/Version 2 and a DIX (DIGITAL-Intel-Xerox packet)—or an IEEE 803.2 packet. Continuing with the IEEE 802.3 packet, one of the children nodes may be the IP protocol, and one of the children of the IP protocol may be the TCP protocol.

FIG. 16 shows the header 1600 (base level 1) of a complete Ethernet frame (i.e., packet) of information and includes information on the destination media access control address (Dst MAC 1602) and the source media access control address (Src MAC 1604). Also shown in FIG. 16 is some (but not all) of the information specified in the PDL files for extraction the signature.

FIG. 17A now shows the header information for the next level (level-2) for an Ethertype packet 1700. For an Ethertype packet 1700, the relevant information from the packet that indicates the next layer level is a two-byte type field 1702 containing the child recognition pattern for the next level. The remaining information 1704 is shown hatched because it not relevant for this level. The list 1712 shows the possible children for an Ethertype packet as indicated by what child recognition pattern is found offset 12. FIG. 17B shows the structure of the header of one of the possible next levels, that of the IP protocol. The possible children of the IP protocol are shown in table 1752.

The pattern, parse, and extraction database (pattern recognition database, or PRD) 308 generated by compilation process 310, in one embodiment, is in the form of a three dimensional structure that provides for rapidly searching packet headers for the next protocol. FIG. 18A shows such a 3-D representation 1800 (which may be considered as an indexed set of 2-D representations). A compressed form of the 3-D structure is preferred.

An alternate embodiment of the data structure used in database 308 is illustrated in FIG. 18B. Thus, like the 3-D structure of FIG. 18A, the data structure permits rapid searches to be performed by the pattern recognition process 304 by indexing locations in a memory rather than performing address link computations. In this alternate embodiment, the PRD 308 includes two parts, a single protocol table 1850 (PT) which has an entry for each protocol known for the monitor, and a series of Look Up Tables 1870 (LUT's) that are used to identify known protocols and their children. The protocol table includes the parameters needed by the pattern analysis and recognition process 304 (implemented by PRE 1006) to evaluate the header information in the packet that is associated with that protocol, and parameters needed by extraction process 306 (implemented by slicer 1007) to process the packet header. When there are children, the PT describes which bytes in the header to evaluate to determine the child protocol. In particular, each PT entry contains the header length, an offset to the child, a slicer command, and some flags.

The pattern matching is carried out by finding particular "child recognition codes" in the header fields, and using these codes to index one or more of the LUT's. Each LUT entry has a node code that can have one of four values, indicating the protocol that has been recognized, a code to indicate that the protocol has been partially recognized (more LUT lookups are needed), a code to indicate that this is a terminal node, and a null node to indicate a null entry. The next LUT to lookup is also returned from a LUT lookup.

Compilation process is described in FIG. 4. The source-code information in the form of protocol description files is shown as 402. In the particular embodiment, the high level decoding descriptions includes a set of protocol description files 336, one for each protocol, and a set of packet layer selections 338, which describes the particular layering (sets of trees of protocols) that the monitor is to be able to handle.

A compiler 403 compiles the descriptions. The set of packet parse-and-extract operations 406 is generated (404), and a set of packet state instructions and operations 407 is generated (405) in the form of instructions for the state processor that implements state processing process 328. Data files for each type of application and protocol to be recognized by the analyzer are downloaded from the pattern, parse, and extraction database 406 into the memory systems of the parser and extraction engines. (See the parsing process 500 description and FIG. 5; the extraction process 600 description and FIG. 6; and the parsing subsystem hardware description and FIG. 10). Data files for each type of application and protocol to be recognized by the analyzer are also downloaded from the state-processor instruction database 407 into the state processor. (see the state processor 1108 description and FIG. 11.).

Note that generating the packet parse and extraction operations builds and links the three dimensional structure (one embodiment) or the or all the lookup tables for the PRD.

Because of the large number of possible protocol trees and subtrees, the compiler process 400 includes optimization that compares the trees and subtrees to see which children share common parents. When implemented in the form of the LUT's, this process can generate a single LUT from a plurality of LUT's. The optimization process further includes a compaction process that reduces the space needed to store the data of the PRD.

As an example of compaction, consider the 3-D structure of FIG. 18A that can be thought of as a set of 2-D structures each representing a protocol. To enable saving space by using only one array per protocol which may have several parents, in one embodiment, the pattern analysis subprocess keeps a "current header" pointer. Each location (offset) index for each protocol 2-D array in the 3-D structure is a relative location starting with the start of header for the particular protocol. Furthermore, each of the two-dimensional arrays is sparse. The next step of the optimization, is checking all the 2-D arrays against all the other 2-D arrays to find out which ones can share memory. Many of these 2-D arrays are often sparsely populated in that they each have only a small number of valid entries. So, a process of "folding" is next used to combine two or more 2-D arrays together into one physical 2-D array without losing the identity of any of the original 2-D arrays (i.e., all the 2-D arrays continue to exist logically). Folding can occur between any 2-D arrays irrespective of their location in the tree as long as certain conditions are met. Multiple arrays may be combined into a single array as long as the individual entries do not conflict with each other. A fold number is then used to associate each element with its original array. A similar folding process is used for the set of LUTs 1850 in the alternate embodiment of FIG. 18B.

In 410, the analyzer has been initialized and is ready to perform recognition.

FIG. 5 shows a flowchart of how actual parser subsystem 301 functions. Starting at 501, the packet 302 is input to the packet buffer in step 502. Step 503 loads the next (initially the first) packet component from the packet 302. The packet components are extracted from each packet 302 one element at a time. A check is made (504) to determine if the load-packet-component operation 503 succeeded, indicating that there was more in the packet to process. If not, indicating all components have been loaded, the parser subsystem 301 builds the packet signature (512)—the next stage (FIG. 6).

If a component is successfully loaded in 503, the node and processes are fetched (505) from the pattern, parse and extraction database 308 to provide a set of patterns and processes for that node to apply to the loaded packet component. The parser subsystem 301 checks (506) to determine if the fetch pattern-node operation 505 completed successfully, indicating there was a pattern node that loaded in 505. If not, step 511 moves to the next packet component. If yes, then the node and pattern matching process are applied in 507 to the component extracted in 503. A pattern match obtained in 507 (as indicated by test 508) means the parser subsystem 301 has found a node in the parsing elements; the parser subsystem 301 proceeds to step 509 to extract the elements.

If applying the node process to the component does not produce a match (test 508), the parser subsystem 301 moves (510) to the next pattern node from the pattern database 308 and to step 505 to fetch the next node and process. Thus, there is an "applying patterns" loop between 508 and 505. Once the parser subsystem 301 completes all the patterns and has either matched or not, the parser subsystem 301 moves to the next packet component (511).

Once all the packet components have been the loaded and processed from the input packet 302, then the load packet will fail (indicated by test 504), and the parser subsystem 301 moves to build a packet signature which is described in FIG. 6

FIG. 6 is a flow chart for extracting the information from which to build the packet signature. The flow starts at 601, which is the exit point 513 of FIG. 5. At this point parser subsystem 301 has a completed packet component and a pattern node available in a buffer (602). Step 603 loads the packet component available from the pattern analysis process of FIG. 5. If the load completed (test 604), indicating that there was indeed another packet component, the parser subsystem 301 fetches in 605 the extraction and process elements received from the pattern node component in 602. If the fetch was successful (test 606), indicating that there are extraction elements to apply, the parser subsystem 301 in step 607 applies that extraction process to the packet component based on an extraction instruction received from that pattern node. This removes and saves an element from the packet component.

In step 608, the parser subsystem 301 checks if there is more to extract from this component, and if not, the parser subsystem 301 moves back to 603 to load the next packet component at hand and repeats the process. If the answer is yes, then the parser subsystem 301 moves to the next packet component ratchet. That new packet component is then loaded in step 603. As the parser subsystem 301 moved through the loop between 608 and 603, extra extraction processes are applied either to the same packet component if there is more to extract, or to a different packet component if there is no more to extract.

The extraction process thus builds the signature, extracting more and more components according to the information in the patterns and extraction database 308 for the particular packet. Once loading the next packet component operation 603 fails (test 604), all the components have been extracted. The built signature is loaded into the signature buffer (610) and the parser subsystem 301 proceeds to FIG. 7 to complete the signature generation process.

Referring now to FIG. 7, the process continues at 701. The signature buffer and the pattern node elements are available (702). The parser subsystem 301 loads the next pattern node element. If the load was successful (test 704) indicating there are more nodes, the parser subsystem 301 in 705 hashes the signature buffer element based on the hash elements that are found in the pattern node that is in the element database. In 706 the resulting signature and the hash are packed. In 707 the parser subsystem 301 moves on to the next packet component which is loaded in 703.

The 703 to 707 loop continues until there are no more patterns of elements left (test 704). Once all the patterns of elements have been hashed, processes 304, 306 and 312 of parser subsystem 301 are complete. Parser subsystem 301 has generated the signature used by the analyzer subsystem 303.

A parser record is loaded into the analyzer, in particular, into the UFKB in the form of a UFKB record which is similar to a parser record, but with one or more different fields.

FIG. 8 is a flow diagram describing the operation of the lookup/update engine (LUE) that implements lookup operation 314. The process starts at 801 from FIG. 7 with the parser record that includes a signature, the hash and at least parts of the payload. In 802 those elements are shown in the form of a UFKB-entry in the buffer. The LUE, the lookup engine 314 computes a "record bin number" from the hash for a flow-entry. A bin herein may have one or more "buckets" each containing a flow-entry. The preferred embodiment has four buckets per bin.

Since preferred hardware embodiment includes the cache, all data accesses to records in the flowchart of FIG. 8 are stated as being to or from the cache.

Thus, in 804, the system looks up the cache for a bucket from that bin using the hash. If the cache successfully returns with a bucket from the bin number, indicating there are more buckets in the bin, the lookup/update engine compares (807) the current signature (the UFKB-entry's signature) from that in the bucket (i.e., the flow-entry signature). If the signatures match (test 808), that record (in the cache) is marked in step 810 as "in process" and a timestamp added. Step 811 indicates to the UFKB that the UFKB-entry in 802 has a status of "found." The "found" indication allows the state processing 328 to begin processing this UFKB element. The preferred hardware embodiment includes one or more state processors, and these can operate in parallel with the lookup/update engine.

In the preferred embodiment, a set of statistical operations is performed by a calculator for every packet analyzed. The statistical operations may include one or more of counting the packets associated with the flow; determining statistics related to the size of packets of the flow; compiling statistics on differences between packets in each direction, for example using times tamps; and determining statistical relationships of timestamps of packets in the same direction. The statistical measures are kept in the flow-entries. Other statistical measures also may be compiled. These statistics may be used singly or in combination by a statistical processor component to analyze many different aspects of the flow. This may include determining network usage metrics from the statistical measures, for example to ascertain the network's ability to transfer information for this application. Such analysis provides for measuring the quality of service of a conversation, measuring how well an application is performing in the network, measuring network resources consumed by an application, and so forth.

To provide for such analyses, the lookup/update engine updates one or more counters that are part of the flow-entry (in the cache) in step 812. The process exits at 813. In our embodiment, the counters include the total packets of the flow, the time, and a differential time from the last timestamp to the present timestamp.

It may be that the bucket of the bin did not lead to a signature match (test 808). In such a case, the analyzer in

809 moves to the next bucket for this bin. Step 804 again looks up the cache for another bucket from that bin. The lookup/update engine thus continues lookup up buckets of the bin until there is either a match in 808 or operation 804 is not successful (test 805), indicating that there are no more buckets in the bin and no match was found.

If no match was found, the packet belongs to a new (not previously encountered) flow. In 806 the system indicates that the record in the unified flow key buffer for this packet is new, and in 812, any statistical updating operations are performed for this packet by updating the flow-entry in the cache. The update operation exits at 813. A flow insertion/deletion engine (FIDE) creates a new record for this flow (again via the cache).

Thus, the update/lookup engine ends with a UFKB-entry for the packet with a "new" status or a "found" status.

Note that the above system uses a hash to which more than one flow-entry can match. A longer hash may be used that corresponds to a single flow-entry. In such an embodiment, the flow chart of FIG. 8 is simplified as would be clear to those in the art.

The Hardware System

Each of the individual hardware elements through which the data flows in the system are now described with reference to FIGS. 10 and 11. Note that while we are describing a particular hardware implementation of the invention embodiment of FIG. 3, it would be clear to one skilled in the art that the flow of FIG. 3 may alternatively be implemented in software running on one or more general-purpose processors, or only partly implemented in hardware. An implementation of the invention that can operate in software is shown in FIG. 14. The hardware embodiment (FIGS. 10 and 11) can operate at over a million packets per second, while the software system of FIG. 14 may be suitable for slower networks. To one skilled in the art it would be clear that more and more of the system may be implemented in software as processors become faster.

FIG. 10 is a description of the parsing subsystem (301, shown here as subsystem 1000) as implemented in hardware. Memory 1001 is the pattern recognition database memory, in which the patterns that are going to be analyzed are stored. Memory 1002 is the extraction-operation database memory, in which the extraction instructions are stored. Both 1001 and 1002 correspond to internal data structure 308 of FIG. 3. Typically, the system is initialized from a microprocessor (not shown) at which time these memories are loaded through a host interface multiplexor and control register 1005 via the internal buses 1003 and 1004. Note that the contents of 1001 and 1002 are preferably obtained by compiling process 310 of FIG. 3.

A packet enters the parsing system via 1012 into a parser input buffer memory 1008 using control signals 1021 and 1023, which control an input buffer interface controller 1022. The buffer 1008 and interface control 1022 connect to a packet acquisition device (not shown). The buffer acquisition device generates a packet start signal 1021 and the interface control 1022 generates a next packet (i.e., ready to receive data) signal 1023 to control the data flow into parser input buffer memory 1008. Once a packet starts loading into the buffer memory 1008, pattern recognition engine (PRE) 1006 carries out the operations on the input buffer memory described in block 304 of FIG. 3. That is, protocol types and associated headers for each protocol layer that exist in the packet are determined.

The PRE searches database 1001 and the packet in buffer 1008 in order to recognize the protocols the packet contains. In one implementation, the database 1001 includes a series of linked lookup tables. Each lookup table uses eight bits of addressing. The first lookup table is always at address zero. The Pattern Recognition Engine uses a base packet offset from a control register to start the comparison. It loads this value into a current offset pointer (COP). It then reads the byte at base packet offset from the parser input buffer and uses it as an address into the first lookup table.

Each lookup table returns a word that links to another lookup table or it returns a terminal flag. If the lookup produces a recognition event the database also returns a command for the slicer. Finally it returns the value to add to the COP.

The PRE 1006 includes of a comparison engine. The comparison engine has a first stage that checks the protocol type field to determine if it is an 802.3 packet and the field should be treated as a length. If it is not a length, the protocol is checked in a second stage. The first stage is the only protocol level that is not programmable. The second stage has two full sixteen bit content addressable memories (CAMs) defined for future protocol additions.

Thus, whenever the PRE recognizes a pattern, it also generates a command for the extraction engine (also called a "slicer") 1007. The recognized patterns and the commands are sent to the extraction engine 1007 that extracts information from the packet to build the parser record. Thus, the operations of the extraction engine are those carried out in blocks 306 and 312 of FIG. 3. The commands are sent from PRE 1006 to slicer 1007 in the form of extraction instruction pointers which tell the extraction engine 1007 where to a find the instructions in the extraction operations database memory (i.e., slicer instruction database) 1002.

Thus, when the PRE 1006 recognizes a protocol it outputs both the protocol identifier and a process code to the extractor. The protocol identifier is added to the flow signature and the process code is used to fetch the first instruction from the instruction database 1002. Instructions include an operation code and usually source and destination offsets as well as a length. The offsets and length are in bytes. A typical operation is the MOVE instruction. This instruction tells the slicer 1007 to copy n bytes of data unmodified from the input buffer 1008 to the output buffer 1010. The extractor contains a byte-wise barrel shifter so that the bytes moved can be packed into the flow signature. The extractor contains another instruction called HASH. This instruction tells the extractor to copy from the input buffer 1008 to the HASH generator.

Thus these instructions are for extracting selected element (s) of the packet in the input buffer memory and transferring the data to a parser output buffer memory 1010. Some instructions also generate a hash.

The extraction engine 1007 and the PRE operate as a pipeline. That is, extraction engine 1007 performs extraction operations on data in input buffer 1008 already processed by PRE 1006 while more (i.e., later arriving) packet information is being simultaneously parsed by PRE 1006. This provides high processing speed sufficient to accommodate the high arrival rate speed of packets.

Once all the selected parts of the packet used to form the signature are extracted, the hash is loaded into parser output buffer memory 1010. Any additional payload from the packet that is required for further analysis is also included. The parser output memory 1010 is interfaced with the analyzer subsystem by analyzer interface control 1011. Once all the information of a packet is in the parser output buffer memory 1010, a data ready signal 1025 is asserted by analyzer interface control. The data from the parser subsystem **1000** is moved to the analyzer subsystem via **1013** when an analyzer ready signal **1027** is asserted.

FIG. **11** shows the hardware components and dataflow for the analyzer subsystem that performs the functions of the analyzer subsystem **303** of FIG. **3**. The analyzer is initialized prior to operation, and initialization includes loading the state processing information generated by the compilation process **310** into a database memory for the state processing, called state processor instruction database (SPID) memory **1109**.

The analyzer subsystem **1100** includes a host bus interface **1122** using an analyzer host interface controller **1118**, which in turn has access to a cache system **1115**. The cache system has bi-directional access to and from the state processor of the system **1108**. State processor **1108** is responsible for initializing the state processor instruction database memory **1109** from information given over the host bus interface **1122**.

With the SPID **1109** loaded, the analyzer subsystem **1100** receives parser records comprising packet signatures and payloads that come from the parser into the unified flow key buffer (UFKB) **1103**. UFKB is comprised of memory set up to maintain UFKB records. A UFKB record is essentially a parser record; the UFKB holds records of packets that are to be processed or that are in process. Furthermore, the UFKB provides for one or more fields to act as modifiable status flags to allow different processes to run concurrently.

Three processing engines run concurrently and access records in the UFKB **1103**: the lookup/update engine (LUE) **1107**, the state processor (SP) **1108**, and the flow insertion and deletion engine (FIDE) **1110**. Each of these is implemented by one or more finite state machines (FSM's). There is bi-directional access between each of the finite state machines and the unified flow key buffer **1103**. The UFKB record includes a field that stores the packet sequence number, and another that is filled with state information in the form of a program counter for the state processor **1108** that implements state processing **328**. The status flags of the UFKB for any entry includes that the LUE is done and that the LUE is transferring processing of the entry to the state processor. The LUE done indicator is also used to indicate what the next entry is for the LUE. There also is provided a flag to indicate that the state processor is done with the current flow and to indicate what the next entry is for the state processor. There also is provided a flag to indicate the state processor is transferring processing of the UFKB-entry to the flow insertion and deletion engine.

A new UFKB record is first processed by the LUE **1107**. A record that has been processed by the LUE **1107** may be processed by the state processor **1108**, and a UFKB record data may be processed by the flow insertion/deletion engine **1110** after being processed by the state processor **1108** or only by the LUE. Whether or not a particular engine has been applied to any unified flow key buffer entry is determined by status fields set by the engines upon completion. In one embodiment, a status flag in the UFKB-entry indicates whether an entry is new or found. In other embodiments, the LUE issues a flag to pass the entry to the state processor for processing, and the required operations for a new record are included in the SP instructions.

Note that each UFKB-entry may not need to be processed by all three engines. Furthermore, some UFKB entries may need to be processed more than once by a particular engine.

Each of these three engines also has bi-directional access to a cache subsystem **1115** that includes a caching engine.

Cache **1115** is designed to have information flowing in and out of it from five different points within the system: the three engines, external memory via a unified memory controller (UMC) **1119** and a memory interface **1123**, and a microprocessor via analyzer host interface and control unit (ACIC) **1118** and host interface bus (HIB) **1122**. The analyzer microprocessor (or dedicated logic processor) can thus directly insert or modify data in the cache.

The cache subsystem **1115** is an associative cache that includes a set of content addressable memory cells (CAMs) each including an address portion and a pointer portion pointing to the cache memory (e.g., RAM) containing the cached flow-entries. The CAMs are arranged as a stack ordered from a top CAM to a bottom CAM. The bottom CAM's pointer points to the least recently used (LRU) cache memory entry. Whenever there is a cache miss, the contents of cache memory pointed to by the bottom CAM are replaced by the flow-entry from the flow-entry database **324**. This now becomes the most recently used entry, so the contents of the bottom CAM are moved to the top CAM and all CAM contents are shifted down. Thus, the cache is an associative cache with a true LRU replacement policy.

The LUE **1107** first processes a UFKB-entry, and basically performs the operation of blocks **314** and **316** in FIG. **3**. A signal is provided to the LUE to indicate that a "new" UFKB-entry is available. The LUE uses the hash in the UFKB-entry to read a matching bin of up to four buckets from the cache. The cache system attempts to obtain the matching bin. If a matching bin is not in the cache, the cache **1115** makes the request to the UMC **1119** to bring in a matching bin from the external memory.

When a flow-entry is found using the hash, the LUE **1107** looks at each bucket and compares it using the signature to the signature of the UFKB-entry until there is a match or there are no more buckets.

If there is no match, or if the cache failed to provide a bin of flow-entries from the cache, a time stamp in set in the flow key of the UFKB record, a protocol identification and state determination is made using a table that was loaded by compilation process **310** during initialization, the status for the record is set to indicate the LUE has processed the record, and an indication is made that the UFKB-entry is ready to start state processing. The identification and state determination generates a protocol identifier which in the preferred embodiment is a "jump vector" for the state processor which is kept by the UFKB for this UFKB-entry and used by the state processor to start state processing for the particular protocol. For example, the jump vector jumps to the subroutine for processing the state.

If there was a match, indicating that the packet of the UFKB-entry is for a previously encountered flow, then a calculator component enters one or more statistical measures stored in the flow-entry, including the timestamp. In addition, a time difference from the last stored timestamp may be stored, and a packet count may be updated. The state of the flow is obtained from the flow-entry is examined by looking at the protocol identifier stored in the flow-entry of database **324**. If that value indicates that no more classification is required, then the status for the record is set to indicate the LUE has processed the record. In the preferred embodiment, the protocol identifier is a jump vector for the state processor to a subroutine to state processing the protocol, and no more classification is indicated in the preferred embodiment by the jump vector being zero. If the protocol identifier indicates more processing, then an indication is made that the UFKB-entry is ready to start state processing and the status for the record is set to indicate the LUE has processed the record.

The state processor **1108** processes information in the cache system according to a UFKB-entry after the LUE has completed. State processor **1108** includes a state processor program counter SPPC that generates the address in the state processor instruction database **1109** loaded by compiler process **310** during initialization. It contains an Instruction Pointer (SPIP) which generates the SPID address. The instruction pointer can be incremented or loaded from a Jump Vector Multiplexor which facilitates conditional branching. The SPIP can be loaded from one of three sources: (1) A protocol identifier from the UFKB, (2) an immediate jump vector form the currently decoded instruction, or (3) a value provided by the arithmetic logic unit (SPALU) included in the state processor.

Thus, after a Flow Key is placed in the UFKB by the LUE with a known protocol identifier, the Program Counter is initialized with the last protocol recognized by the Parser. This first instruction is a jump to the subroutine which analyzes the protocol that was decoded.

The State Processor ALU (SPALU) contains all the Arithmetic, Logical and String Compare functions necessary to implement the State Processor instructions. The main blocks of the SPALU are: The A and B Registers, the Instruction Decode & State Machines, the String Reference Memory the Search Engine, an Output Data Register and an Output Control Register.

The Search Engine in turn contains the Target Search Register set, the Reference Search Register set, and a Compare block which compares two operands by exclusive-or-ing them together.

Thus, after the UFKB sets the program counter, a sequence of one or more state operations are be executed in state processor **1108** to further analyze the packet that is in the flow key buffer entry for this particular packet.

FIG. **13** describes the operation of the state processor **1108**. The state processor is entered at **1301** with a unified flow key buffer entry to be processed. The UFKB-entry is new or corresponding to a found flow-entry. This UFKB-entry is retrieved from unified flow key buffer **1103** in **1301**. In **1303**, the protocol identifier for the UFKB-entry is used to set the state processor's instruction counter. The state processor **1108** starts the process by using the last protocol recognized by the parser subsystem **301** as an offset into a jump table. The jump table takes us to the instructions to use for that protocol. Most instructions test something in the unified flow key buffer or the flow-entry if it exists. The state processor **1108** may have to test bits, do comparisons, add or subtract to perform the test.

The first state processor instruction is fetched in **1304** from the state processor instruction database memory **1109**. The state processor performs the one or more fetched operations (**1304**). In our implementation, each single state processor instruction is very primitive (e.g., a move, a compare, etc.), so that many such instructions need to be performed on each unified flow key buffer entry. One aspect of the state processor is its ability to search for one or more (up to four) reference strings in the payload part of the UFKB entry. This is implemented by a search engine component of the state processor responsive to special searching instructions.

In **1307**, a check is made to determine if there are any more instructions to be performed for the packet. If yes, then in **1308** the system sets the state processor instruction pointer (SPIP) to obtain the next instruction. The SPIP may be set by an immediate jump vector in the currently decoded instruction, or by a value provided by the SPALU during processing.

The next instruction to be performed is now fetched (**1304**) for execution. This state processing loop between **1304** and **1307** continues until there are no more instructions to be performed.

At this stage, a check is made in **1309** if the processing on this particular packet has resulted in a final state. That is, is the analyzer is done processing not only for this particular packet, but for the whole flow to which the packet belongs, and the flow is fully determined. If indeed there are no more states to process for this flow, then in **1311** the processor finalizes the processing. Some final states may need to put a state in place that tells the system to remove a flow—for example, if a connection disappears from a lower level connection identifier. In that case, in **1311**, a flow removal state is set and saved in the flow-entry. The flow removal state may be a NOP (no-op) instruction which means there are no removal instructions.

Once the appropriate flow removal instruction as specified for this flow (a NOP or otherwise) is set and saved, the process is exited at **1313**. The state processor **1108** can now obtain another unified flow key buffer entry to process.

If at **1309** it is determined that processing for this flow is not completed, then in **1310** the system saves the state processor instruction pointer in the current flow-entry in the current flow-entry. That will be the next operation that will be performed the next time the LRE **1107** finds packet in the UFKB that matches this flow. The processor now exits processing this particular unified flow key buffer entry at **1313**.

Note that state processing updates information in the unified flow key buffer **1103** and the flow-entry in the cache. Once the state processor is done, a flag is set in the UFKB for the entry that the state processor is done. Furthermore, If the flow needs to be inserted or deleted from the database of flows, control is then passed on to the flow insertion/deletion engine **1110** for that flow signature and packet entry. This is done by the state processor setting another flag in the UFKB for this UFKB-entry indicating that the state processor is passing processing of this entry to the flow insertion and deletion engine.

The flow insertion and deletion engine **1110** is responsible for maintaining the flow-entry database. In particular, for creating new flows in the flow database, and deleting flows from the database so that they can be reused.

The process of flow insertion is now described with the aid of FIG. **12**. Flows are grouped into bins of buckets by the hash value. The engine processes a UFKB-entry that may be new or that the state processor otherwise has indicated needs to be created. FIG. **12** shows the case of a new entry being created. A conversation record bin (preferably containing 4 buckets for four records) is obtained in **1203**. This is a bin that matches the hash of the UFKB, so this bin may already have been sought for the UFKB-entry by the LUE. In **1204** the FIDE **1110** requests that the record bin/bucket be maintained in the cache system **1115**. If in **1205** the cache system **1115** indicates that the bin/bucket is empty, step **1207** inserts the flow signature (with the hash) into the bucket and the bucket is marked "used" in the cache engine of cache **1115** using a timestamp that is maintained throughout the process. In **1209**, the FIDE **1110** compares the bin and bucket record flow signature to the packet to verify that all the elements are in place to complete the record. In **1211** the system marks the record bin and bucket as "in process" and as "new" in the cache system (and hence in the external memory). In **1212**, the initial statistical measures for the flow-record are set in the cache system. This in the preferred embodiment clears the set of counters used to maintain statistics, and may perform other procedures for statistical operations requires by the analyzer for the first packet seen for a particular flow.

Back in step **1205**, if the bucket is not empty, the FIDE **1110** requests the next bucket for this particular bin in the cache system. If this succeeds, the processes of **1207**, **1209**, **1211** and **1212** are repeated for this next bucket. If at **1208**, there is no valid bucket, the unified flow key buffer entry for the packet is set as "drop," indicating that the system cannot process the particular packet because there are no buckets left in the system. The process exits at **1213**. The FIDE **1110** indicates to the UFKB that the flow insertion and deletion operations are completed for this UFKB-entry. This also lets the UFKB provide the FIDE with the next UFKB record.

Once a set of operations is performed on a unified flow key buffer entry by all of the engines required to access and manage a particular packet and its flow signature, the unified flow key buffer entry is marked as "completed." That element will then be used by the parser interface for the next packet and flow signature coming in from the parsing and extracting system.

All flow-entries are maintained in the external memory and some are maintained in the cache **1115**. The cache system **1115** is intelligent enough to access the flow database and to understand the data structures that exists on the other side of memory interface **1123**. The lookup/update engine **1107** is able to request that the cache system pull a particular flow or "buckets" of flows from the unified memory controller **1119** into the cache system for further processing. The state processor **1108** can operate on information found in the cache system once it is looked up by means of the lookup/update engine request, and the flow insertion/deletion engine **1110** can create new entries in the cache system if required based on information in the unified flow key buffer **1103**. The cache retrieves information as required from the memory through the memory interface **1123** and the unified memory controller **1119**, and updates information as required in the memory through the memory controller **1119**.

There are several interfaces to components of the system external to the module of FIG. **11** for the particular hardware implementation. These include host bus interface **1122**, which is designed as a generic interface that can operate with any kind of external processing system such as a microprocessor or a multiplexor (MUX) system. Consequently, one can connect the overall traffic classification system of FIGS. **11** and **12** into some other processing system to manage the classification system and to extract data gathered by the system.

The memory interface **1123** is designed to interface to any of a variety of memory systems that one may want to use to store the flow-entries. One can use different types of memory systems like regular dynamic random access memory (DRAM), synchronous DRAM, synchronous graphic memory (SGRAM), static random access memory (SRAM), and so forth.

FIG. **10** also includes some "generic" interfaces. There is a packet input interface **1012**—a general interface that works in tandem with the signals of the input buffer interface control **1022**. These are designed so that they can be used with any kind of generic systems that can then feed packet information into the parser. Another generic interface is the interface of pipes **1031** and **1033** respectively out of and into host interface multiplexor and control registers **1005**. This enables the parsing system to be managed by an external system, for example a microprocessor or another kind of external logic, and enables the external system to program and otherwise control the parser.

The preferred embodiment of this aspect of the invention is described in a hardware description language (HDL) such as VHDL or Verilog. It is designed and created in an HDL so that it may be used as a single chip system or, for instance, integrated into another general-purpose system that is being designed for purposes related to creating and analyzing traffic within a network. Verilog or other HDL implementation is only one method of describing the hardware.

In accordance with one hardware implementation, the elements shown in FIGS. **10** and **11** are implemented in a set of six field programmable logic arrays (FPGA's). The boundaries of these FPGA's are as follows. The parsing subsystem of FIG. **10** is implemented as two FPGAS; one FPGA, and includes blocks **1006**, **1008** and **1012**, parts of **1005**, and memory **1001**. The second FPGA includes **1002**, **1007**, **1013**, **1011** parts of **1005**. Referring to FIG. **11**, the unified look-up buffer **1103** is implemented as a single FPGA. State processor **1108** and part of state processor instruction database memory **1109** is another FPGA. Portions of the state processor instruction database memory **1109** are maintained in external SRAM's. The lookup/update engine **1107** and the flow insertion/deletion engine **1110** are in another FPGA. The sixth FPGA includes the cache system **1115**, the unified memory control **1119**, and the analyzer host interface and control **1118**.

Note that one can implement the system as one or more VSLI devices, rather than as a set of application specific integrated circuits (ASIC's) such as FPGA's. It is anticipated that in the future device densities will continue to increase, so that the complete system may eventually form a sub-unit (a "core") of a larger single chip unit.

Operation of the Invention

FIG. **15** shows how an embodiment of the network monitor **300** might be used to analyze traffic in a network **102**. Packet acquisition device **1502** acquires all the packets from a connection point **121** on network **102** so that all packets passing point **121** in either direction are supplied to monitor **300**. Monitor **300** comprises the parser sub-system **301**, which determines flow signatures, and analyzer sub-system **303** that analyzes the flow signature of each packet. A memory **324** is used to store the database of flows that are determined and updated by monitor **300**. A host computer **1504**, which might be any processor, for example, a general-purpose computer, is used to analyze the flows in memory **324**. As is conventional, host computer **1504** includes a memory, say RAM, shown as host memory **1506**. In addition, the host might contain a disk. In one application, the system can operate as an RMON probe, in which case the host computer is coupled to a network interface card **1510** that is connected to the network **102**.

The preferred embodiment of the invention is supported by an optional Simple Network Management Protocol (SNMP) implementation. FIG. **15** describes how one would, for example, implement an RMON probe, where a network interface card is used to send RMON information to the network. Commercial SNMP implementations also are available, and using such an implementation can simplify the process of porting the preferred embodiment of the invention to any platform.

In addition, MEB Compilers are available. An MIB Compiler is a tool that greatly simplifies the creation and maintenance of proprietary MIB extensions.

Examples of Packet Elucidation

Monitor **300**, and in particular, analyzer **303** is capable of carrying out state analysis for packet exchanges that are commonly referred to as "server announcement" type exchanges. Server announcement is a process used to ease communications between a server with multiple applications that can all be simultaneously accessed from multiple clients. Many applications use a server announcement process as a means of multiplexing a single port or socket into many applications and services. With this type of exchange, messages are sent on the network, in either a broadcast or multicast approach, to announce a server and application, and all stations in the network may receive and decode these messages. The messages enable the stations to derive the appropriate connection point for communicating that particular application with the particular server. Using the server announcement method, a particular application communicates using a service channel, in the form of a TCP or UDP socket or port as in the IP protocol suite, or using a SAP as in the Novell IPX protocol suite.

The analyzer **303** is also capable of carrying out "in-stream analysis" of packet exchanges. The "in-stream analysis" method is used either as a primary or secondary recognition process. As a primary process, in-stream analysis assists in extracting detailed information which will be used to further recognize both the specific application and application component. A good example of in-stream analysis is any Web-based application. For example, the commonly used PointCast Web information application can be recognized using this process; during the initial connection between a PointCast server and client, specific key tokens exist in the data exchange that will result in a signature being generated to recognize PointCast.

The in-stream analysis process may also be combined with the server announcement process. In many cases in-stream analysis will augment other recognition processes. An example of combining in-stream analysis with server announcement can be found in business applications such as SAP and BAAN.

"Session tracking" also is known as one of the primary processes for tracking applications in client/server packet exchanges. The process of tracking sessions requires an initial connection to a predefined socket or port number. This method of communication is used in a variety of transport layer protocols. It is most commonly seen in the TCP and UDP transport protocols of the IP protocol.

During the session tracking, a client makes a request to a server using a specific port or socket number. This initial request will cause the server to create a TCP or UDP port to exchange the remainder of the data between the client and the server. The server then replies to the request of the client using this newly created port. The original port used by the client to connect to the server will never be used again during this data exchange.

One example of session tracking is TFTP (Trivial File Transfer Protocol), a version of the TCP/IP FTP protocol that has no directory or password capability. During the client/server exchange process of TFTP, a specific port (port number **69**) is always used to initiate the packet exchange. Thus, when the client begins the process of communicating, a request is made to UDP port **69**. Once the server receives this request, a new port number is created on the server. The server then replies to the client using the new port. In this example, it is clear that in order to recognize TFTP; network monitor **300** analyzes the initial request from the client and generates a signature for it. Monitor **300** uses that signature to recognize the reply. Monitor **300** also analyzes the reply from the server with the key port information, and uses this to create a signature for monitoring the remaining packets of this data exchange.

Network monitor **300** can also understand the current state of particular connections in the network. Connection-oriented exchanges often benefit from state tracking to correctly identify the application. An example is the common TCP transport protocol that provides a reliable means of sending information between a client and a server. When a data exchange is initiated, a TCP request for synchronization message is sent. This message contains a specific sequence number that is used to track an acknowledgement from the server. Once the server has acknowledged the synchronization request, data may be exchanged between the client and the server. When communication is no longer required, the client sends a finish or complete message to the server, and the server acknowledges this finish request with a reply containing the sequence numbers from the request. The states of such a connection-oriented exchange relate to the various types of connection and maintenance messages.

Server Announcement Example

The individual methods of server announcement protocols vary. However, the basic underlying process remains similar. A typical server announcement message is sent to one or more clients in a network. This type of announcement message has specific content, which, in another aspect of the invention, is salvaged and maintained in the database of flow-entries in the system. Because the announcement is sent to one or more stations, the client involved in a future packet exchange with the server will make an assumption that the information announced is known, and an aspect of the inventive monitor is that it too can make the same assumption.

Sun-RPC is the implementation by Sun Microsystems, Inc. (Palo Alto, Calif.) of the Remote Procedure Call (RPC), a programming interface that allows one program to use the services of another on a remote machine. A Sun-RPC example is now used to explain how monitor **300** can capture server announcements.

A remote program or client that wishes to use a server or procedure must establish a connection, for which the RPC protocol can be used.

Each server running the Sun-RPC protocol must maintain a process and database called the port Mapper. The port Mapper creates a direct association between a Sun-RPC program or application and a TCP or UDP socket or port (for TCP or UDP implementations). An application or program number is a 32-bit unique identifier assigned by ICANN (the Internet Corporation for Assigned Names and Numbers, www.icann.org), which manages the huge number of parameters associated with Internet protocols (port numbers, router protocols, multicast addresses, etc.) Each port Mapper on a Sun-RPC server can present the mappings between a unique program number and a specific transport socket through the use of specific request or a directed announcement. According to ICANN, port number **111** is associated with Sun RPC.

As an example, consider a client (e.g., CLIENT 3 shown as **106** in FIG. **1**) making a specific request to the server (e.g., SERVER 2 of FIG. **1**, shown as **110**) on a predefined UDP or TCP socket. Once the port Mapper process on the sun RPC server receives the request, the specific mapping is returned in a directed reply to the client.

1. A client (CLIENT 3, **106** in FIG. **1**) sends a TCP packet to SERVER 2 (**110** in FIG. **1**) on port **111**, with an RPC Bind

Lookup Request (rpcBindLookup). TCP or UDP port 111 is always associated Sun RPC. This request specifies the program (as a program identifier), version, and might specify the protocol (UDP or TCP).

2. The server SERVER 2 (110 in FIG. 1) extracts the program identifier and version identifier from the request. The server also uses the fact that this packet came in using the TCP transport and that no protocol was specified, and thus will use the TCP protocol for its reply.

3. The server 110 sends a TCP packet to port number 111, with an RPC Bind Lookup Reply. The reply contains the specific port number (e.g., port number 'port') on which future transactions will be accepted for the specific RPC program identifier (e.g., Program 'program') and the protocol (UDP or TCP) for use.

It is desired that from now on every time that port number 'port' is used, the packet is associated with the application program 'program' until the number 'port' no longer is to be associated with the program 'program'. Network monitor 300 by creating a flow-entry and a signature includes a mechanism for remembering the exchange so that future packets that use the port number 'port' will be associated by the network monitor with the application program 'program'.

In addition to the Sun RPC Bind Lookup request and reply, there are other ways that a particular program—say 'program'-might be associated with a particular port number, for example number 'port'. One is by a broadcast announcement of a particular association between an application service and a port number, called a Sun RPC portMapper Announcement. Another, is when some server—say the same SERVER 2—replies to some client—say CLIENT 1—requesting some portMapper assignment with a RPC portMapper Reply. Some other client—say CLIENT 2—might inadvertently see this request, and thus know that for this particular server, SERVER 2, port number 'port' is associated with the application service 'program', It is desirable for the network monitor 300 to be able to associate any packets to SERVER 2 using port number 'port' with the application program 'program',

FIG. 9 represents a dataflow 900 of some operations in the monitor 300 of FIG. 3 for Sun Remote Procedure Call. Suppose a client 106 (e.g., CLIENT 3 in FIG. 1) is communicating via its interface to the network 118 to a server 110 (e.g., SERVER 2 in FIG. 1) via the server's interface to the network 116. Further assume that Remote Procedure Call is used to communicate with the server 110. One path in the data flow 900 starts with a step 910 that a Remote Procedure Call bind lookup request is issued by client 106 and ends with the server state creation step 904. Such RPC bind lookup request includes values for the 'program,' 'version,' and 'protocol' to use, e.g., TCP or UDP. The process for Sun RPC analysis in the network monitor 300 includes the following aspects.:

Process 909: Extract the 'program,' 'version,' and 'protocol' (UDP or TCP).

Extract the TCP or UDP port (process 909) which is 111 indicating Sun RPC.

Process 908: Decode the Sun RPC packet. Check RPC type field for ID. If value is portMapper, save paired socket (i.e., dest for destination address, src for source address). Decode ports and mapping, save ports with socket/addr key. There may be more than one pairing per mapper packet. Form a signature (e.g., a key). A flow-entry is created in database 324. The saving of the request is now complete.

At some later time, the server (process 907) issues a RPC bind lookup reply. The packet monitor 300 will extract a signature from the packet and recognize it from the previously stored flow. The monitor will get the protocol port number (906) and lookup the request (905). A new signature (i.e., a key) will be created and the creation of the server state (904) will be stored as an entry identified by the new signature in the flow-entry database. That signature now may be used to identify packets associated with the server.

The server state creation step 904 can be reached not only from a Bind Lookup Request/Reply pair, but also from a RPC Reply portMapper packet shown as 901 or an RPC Announcement portMapper shown as 902. The Remote Procedure Call protocol can announce that it is able to provide a particular application service. Embodiments of the present invention preferably can analyze when an exchange occurs between a client and a server, and also can track those stations that have received the announcement of a service in the network.

The RPC Announcement portMapper announcement 902 is a broadcast. Such causes various clients to execute a similar set of operations, for example, saving the information obtained from the announcement. The RPC Reply portMapper step 901 could be in reply to a portMapper request, and is also broadcast. It includes all the service parameters.

Thus monitor 300 creates and saves all such states for later classification of flows that relate to the particular service 'program',

FIG. 2 shows how the monitor 300 in the example of Sun RPC builds a signature and flow states. A plurality of packets 206–209 are exchanged, e.g., in an exemplary Sun Microsystems Remote Procedure Call protocol. A method embodiment of the present invention might generate a pair of flow signatures, "signature-1" 210 and "signature-2" 212, from information found in the packets 206 and 207 which, in the example, correspond to a Sun RPC Bind Lookup request and reply, respectively.

Consider first the Sun RPC Bind Lookup request. Suppose packet 206 corresponds to such a request sent from CLIENT 3 to SERVER 2. This packet contains important information that is used in building a signature according to an aspect of the invention. A source and destination network address occupy the first two fields of each packet, and according to the patterns in pattern database 308, the flow signature (shown as KEY1 230 in FIG. 2) will also contain these two fields, so the parser subsystem 301 will include these two fields in signature KEY 1 (230). Note that in FIG. 2, if an address identifies the client 106 (shown also as 202), the label used in the drawing is "C_1". If such address identifies the server 110 (shown also as server 204), the label used in the drawing is "S_1". The first two fields 214 and 215 in packet 206 are "S_1" and "C_1" because packet 206 is provided from the server 110 and is destined for the client 106. Suppose for this example, "S_1" is an address numerically less than address "C_1". A third field "p^1" 216 identifies the particular protocol being used, e.g., TCP, UDP, etc.

In packet 206, a fourth field 217 and a fifth field 218 are used to communicate port numbers that are used. The conversation direction determines where the port number field is. The diagonal pattern in field 217 is used to identify a source-port pattern, and the hash pattern in field 218 is used to identify the destination-port pattern. The order indicates the client-server message direction. A sixth field denoted "i^1" 219 is an element that is being requested by the client from the server. A seventh field denoted "s_1a" 220 is the service requested by the client from server 110. The following eighth field “QA” **221** (for question mark) indicates that the client **106** wants to know what to use to access application “s_1a”. A tenth field “QP” **223** is used to indicate that the client wants the server to indicate what protocol to use for the particular application.

Packet **206** initiates the sequence of packet exchanges, e.g., a RPC Bind Lookup Request to SERVER 2. It follows a well-defined format, as do all the packets, and is transmitted to the server **110** on a well-known service connection identifier (port **111** indicating Sun RPC).

Packet **207** is the first sent in reply to the client **106** from the server. It is the RPC Bind Lookup Reply as a result of the request packet **206**.

Packet **207** includes ten fields **224–233**. The destination and source addresses are carried in fields **224** and **225**, e.g., indicated “C_1” and “S_1”, respectively. Notice the order is now reversed, since the client-server message direction is from the server **110** to the client **106**. The protocol “p^1” is used as indicated in field **226**. The request “i^1” is in field **229**. Values have been filled in for the application port number, e.g., in field **233** and protocol “p^2” in field **233**.

The flow signature and flow states built up as a result of this exchange are now described. When the packet monitor **300** sees the request packet **206** from the client, a first flow signature **210** is built in the parser subsystem **301** according to the pattern and extraction operations database **308**. This signature **210** includes a destination and a source address **240** and **241**. One aspect of the invention is that the flow keys are built consistently in a particular order no matter what the direction of conversation. Several mechanisms may be used to achieve this. In the particular embodiment, the numerically lower address is always placed before the numerically higher address. Such least to highest order is used to get the best spread of signatures and hashes for the lookup operations. In this case, therefore, since we assume “S_1”<“C_1”, the order is address “S_1” followed by client address “C_1”. The next field used to build the signature is a protocol field **242** extracted from packet **206**’s field **216**, and thus is the protocol “p^1”. The next field used for the signature is field **243**, which contains the destination source port number shown as a crosshatched pattern from the field **218** of the packet **206**. This pattern will be recognized in the payload of packets to derive how this packet or sequence of packets exists as a flow. In practice, these may be TCP port numbers, or a combination of TCP port numbers. In the case of the Sun RPC example, the crosshatch represents a set of port numbers of UDS for p^1 that will be used to recognize this flow (e.g., port **111**). Port **111** indicates this is Sun RPC. Some applications, such as the Sun RPC Bind Lookups, are directly determinable (“known”) at the parser level. So in this case, the signature KEY-1 points to a known application denoted “a^1” (Sun RPC Bind Lookup), and a next—state that the state processor should proceed to for more complex recognition jobs, denoted as state “st_D” is placed in the field **245** of the flow-entry.

When the Sun RPC Bind Lookup reply is acquired, a flow signature is again built by the parser. This flow signature is identical to KEY-1. Hence, when the signature enters the analyzer subsystem **303** from the parser subsystem **301**, the complete flow-entry is obtained, and in this flow-entry indicates state “st_D”. The operations for state “st_D” in the state processor instruction database **326** instructs the state processor to build and store a new flow signature, shown as KEY-2 (**212**) in FIG. 2. This flow signature built by the state processor also includes the destination and a source addresses **250** and **251**, respectively, for server “S_1” followed by (the numerically higher address) client “C_1”. A protocol field **252** defines the protocol to be used, e.g., “p^2” which is obtained from the reply packet. A field **253** contains a recognition pattern also obtained from the reply packet. In this case, the application is Sun RPC, and field **254** indicates this application “a^2”. A next-state field **255** defines the next state that the state processor should proceed to for more complex recognition jobs, e.g., a state “st^1”. In this particular example, this is a final state. Thus, KEY-2 may now be used to recognize packets that are in any way associated with the application “a^2”. Two such packets **208** and **209** are shown, one in each direction. They use the particular application service requested in the original Bind Lookup Request, and each will be recognized because the signature KEY-2 will be built in each case.

The two flow signatures **210** and **212** always order the destination and source address fields with server “S_1” followed by client “C_1”. Such values are automatically filled in when the addresses are first created in a particular flow signature. Preferably, large collections of flow signatures are kept in a lookup table in a least-to-highest order for the best spread of flow signatures and hashes.

Thereafter, the client and server exchange a number of packets, e.g., represented by request packet **208** and response packet **209**. The client **106** sends packets **208** that have a destination and source address S_1 and C_1, in a pair of fields **260** and **261**. A field **262** defines the protocol as “p^2”, and a field **263** defines the destination port number.

Some network-server application recognition jobs are so simple that only a single state transition has to occur to be able to pinpoint the application that produced the packet. Others require a sequence of state transitions to occur in order to match a known and predefined climb from state-to-state.

Thus the flow signature for the recognition of application “a^2” is automatically set up by predefining what packet-exchange sequences occur for this example when a relatively simple Sun Microsystems Remote Procedure Call bind lookup request instruction executes. More complicated exchanges than this may generate more than two flow signatures and their corresponding states. Each recognition may involve setting up a complex state transition diagram to be traversed before a “final” resting state such as “st_1” in field **255** is reached. All these are used to build the final set of flow signatures for recognizing a particular application in the future.

Embodiments of the present invention automatically generate flow signatures with the necessary recognition patterns and state transition climb procedure. Such comes from analyzing packets according to parsing rules, and also generating state transitions to search for. Applications and protocols, at any level, are recognized through state analysis of sequences of packets.

Note that one in the art will understand that computer networks are used to connect many different types of devices, including network appliances such as telephones, “Internet” radios, pagers, and so forth. The term computer as used herein encompasses all such devices and a computer network as used herein includes networks of such computers.

Although the present invention has been described in terms of the presently preferred embodiments, it is to be understood that the disclosure is not to be interpreted as limiting. Various alterations and modifications will no doubt become apparent to those or ordinary skill in the art after having read the above disclosure. Accordingly, it is intended that the claims be interpreted as covering all alterations and modifications as fall within the true spirit and scope of the present invention.

What is claimed is:

1. A packet monitor for examining packets passing through a connection point on a computer network in real-time, the packets provided ,to the packet monitor via a packet acquisition device connected to the connection point, the packet monitor comprising:

(a) a packet-buffer memory configured to accept a packet from the packet acquisition device;

(b) a parsing/extraction operations memory configured to store a database of parsing/extraction operations that includes information describing how to determine at least one of the protocols used in a packet from data in the packet;

(c) a parser subsystem coupled to the packet buffer and to the pattern/extraction operations memory, the parser subsystem configured to examine the packet accepted by the buffer, extract selected portions of the accepted packet, and form a function of the selected portions sufficient to identify that the accepted packet is part of a conversational flow-sequence;

(d) a memory storing a flow-entry database including a plurality of flow-entries for conversational flows encountered by the monitor;

(e) a lookup engine connected to the parser subsystem and to the flow-entry database, and configured to determine using at least some of the selected portions of the accepted packet if there is an entry in the flow-entry database for the conversational flow sequence of the accepted packet;

(f) a state patterns/operations memory configured to store a set of predefined state transition patterns and state operations such that traversing a particular transition pattern as a result of a particular conversational flow-sequence of packets indicates that the particular conversational flow-sequence is associated with the operation of a particular application program, visiting each state in a traversal including carrying out none or more predefined state operations;

(g) a protocol/state identification mechanism coupled to the state patterns/operations memory and to the lookup engine, the protocol/state identification engine configured to determine the protocol and state of the conversational flow of the packet; and

(h) a state processor coupled to the flow-entry database, the protocol/state identification engine, and to the state patterns/operations memory, the state processor, configured to carry out any state operations specified in the state patterns/operations memory for the protocol and state of the flow of the packet,

the carrying out of the state operations furthering the process of identifying which application program is associated with the conversational flow-sequence of the packet, the state processor progressing through a series of states and state operations until there are no more state operations to perform for the accepted packet, in which case the state processor updates the flow-entry, or until a final state is reached that indicates that no more analysis of the flow is required, in which case the result of the analysis is announced.

2. A packet monitor according to claim 1, wherein the flow-entry includes the state of the flow, such that the protocol/state identification mechanism determines the state of the packet from the flow-entry in the case that the lookup engine finds a flow-entry for the flow of the accepted packet.

3. A packet monitor according to claim 1, wherein the parser subsystem includes a mechanism for building a hash from the selected portions, and wherein the hash is used by the lookup engine to search the flow-entry database, the hash designed to spread the flow-entries across the flow-entry database.

4. A packet monitor according to claim 1, further comprising:

a compiler processor coupled to the parsing/extraction operations memory, the compiler processor configured to run a compilation process that includes:

receiving commands in a high-level protocol description language that describe the protocols that may be used in packets encountered by the monitor, and

translating the protocol description language commands into a plurality of parsing/extraction operations that are initialized into the parsing/extraction operations memory.

5. A packet monitor according to claim 4, wherein the protocol description language commands also describe a correspondence between a set of one or more application programs and the state transition patterns/operations that occur as a result of particular conversational flow-sequences associated with an application program, wherein the compiler processor is also coupled to the state patterns/operations memory, and wherein the compilation process further includes translating the protocol description language commands into a plurality of state patterns and state operations that are initialized into the state patterns/operations memory.

6. A packet monitor according to claim 1, further comprising:

a cache memory coupled to and between the lookup engine and the flow-entry database providing for fast access of a set of likely-to-be-accessed flow-entries from the flow-entry database.

7. A packet monitor according to claim 6, wherein the cache functions as a fully associative, least-recently-used cache memory.

8. A packet monitor according to claim 7, wherein the cache functions as a fully associative, least-recently-used cache memory and includes content addressable memories configured as a stack.

9. A packet monitor according to claim 1, wherein one or more statistical measures about a flow are stored in each flow-entry, the packet monitor further comprising:

a calculator for updating the statistical measures in a flow-entry of the accepted packet.

10. A packet monitor according to claim 9, wherein, when the application program of a flow is determined, one or more network usage metrics related to said application and determined from the statistical measures are presented to a user for network performance monitoring.

* * * * *