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 18, 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: *09/609,179*
FILING DATE: *June 30, 2000*
PATENT NUMBER: *6,665,725*
ISSUE DATE: *December 16, 2003*

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

P. SWAIN
Certifying Officer

PART (1) OF (2) PART(S)

ISSUE CLASSIFICATION	Class	Subclass
	709	230

PATENT NUMBER

6665725

6665725

U.S. **UTILITY** Patent Application

O.I.P.E.	PATENT DATE
SCANNED ✓ Q.A. ✓	DEC 16 2003

APPLICATION NO.	CONT/PRIOR	CLASS	SUBCLASS	ART UNIT	EXAMINER
09/609179	D	709		2152	

APPLICANTS

TITLE

Certificate of Correction

JUN 29 2004

PTO-2040 12/99

ISSUING CLASSIFICATION

ORIGINAL		CROSS REFERENCE(S)						
CLASS	SUBCLASS	CLASS	SUBCLASS (ONE SUBCLASS PER BLOCK)					
709	230	709	246	228				
INTERNATIONAL CLASSIFICATION		370	389					
G06F 13/00								

☐ Continued on Issue Slip Inside File Jacket

10/30/03 Formal Drawings (20 shts) set 6/30/00

☐ TERMINAL DISCLAIMER	DRAWINGS			CLAIMS ALLOWED	
	Sheets Drwg.	Figs. Drwg.	Print Fig.	Total Claims	Print Claim for O.G.
	20	22	15	17	1

☐ The term of this patent subsequent to _____ (date) has been disclaimed.	KHANH QUANG DINH (Assistant Examiner) 6/19/03 (Date)	NOTICE OF ALLOWANCE MAILED: 7/01/03
☐ The term of this patent shall not extend beyond the expiration date of U.S Patent. No. _____	HOSAIN T. ALAM PRIMARY EXAMINER (Primary Examiner) 6/24/03 (Date)	ISSUE FEE ✓ Amount Due #1300.00 / Date Paid 9-24-03
☐ The terminal ____ months of this patent have been disclaimed.	R. Johnson (Legal Instruments Examiner) 7-2-03 (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

FORMAL DRAWINGS

(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	FILING DATE	CLASS	GROUP ART UNIT	ATTORNEY DOCKET NO.
09/609,179	06/30/2000 RULE _	709	~~2756~~ 2155	APPT-001-2

APPLICANTS

Russell S. Dietz, San Jose, CA ;
Andrew A. Koppenhaver, Littleton, CO ;
James F. Torgerson, Andover, MN ;

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

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

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

NONE, LOD

IF REQUIRED, FOREIGN FILING LICENSE GRANTED ** 08/23/2000

Foreign Priority claimed	STATE OR COUNTRY	SHEETS DRAWING	TOTAL CLAIMS	INDEPENDENT CLAIMS
☐ yes ☑ no 35 USC 119 (a-d) conditions met ☐ yes ☑ no ☐ Met after Allowance Verified and Acknowledged _____ Examiner's Signature Initials LOD	CA	20	18	1

ADDRESS

Dov Rosenfeld
5507 College Avenue
Suite 2
Oakland ,CA 94618

TITLE

Method and apparatus for monitoring traffic in a network

FILING FEE RECEIVED		
840	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

07-03-00

JC844 U.S. PTO 06/30/00

IN THE U.S. PATENT AND TRADEMARK OFFICE
Application Transmittal Sheet

Our Ref./Docket No.: APPT-001-2

JC858 U.S. PTO 09/609179 06/30/00

Box Patent Application
ASSISTANT COMMISSIONER FOR PATENTS
Washington, D.C. 20231

Dear Assistant Commissioner:

Transmitted herewith is the patent application of

INVENTOR(s)/APPLICANT(s)

Last Name	First Name, MI	Residence (City and State or Country)
Dietz	Russell S.	San Jose, CA
Torgerson	James F.	Andover, MN
Koppenhaver	Andrew A.	Fairfax, VA

TITLE OF THE INVENTION

PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE

CORRESPONDENCE ADDRESS AND AGENT FOR APPLICANT(S)

Dov Rosenfeld, Reg. No. 38,387
5507 College Avenue, Suite 2
Oakland, California, 94618
Telephone: (510) 547-3378; Fax: (510) 653-7992

ENCLOSED APPLICATION PARTS (check all that apply)

Included are:
- X 129 sheet(s) of specification, claims, and abstract
- X 20 sheet(s) of formal Drawing(s) with a submission letter to the Official Draftsperson
- _____ Information Disclosure Statement.
- _____ Form PTO-1449: INFORMATION DISCLOSURE CITATION IN ANAPPLICATION, together with a copy of each references included in PTO-1449.
- _____ Declaration and Power of Attorney
- _____ An assignment of the invention to Apptitude, Inc.
- _____ A letter requesting recordation of the assignment.
- _____ An assignment Cover Sheet.
- _____ Additional inventors are being named on separately numbered sheets attached hereto.
- X Return postcard.

This application has:
- _____ a small entity status. A verified statement:
  - _____ is enclosed
  - _____ was already filed.

The fee has been calculated as shown in the following page.

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: EI417961935US 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

	TOTAL CLAIMS	NO. OF EXTRA CLAIMS	RATE	EXTRA CLAIM FEE
TOTAL CLAIMS	18	0	$18	$ 0.00
INDEP. CLAIMS	1	0	$78	$ 0.00
			BASIC APPLICATION FEE:	$ 690.00
			TOTAL FEES PAYABLE:	$ 690.00

METHOD OF PAYMENT

_____ A check in the amount of _____is attached for application fee and presentation of claims.
_____ A check in the amount of $ 40.00 is attached for recordation of the Assignment.
_____ The Commissioner is hereby authorized to charge payment of the any missing filing or other fees required for this filing or credit any overpayment to Deposit Account No. 50-0292
(A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED):

Respectfully Submitted,

June 30, 2000
Date

Dov Rosenfeld , Reg. No. 38687

Correspondence Address:
Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, California, 94618
Telephone: (510) 547-3378; Fax: (510) 653-7992

Our Ref./Docket No: APPT-001-2 Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz, *et al.*	Group Art Unit: unassigned
Title: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE	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 20 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: EI417961935US 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

Our Ref./Docket No.: APPT-001-2

PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE

Inventor(s):

DIETZ, Russell S.
San Jose, CA

KOPPENHAVER, Andrew A.
Fairfax, VA

TORGERSON, James F.
Andover, MN

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 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. 09/608,237 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. 09/608,126 for RE-USING INFORMATION FROM DATA TRANSACTIONS FOR MAINTAINING STATISTICS IN NETWORK MONITORING, to inventors Dietz, et al., filed June 30, 2000, Attorney/Agent Reference Number APPT-001-3, and incorporated herein by reference.

U.S. Patent Application Serial No. 09/608,166 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. 09/608,161 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

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.

The recognizing and classifying in such a network monitor should be at all protocol layer levels in conversational flows that pass in either direction at a point in a network. Furthermore, the monitor should provide for properly analyzing each of the packets exchanged between a client and a server, maintaining information relevant to the current state of each of these conversational flows.

Related and incorporated by reference U.S. Patent application 09/166,137 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.

A network analyzer should be able to analyze many different protocols. At a base level, there are a number of standards used in digital telecommunications, including Ethernet, HDLC, ISDN, Lap B, ATM, X.25, Frame Relay, Digital Data Service, FDDI (Fiber Distributed Data Interface), T1, and others. Many of these standards employ different packet and/or frame formats. For example, data is transmitted in ATM and frame-relay systems in the form of fixed length packets (called "cells") that are 53 octets (*i.e.*, bytes) long. Several such cells may be needed to make up the information that might be included in the packet employed by some other protocol for the same payload

information—for example in a conversational flow that uses the frame-relay standard or the Ethernet protocol.

In order for a network monitor to be able to analyze different packet or frame formats, the monitor needs to be able to perform protocol specific operations on each packet with each packet carrying information conforming to different protocols and related to different applications. For example, the monitor needs to be able to parse packets of different formats into fields to understand the data encapsulated in the different fields. As the number of possible packet formats or types increases, the amount of logic required to parse these different packet formats also increases.

Prior art network monitors exist that parse individual packets and look for information at different fields to use for building a signature for identifying packets. Chiu, *et al.*, describe a method for collecting information at the session level in a computer network in United States Patent 5,101,402, titled "APPARATUS AND METHOD FOR REAL-TIME MONITORING OF NETWORK SESSIONS AND A LOCAL AREA NETWORK." In this patent, there are fixed locations specified for particular types of packets. For example, if a DECnet packet appears, the Chiu system 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 locations are examined. The system looks only at the lowest levels up to the protocol layer. There are fixed locations for each of the fields that specified the next level. 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.

It is desirable to be able to adaptively determine the locations and the information extracted from any packet for the particular type of packet. In this way, an optimal signature may be defined using a protocol-dependent and packet-content-dependent definition of what to look for and where to look for it in order to form a signature.

There thus is also a need for a network monitor that can be tailored or adapted for different protocols and for different application programs. There thus is also a need for a network monitor that can accommodate new protocols and for new application programs. There also is a need for means for specifying new protocols and new levels, including new applications. There also is a need for a mechanism to describe protocol specific operations, including, for example, what information is relevant to packets and packets

that need to be decoded, and to include specifying parsing operations and extraction operations. There also is a need for a mechanism to describe state operations to perform on packets that are at a particular state of recognition of a flow in order to further recognize the flow.

SUMMARY

One embodiment of the invention is a method of performing protocol specific operations on a packet passing through a connection point on a computer network. The packet contents conform to protocols of a layered model wherein the protocol at a particular layer level may include one or a set of child protocols defined for that level. The method includes receiving the packet and receiving a set of protocol descriptions for protocols may be used in the packet. A protocol description for a particular protocol at a particular layer level includes any child protocols of the particular protocol, and for any child protocol, where in the packet information related to the particular child protocol may be found. A protocol description also includes any protocol specific operations to be performed on the packet for the particular protocol at the particular layer level. The method includes performing the protocol specific operations on the packet specified by the set of protocol descriptions based on the base protocol of the packet and the children of the protocols used in the packet. A particular embodiment includes providing the protocol descriptions in a high-level protocol description language, and compiling to the descriptions into a data structure. The compiling may further include compressing the data structure into a compressed data structure. The protocol specific operations may include parsing and extraction operations to extract identifying information. The protocol specific operations may also include state processing operations defined for a particular state of a conversational flow of the packet.

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.

FIG. 19 shows various PDL file modules to be compiled together by the compiling process illustrated in FIG. 20 as an example, in accordance with a compiling aspect of the invention.

FIG. 20 is a flowchart of the process of compiling high-level language files according to an aspect 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.

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.

The Pattern Parse and Extraction Database Format

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

base level). Each protocol is either a parent node of some other protocol at the next later 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.

As an example of the tree structure, consider an Ethernet 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. Another child of the IP may be the UDP protocol.

A packet includes at least one header for each protocol used. The child protocol of a particular protocol used in a packet is indicated by the contents at a location within the header of the particular protocol. The contents of the packet that specify the child are in the form of a child recognition pattern.

A network analyzer preferably can analyze many different protocols. At a base level, there are a number of packet types used in digital telecommunications, including Ethernet, HDLC, ISDN, Lap B, ATM, X.25, Frame Relay, Digital Data Service, FDDI (Fiber Distributed Data Interface), and T1, among others. Many of these packet types use different packet and/or frame formats. For example, data is transmitted in ATM and frame-relay systems in the form of fixed length packets (called "cells") that are 53 octets (*i.e.*, bytes) long; several such cells may be needed to make up the information that might be included in a single packet of some other type.

Note that the term packet herein 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," "frame format" and "cell format" are generally synonymous.

The packet monitor 300 can analyze different protocols, and thus can perform different protocol specific operations on a packet wherein the protocol headers of any protocol are located at different locations depending on the parent protocol or protocols used in the packet. Thus, the packet monitor adapts to different protocols according to the contents of the packet. The locations and the information extracted from any packet are adaptively determined for the particular type of packet. For example, there is no fixed

definition of what to look for or where to look in order to form the flow signature. In some prior art systems, such as that described in United States Patent 5,101,402 to Chiu, *et al.*, there are fixed locations specified for particular types of packets. With the proliferation of protocols, 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 number of levels is variable for any protocol and is whatever number is sufficient to uniquely identify as high up the level system as we wish to go, all the way to the application level (in the OSI model).

Even the same protocol may have different variants. Ethernet packets for example, have several known variants, each having a basic format that remains substantially the same. 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 Ethernet (IEEE 803.x) packet. A monitor should be able to handle all types of Ethernet protocols. With the Ethertype protocol, the contents that indicate the child protocol is in one location, while with an IEEE type, the child protocol is specified in a different location. The child protocol is indicated by a child recognition pattern.

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. Such information is also to be specified in the parsing structures and extraction operations database 308. This includes all of the header information at this level in the form of 6 bytes of Dst MAC information 1606 and 6 bytes of Src MAC information 1610. Also specified are the source and destination address components, respectively, of the hash. These are shown as 2 byte Dst Hash 1608 from the Dst MAC address and the 2 byte Src Hash 1612 from the Src MAC address. Finally, information is included (1614) on where to the header starts for information related to the next layer level. In this case the next layer level (level 2) information starts at packet offset 12.

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.

Also shown is some of the extracted part used for the parser record and to locate the next header information. The signature part of the parser record includes extracted part 1702. Also included is the 1-byte Hash component 1710 from this information.

An offset field 1710 provides the offset to go to the next level information, i.e., to locate the start of the next layer level header. For the Ethertype packet, the start of the next layer header 14 bytes from the start of the frame.

Other packet types are arranged differently. For example, in an ATM system, each ATM packet comprises a five-octet "header" segment followed by a forty-eight octet "payload" segment. The header segment of an ATM cell contains information relating to the routing of the data contained in the payload segment. The header segment also contains traffic control information. Eight or twelve bits of the header segment contain the Virtual Path Identifier (VPI), and sixteen bits of the header segment contain the Virtual Channel Identifier (VCI). Each ATM exchange translates the abstract routing information represented by the VPI and VCI bits into the addresses of physical or logical network links and routes each ATM cell appropriately.

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 header starts at a different location (L3) depending on the parent protocol. Also included in FIG. 17B are some of the fields to be extracted for the signature, and an indication of where the next level's header would start in the packet.

Note that the information shown in FIGS. 16, 17A, and 17B would be specified to the monitor in the form of PDL files and compiled into the database 308 of pattern structures and extraction operations.

The parsing subsystem 301 performs operations on the packet header data based on information stored in the database 308. Because data related to protocols can be considered as organized in the form of a tree, it is required in the parsing subsystem to

search through data that is originally organized in the form of a tree. Since real time operation is preferable, it is required to carry out such searches rapidly.

Data structures are known for efficiently storing information organized as trees. Such storage-efficient means typically require arithmetic computations to determine pointers to the data nodes. Searching using such storage-efficient data structures may therefore be too time consuming for the present application. It is therefore desirable to store the protocol data in some form that enables rapid searches.

In accordance with another aspect of the invention, the database 308 is stored in a memory and includes a data structure used to store the protocol specific operations that are to be performed on a packet. In particular, a compressed representation is used to store information in the pattern parse and extraction database 308 used by the pattern recognition process 304 and the extraction process 306 in the parser subsystem 301. The data structure is organized for rapidly locating the child protocol related information by using a set of one or more indices to index the contents of the data structure. A data structure entry includes an indication of validity. Locating and identifying the child protocol includes indexing the data structure until a valid entry is found. Using the data structure to store the protocol information used by the pattern recognition engine (PRE) 1006 enables the parser subsystem 301 to perform rapid searches.

In one embodiment, the data structure is in the form of a three-dimensional structure. Note that this three dimensional structure in turn is typically stored in memory as a set of two-dimensional structures whereby one of the three dimensions of the 3-D structure is used as an index to a particular 2-D array. This forms a first index to the data structure.

FIG. 18A shows such a 3-D representation 1800 (which may be considered as an indexed set of 2-D representations). The three dimensions of this data structure are:

1. Type identifier [1:M]. This is the identifier that identifies a type of protocol at a particular level. For example, 01 indicates an Ethernet frame. 64 indicates IP, 16 indicates an IEEE type Ethernet packet, *etc.* Depending on how many protocols the packet parser can handle, M may be a large number; M may grow over time as the capability of analyzing more protocols is added to monitor 300. When the 3-D structure is considered a set of 2-D structures, the type ID is an index to a particular 2-D structure.

2. Size [1:64]. The size of the field of interest within the packet.

3. Location [1:512]. This is the offset location within the packet, expressed as a number of octets (bytes).

At any one of these locations there may or may not be valid data. Typically, there will not be valid data in most locations. The size of the 3-D array is M by 64 by 512, which can be large; M for example may be 10,000. This is a sparse 3-D matrix with most entries empty (i.e., invalid).

Each array entry includes a "node code" that indicates the nature of the contents. This node code has one of four values: (1) a "protocol" node code indicating to the pattern recognition process 304 that a known protocol has been recognized as the next (i.e., child) protocol; (2) a "terminal" node code indicating that there are no children for the protocol presently being searched, i.e., the node is a final node in the protocol tree; (3) a "null" (also called "flush") node code indicating that there is no valid entry.

In the preferred embodiment, the possible children and other information are loaded into the data structure by an initialization that includes compilation process 310 based on the PDL files 336 and the layering selections 338. The following information is included for any entry in the data structure that represents a protocol.

(a) A list of children (as type IDs) to search next. For example, for an Ethernet type 2, the children are Ethertype (IP, IPX, *etc,* as shown in 1712 of FIG. 17). These children are compiled into the type codes. The code for IP is 64, that for IPX is 83, *etc.*

(b) For each of the IDs in the list, a list of the child recognition patterns that need to be compared. For example, $64{:}0800_{16}$ in the list indicates that the

value to look for is 0800 (hex) for the child to be type ID 64 (which is the IP protocol). $83{:}8137_{16}$ in the list indicates that the value to look for is 8137 (hex) for the child to be type ID 83 (which is the IPX protocol), *etc.*

(c) The extraction operations to perform to build the identifying signature for the flow. The format used is (offset, length, flow_signature_value_identifier), the flow_signature_value_identifier indicating where the extracted entry goes in the signature, including what operations (AND, ORs, *etc.*) may need to be carried out. If there is also a hash key component, for instance, then information on that is included. For example, for an Ethertype packet, the 2-byte type (1706 in FIG 17) is used in the signature. Furthermore, a 1-byte hash (1708 in FIG. 17A) of the type is included. . Note furthermore, the child protocol starts at offset 14.

An additional item may be the "fold." Folding is used to reduce the storage requirements for the 3-D structure. Since each 2-D array for each protocol ID may be sparsely populated, multiple arrays may be combined into a single 2-D array as long as the individual entries do not conflict with each other. A fold number is then used to associate each element. For a given lookup, the fold number of the lookup must match the fold number entry. Folding is described in more detail below.

In the case of the Ethernet, the next protocol field may indicate a length, which tells the parser that this is a IEEE type packet, and that the next protocol is elsewhere. Normally, the next protocol field contains a value which identifies the next, i.e., child protocol.

The entry point for the parser subsystem is called the virtual base layer and contains the possible first children, i.e., the packet types. An example set of protocols written in a high level protocol description language (PDL) is included herein. The set includes PDL files, and the file describing all the possible entry points (i.e., the virtual base) is called virtual.pdl. There is only one child, 01, indicating the Ethernet, in this file. Thus, the particular example can only handle Ethernet packets. In practice, there can be multiple entry points.

In one embodiment, the packet acquisition device provides a header for every packet acquired and input into monitor 300 indicating the type of packet. This header is

used to determine the virtual base layer entry point to the parser subsystem. Thus, even at the base layer, the parser subsystem can identify the type of packet.

Initially, the search starts at the child of the virtual base, as obtained in the header supplied by the acquisition device. In the case of the example, this has ID value 01, which is the 2-D array in the overall 3-D structure for Ethernet packets.

Thus hardware implementing pattern analysis process 304 (e.g., pattern recognition engine (PRE) 1006 of FIG. 10) searches to determine the children (if any) for the 2-D array that has protocol ID 01. In the preferred embodiment that uses the 3-D data structure, the hardware PRE 1006 searches up to four lengths (i.e., sizes) simultaneously. Thus, the process 304 searches in groups of four lengths. Starting at protocol ID 01, the first two sets of 3-D locations searched are

(1, 1, 1)	(1, 1, 2)	...
(1, 2, 1)	(1, 2, 2)	
(1, 3, 1)	(1, 3, 2)	
(1, 4, 1)	(1, 4, 2)	

At each stage of a search, the analysis process 304 examines the packet and the 3-D data structure to see if there is a match (by looking at the node code). If no valid data is found, e.g., using the node code, the size is incremented (to maximum of 4) and the offset is then incremented as well.

Continuing with the example, suppose the pattern analysis process 304 finds something at 1, 2, 12. By this, we mean that the process 304 has found that for protocol ID value 01 (Ethernet) at packet offset 12, there is information in the packet having a length of 2 bytes (octets) that may relate to the next (child) protocol. The information, for example, may be about a child for this protocol expressed as a child recognition pattern. The list of possible child recognition patterns that may be in that part of the packet is obtained from the data structure.

The Ethernet packet structure comes in two flavors, the Ethertype packet and newer IEEE types, and the packet location that indicates the child is different for both. The location that for the Ethertype packet indicates the child is a "length" for the IEEE type, so a determination is made for the Ethernet packet whether the "next protocol" location contains a value or a length (this is called a "LENGTH" operation). A successful

LENGTH operation is indicated by contents less than or equal to $05DC_{16}$, then this is an IEEE type Ethernet frame. In such a case, the child recognition pattern is looked for elsewhere. Otherwise, the location contains a value that indicates the child.

Note that while this capability of the entry being a value (e.g., for a child protocol ID) or a length (indicating further analysis to determine the child protocol) is only used for Ethernet packets, in the future, other packets may end up being modified. Accordingly, this capability in the form of a macro in the PDL files still enables such future packets to be decoded.

Continuing with the example, suppose that the LENGTH operation fails. In that case, we have an Ethertype packet, and the next protocol field (containing the child recognition pattern) is 2 bytes long starting at offset 12 as shown as packet field 1702 in FIG. 17A. This will be one of the children of the Ethertype shown in table 1712 in FIG. 17A. The PRE uses the information in the data structure to check what the ID code is for the found 2-byte child recognition pattern. For example, if the child recognition pattern is 0800 (Hex), then the protocol is IP. If the child recognition pattern is 0BAD (Hex) the protocol is VIP (VINES).

Note that an alternate embodiment may keep a separate table that includes all the child recognition patterns and their corresponding protocol ID's

To follow the example, suppose the child recognition pattern at 1,2,12 is 0800_{16}, indicating IP. The ID code for the IP protocol is 64_{10}). To continue with the Ethertype example, once the parser matches one of the possible children for the protocl--in the example, the protocol type is IP with an ID of 64--then the parser continues the search for the next level. The ID is 64, the length is unknown, and offset is known to be equal or larger than 14 bytes (12 offset for type, plus 2, the length of type), so the search of the 3-D structure commences from location (64, 1) at packet offset 14. A populated node is found at (64, 2) at packet offset 14. Heading details are shown as 1750 in FIG. 17B. The possible children are shown in table 1752.

Alternatively, suppose that at (1, 2, 12) there was a length 1211_{10}. This indicates that this is an IEEE type Ethernet frame, which stores its type elsewhere. The PRE now continues its search at the same level, but for a new ID, that of an IEEE type Ethernet frame. An IEEE Ethernet packet has protocol ID 16, so the PRE continues its search of

the three-dimensional space with ID 16 starting at packet offset 14.

In our example, suppose there is a "protocol" node code found at (16, 2) at packet offset 14, and the next protocol is specified by child recognition pattern 0800_{16}. This indicates that the child is the IP protocol, which has type ID 64. Thus the search continues, starting at (64, 1) at packet offset 16.

Compression.

As noted above, the 3-D data structure is very large, and sparsely populated. For example, if 32 bytes are stored at each location, then the length is M by 64 by 512 by 32 bytes, which is M megabytes. If M = 10,000, then this is about 10 gigabytes. It is not practical to include 10 Gbyte of memory in the parser subsystem for storing the database 308. Thus a compressed form of storing the data is used in the preferred embodiment. The compression is preferably carried out by an optimizer component of the compilation process 310.

Recall that the data structure is sparse. Different embodiments may use different compression schemes that take advantage of the sparseness of the data structure. One embodiment uses a modification of multi-dimensional run length encoding.

Another embodiment uses a smaller number two-dimensional structures to store the information that otherwise would be in one large three-dimensional structure. The second scheme is used in the preferred embodiment.

FIG. 18A illustrated how the 3-D array 1800 can be considered a set of 2-D arrays, one 2-D array for each protocol (*i.e.*, each value of the protocol ID). The 2-D structures are shown as 1802-1, 1802-2, ..., 1802-M for up to M protocol ID's. One table entry is shown as 1804. Note that the gaps in table are used to illustrate that each 2-D structure table is typically large.

Consider the set of trees that represent the possible protocols. Each node represents a protocol, and a protocol may have a child or be a terminal protocol. The base (root) of the tree has all packet types as children. The other nodes form the nodes in the tree at various levels from level 1 to the final terminal nodes of the tree. Thus, one element in the base node may reference node ID 1, another element in the base node may reference node ID 2 and so on. As the tree is traversed from the root, there may be points

in the tree where the same node is referenced next. This would occur, for example, when an application protocol like Telnet can run on several transport connections like TCP or UDP. Rather than repeating the Telnet node, only one node is represented in the patterns database 308 which can have several parents. This eliminates considerable space explosion.

Each 2-D structure in FIG. 18A represents 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.

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.

Assume two 2-D arrays are being considered for folding. Call the first 2-D arrays A and the second 2-D array B. Since both 2-D arrays are partially populated, 2-D array B can be combined with 2-D arrays A if and only if none of the individual elements of these two 2-D arrays that have the same 2-D location conflict. If the result is foldable, then the valid entries of 2-D array B are combined with the valid entries of 2-D array A yielding one physical 2-D array. However, it is necessary to be able to distinguish the original 2-D array A entries from those of 2-D array B. For example, if a parent protocol of the protocol represented by 2-D array B wants to reference the protocol ID of 2-D array B, it must now reference 2-D array A instead. However, only the entries that were in the original 2-D array B are valid entries for that lookup. To accomplish this, each element in any given 2-D array is tagged with a fold number. When the original tree is created, all elements in all the 2-D arrays are initialized with a fold value of zero. Subsequently, if 2-D array B is folded into 2-D array A, all valid elements of 2-D array B are copied to the corresponding locations in 2-D array A and are given different fold numbers than any of

the elements in 2-D array A. For example, if both 2-D array A and 2-D array B were original 2-D arrays in the tree (i.e., not previously folded) then, after folding, all the 2-D array A entries would still have fold 0 and the 2-D array B entries would now all have a fold value of 1. After 2-D array B is folded into 2-D array A, the parents of 2-D array B need to be notified of the change in the 2-D array physical location of their children and the associated change in the expected fold value.

This folding process can also occur between two 2-D arrays that have already been folded, as long as none of the individual elements of the two 2-D arrays conflict for the same 2-D array location. As before, each of the valid elements in 2-D array B must have fold numbers assigned to them that are unique from those of 2-D array A. This is accomplished by adding a fixed value to all the 2-D array B fold numbers as they are merged into 2-D array A. This fixed value is one larger than the largest fold value in the original 2-D array A. It is important to note that the fold number for any given 2-D array is relative to that 2-D array only and does not span across the entire tree of 2-D arrays.

This process of folding can now be attempted between all combinations of two 2-D arrays until there are no more candidates that qualify for folding. By doing this, the total number of 2-D arrays can be significantly reduced.

Whenever a fold occurs, the 3-D structure (i.e., all 2-D arrays) must be searched for the parents of the 2-D array being folded into another array. The matching pattern which previously was mapped to a protocol ID identifying a single 2-D array must now be replaced with the 2-D array ID and the next fold number (i.e., expected fold).

Thus, in the compressed data structure, each entry valid entry includes the fold number for that entry, and additionally, the expected fold for the child.

An alternate embodiment of the data structure used in database 308 is illustrated in FIG. 18B. Thus, like the 3-D structure described above, it permits rapid searches to be performed by the pattern recognition process 304 by indexing locations in a memory rather than performing address link computations. The structure, like that of FIG. 18A, is suitable for implementation in hardware, for example, for implementation to work with the pattern recognition engine (PRE) 1006 of FIG. 10.

A table 1850, called the protocol table (PT) has an entry for each protocol known by the monitor 300, and includes some of the characteristics of each protocol, including a

description of where the field that specifies next protocol (the child recognition pattern) can be found in the header, the length of the next protocol field, flags to indicate the header length and type, and one or more slicer commands, the slicer can build the key components and hash components for the packet at this protocol at this layer level.

For any protocol, there also are one or more lookup tables (LUTs). Thus database 308 for this embodiment also includes a set of LUTs 1870. Each LUT has 256 entries indexed by one byte of the child recognition pattern that is extracted from the next protocol field in the packet. Such a protocol specification may be several bytes long, and so several of LUTs 1870 may need to be looked up for any protocol.

Each LUT's entry includes a 2-bit "node code" that indicates the nature of the contents, including its validity. This node code has one of four values: (1) a "protocol" node code indicating to the pattern recognition engine 1006 that a known protocol has been recognized; (2) an "intermediate" node code, indicating that a multi-byte protocol code has been partially recognized, thus permitting chaining a series of LUTs together before; (3) a "terminal" node code indicating that there are no children for the protocol presently being searched, i.e., the node is a final node in the protocol tree; (4) a "null" (also called "flush" and "invalid") node code indicating that there is no valid entry.

In addition to the node code, each LUT entry may include the next LUT number, the next protocol number (for looking up the protocol table 1850), the fold of the LUT entry, and the next fold to expect. Like in the embodiment implementing a compressed form of the 3-D representation, folding is used to reduce the storage requirements for the set of LUTs. Since the LUTs 1870 may be sparsely populated, multiple LUTs may be combined into a single LUT 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 LUT.

For a given lookup, the fold number of the lookup must match the fold number in the lookup table. The expected fold is obtained from the previous table lookup (the "next fold to expect" field). The present implementation uses 5-bits to describe the fold and thus allows up to 32 tables to be folded into one table.

When using the data structure of FIG. 18B, when a packet arrives at the parser, the virtual base has been pre-pended or is known. The virtual base entry tells the packet recognition engine where to find the first child recognition pattern in the packet. The

pattern recognition engine then extracts the child recognition pattern bytes from the packet and uses them as an address into the virtual base table (the first LUT). If the entry looked up in the specified next LUT by this method matches the expected next fold value specified in the virtual base entry, the lookup is deemed valid. The node code is then examined. If it is an intermediate node then the next table field obtained from the LUT lookup is used as the most significant bits of the address. The next expected fold is also extracted from the entry. The pattern recognition engine 1006 then uses the next byte from the child recognition pattern as the for the next LUT lookup.

Thus, the operation of the PRE continues until a terminal code is found. The next (initially base layer) protocol is looked up in the protocol table 1850 to provide the PRE 1006 with information on what field in the packet (in input buffer memory 1008 of parser subsystem 1000) to use for obtaining the child recognition pattern of the next protocol, including the size of the field. The child recognition pattern bytes are fetched from the input buffer memory 1008. The number of bytes making up the child recognition pattern is also now known.

The first byte of the protocol code bytes is used as the lookup in the next LUT. If a LUT lookup results in a node code indicating a protocol node or a terminal node, the Next LUT and next expected fold is set, and the "next protocol" from LUT lookup is used as an index into the protocol table 1850. This provides the instructions to the slicer 1007, and where in the packet to obtain the field for the next protocol. Thus, the PRE 1006 continues until it is done processing all the fields (i.e., the protocols), as indicated by the terminal node code reached.

Note that when a child recognition pattern is checked against a table there is always an expected fold. If the expected fold matches the fold information in the table, it is used to decide what to do next. If the fold does not match, the optimizer is finished.

Note also that an alternate embodiment may use different size LUTs, and then index a LUT by a different amount of the child recognition pattern.

The present implementation of this embodiment allows for child recognition patterns of up to four bytes. Child recognition patterns of more than 4 bytes are regarded as special cases.

In the preferred embodiment, the database is generated by the compiler process

310. The compiler process first builds a single protocol table of all the links between protocols. Links consist of the connection between parent and child protocols. Each protocol can have zero or more children. If a protocol has children, a link is created that consists of the parent protocol, the child protocol, the child recognition pattern, and the child recognition pattern size. The compiler first extracts child recognition patterns that are greater than two bytes long. Since there are only a few of these, they are handled separately. Next sub links are created for each link that has a child recognition pattern size of two.

All the links are then formed into the LUTs of 256 entries.

Optimization is then carried out. The first step in the optimization is checking all the tables against all the other tables to find out which ones can share a table. This process proceeds the same way as described above for two-dimensional arrays, but now for the sparse lookup tables.

Part of the initialization process (e.g., compiler process 310) loads a slicer instruction database with data items including of instruction, source address, destination address, and length. The PRE 1006 when it sends a slicer instruction sends this instruction as an offset into the slicer instruction database. The instruction or Op code tells the slicer what to extract from the incoming packet and where to put it in the flow signature. Writing into certain fields of the flow signature automatically generates a hash. The instruction can also tell the slicer how to determine the connection status of certain protocols.

Note that alternate embodiments may generate the pattern, parse and extraction database other than by compiling PDL files.

The compilation process

The compilation process 310 is now described in more detail. This process 310 includes creating the parsing patterns and extractions database 308 that provides the parsing subsystem 301 with the information needed to parse packets and extract identifying information, and the state processing instructions database 326 that provides the state processes that need to be performed in the state processing operation 328.

Input to the compiler includes a set of files that describe each of the protocols that

can occur. These files are in a convenient protocol description language (PDL) which is a high level language. PDL is used for specifying new protocols and new levels, including new applications. The PDL is independent of the different types of packets and protocols that may be used in the computer network. A set of PDL files is used to describe what information is relevant to packets and packets that need to be decoded. The PDL is further used to specify state analysis operations. Thus, the parser subsystem and the analyzer subsystems can adapt and be adapted to a variety of different kinds of headers, layers, and components and need to be extracted or evaluated, for example, in order to build up a unique signature.

There is one file for each packet type and each protocol. Thus there is a PDL file for Ethernet packets and there is a PDL file for frame relay packets. The PDL files are compiled to form one or more databases that enable monitor 300 to perform different protocol specific operations on a packet wherein the protocol headers of any protocol are located at different locations depending on the parent protocol or protocols used in the packet. Thus, the packet monitor adapts to different protocols according to the contents of the packet. In particular, the parser subsystem 301 is able to extract different types of data for different types of packets. For example, the monitor can know how to interpret a Ethernet packet, including decoding the header information, and also how to interpret an frame relay packet, including decoding the header information.

The set of PDL files, for example, may include a generic Ethernet packet file. There also is included a PDL file for each variation Ethernet file, for example, an IEEE Ethernet file.

The PDL file for a protocol provides the information needed by compilation process 310 to generate the database 308. That database in turn tells the parser subsystem how to parse and/or extract information, including one or more of what protocol-specific components of the packet to extract for the flow signature, how to use the components to build the flow signature, where in the packet to look for these components, where to look for any child protocols, and what child recognition patterns to look for. For some protocols, the extracted components may include source and destination addresses, and the PDL file may include the order to use these addresses to build the key. For example, Ethernet frames have end-point addresses that are useful in building a better flow signature. Thus the PDL file for an Ethernet packet includes information on how the

parsing subsystem is to extract the source and destination addresses, including where the locations and sizes of those addresses are. In a frame-relay base layer, for example, there are no specific end point addresses that help to identify the flow better, so for those type of packets, the PDL file does not include information that will cause the parser subsystem to extract the end-point addresses.

Some protocols also include information on connections. TCP is an example of such a protocol. Such protocol use connection identifiers that exist in every packet. The PDL file for such a protocol includes information about what those connection identifiers are, where they are, and what their length is. In the example of TCP, for example running over IP, these are port numbers. The PDL file also includes information about whether or not there are states that apply to connections and disconnections and what the possible children are states. So, at each of these levels, the packet monitor 300 learns more about the packet. The packet monitor 300 can identify that a particular packet is part of a particular flow using the connection identifier. Once the flow is identified, the system can determine the current state and what states to apply that deal with connections or disconnections that exist in the next layer up to these particular packets.

For the particular PDL used in the preferred embodiment, a PDL file may include none or more FIELD statement each defining a specific string of bits or bytes (i.e., a field) in the packet. A PDL file may further include none or more GROUP statements each used to tie together several defined fields. A set of such tied together fields is called a group. A PDL file may further include none or more PROTOCOL statements each defining the order of the fields and groups within the header of the protocol. A PDL file may further include none or more FLOW statements each defining a flow by describing where the address, protocol type, and port numbers are in a packet. The FLOW statement includes a description of how children flows of this protocol are determined using state operations. States associated may have state operations that may be used for managing and maintaining new states learned as more packets of a flow are analyzed.

FIG. 19 shows a set of PDL files for a layering structure for an Ethernet packet that runs TCP on top of IP. The contents of these PDL files are attached as an APPENDIX hereto. Common.pdl (1903) is a file containing the common protocol definitions, i.e., some field definitions for commonly used fields in various network protocols. Flows.pdl (1905) is a file containing general flow definitions. Virtual.pdl (1907) is a PDL file containing the definition for the VirtualBase layer used. Ethernet.pdl

(1911) is the PDL file containing the definition for the Ethernet packet. The decision on Ethertype vs. IEEE type Ethernet file is described herein. If this is Ethertype, the selection is made from the file Ethertype.pdl (1913). In an alternate embodiment, the Ethertype selection definition may be in the same Ethernet file 1911. In a typical implementation, PDL files for other Ethernet types would be included. IP.pdl (1915) is a PDL file containing the packet definitions for the Internet Protocol. TCP.pdl (1917) is the PDL file containing the packet definitions for the Transmission Control Protocol, which in this case is a transport service for the IP protocol. In addition to extracting the protocol information the TCP protocol definition file assists in the process of identification of connections for the processing of states. In a typical set of files, there also would be a file UDP.pdl for the User Datagram Protocol (UDP) definitions. RPC.pdl (1919) is a PDL file file containing the packet definitions for Remote Procedure Calls.

NFS.pdl (1921) is a PDL file containing the packet definitions for the Network File System. Other PDL files would typically be included for all the protocols that might be encountered by monitor 300.

Input to the compilation process 310 is the set of PDL files (e.g., the files of FIG 19) for all protocols of interest. Input to process 310 may also include layering information shown in FIG. 3 as datagram layer selections 338. The layer selections information describes the layering of the protocols—what protocol(s) may be on top of any particular protocols. For example, IP may run over Ethernet, and also over many other types of packets. TCP may run on top of IP. UDP also may run on top of IP. When no layering information is explicitly included, it is inherent; the PDL files include the children protocols, and this provides the layering information.

The compiling process 310 is illustrated in FIG. 20. The compiler loads the PDL source files into a scratch pad memory (step 2003) and reviews the files for the correct syntax (parse step 2005). Once completed, the compiler creates an intermediate file containing all the parse elements (step 2007). The intermediate file in a format called "Compiled Protocol Language" (CPL). CPL instructions have a fixed layer format, and include all of the patterns, extractions, and states required for each layer and for the entire tree for a layer. The CPL file includes the number of protocols and the protocol definitions. A protocol definition for each protocol can include one or more of the protocol name, the protocol ID, a header section, a group identification section, sections

for any particular layers, announcement sections, a payload section, a children section, and a states section. The CPL file is then run by the optimizer to create the final databases that will be used by monitor 300. It would be clear to those in the art that alternate implementations of the compilation process 310 may include a different form of intermediate output, or no intermediate output at all, directly generating the final database(s).

After the parse elements have been created, the compiler builds the flow signature elements (step 2009). This creates the extraction operations in CPL that are required at each level for each PDL module for the building of the flow signature (and hash key) and for links between layers (2009).

With the flow signature operations complete, the PDL compiler creates (step 2011) the operations required to extract the payload elements from each PDL module. These payload elements are used by states in other PDL modules at higher layers in the processing.

The last pass is to create the state operations required by each PDL module. The state operations are complied from the PDL files and created in CPL form for later use (2013).

The CPL file is now run through an optimizer that generates the final databases used by monitor 300.

PROTOCOL DEFINITION LANGUAGE (PDL) REFERENCE GUIDE (VERSION A0.02)

Included herein is this reference guide (the "guide") for the page description language (PDL) which, in one aspect of the invention, permits the automatic generation of the databases used by the parser and analyzer sub-systems, and also allows for including new and modified protocols and applications to the capability of the monitor.

COPYRIGHT NOTICE

1. INTRODUCTION

The inventive Protocol Definition Language (PDL) is a special purpose language used to describe network protocols and all the fields within the protocol headers.

Within this guide, protocol descriptions (PDL files) are referred to as *PDL* or *rules* when there in no risk of confusion with other types of descriptions.

PDL uses both form and organization similar to the data structure definition part of the C programming language and the PERL scripting language. Since PDL was derived from a language used to decode network packet contact, the authors have mixed the language format with the requirements of packet decoding. This results in an expressive language that is very familiar and comfortable for describing packet content and the details required representing a flow.

1.1 Summary

The PDL is a non-procedural Forth Generation language (4GL). This means is describes *what* needs to be done without describing *how* to do it. The details of *how* are hidden in the compiler and the Compiled Protocol Layout (CPL) optimization utility.

In addition, it is used to describe network flows by defining which fields are the address fields, which are the protocol type fields, etc.

Once a PDL file is written, it is compiled using the Netscope compiler (**nsc**), which produces the *MeterFlow* database (MeterFlow.db) and the Netscope database (Netscope.db). The MeterFlow database contains the flow definitions and the Netscope database contains the protocol header definitions.

These databases are used by programs like: **mfkeys**, which produces flow keys (also called flow signatures); **mfcpl**, which produces flow definitions in CPL format; **mfpkts** which produces sample packets of all known protocols; and **netscope**, which decodes Sniffer™ and tcpdump files.

1.2 Guide Conventions

The following conventions will be used throughout this guide:

Small `courier` typeface indicates C code examples or function names. Functions are written with parentheses after them [**`function()`**], variables are written just as their names [**`variables`**], and structure names are written prefixed with "**`struct`**" [**`struct packet`**].

Italics indicate a filename (for instance, *mworks/base/h/base.h*). Filenames will usually be written relative to the root directory of the distribution.

Constants are expressed in decimal, unless written "`0x...`", the C language notation for hexadecimal numbers.

Note that any contents on any line in a PDL file following two hyphen (--) are ignored by the compiler. That is, they are comments.

2. PROGRAM STRUCTURE

A *MeterFlow* PDL decodes and flow set is a non-empty sequence of statements.

There are four basic types of statements or definitions available in *MeterFlow* PDL:

FIELD,

GROUP,

PROTOCOL and

FLOW.

2.1 FIELD Definitions

The FIELD definition is used to define a specific string of bits or bytes in the packet. The FIELD definition has the following format:

```
Name FIELD
SYNTAX Type [ { Enums } ]
DISPLAY-HINT "FormatString"
LENGTH "Expression"
FLAGS FieldFlags
ENCAP FieldName [ , FieldName2 ]
LOOKUP LookupType [ Filename ]
ENCODING EncodingType
DEFAULT "value"
DESCRIPTION "Description"
```

Where only the **FIELD** and **SYNTAX** lines are required. All the other lines are attribute lines, which define special characteristics about the **FIELD**. Attribute lines are optional and may appear in any order. Each of the attribute lines are described in detail below:

2.1.1 SYNTAX Type [{ Enums }]

This attribute defines the type and, if the type is an INT, BYTESTRING, BITSTRING, or SNMPSEQUENCE type, the enumerated values for the FIELD. The currently defined types are:

INT(*numBits*)	Integer that is *numBits* bits long.
UNSIGNED INT(*numBits*)	Unsigned integer that is *numBits* bits long.
BYTESTRING(*numBytes*)	String that is *numBytes* bytes long.
BYTESTRING(*R1..R2*)	String that ranges in size from *R1* to *R2* bytes.
BITSTRING(*numBits*)	String that is *numBits* bits long.
LSTRING(*lenBytes*)	String with *lenBytes* header.
NSTRING	Null terminated string.
DNSSTRING	DNS encoded string.
SNMPOID	SNMP Object Identifier.
SNMPSEQUENCE	SNMP Sequence.
SNMPTIMETICKS	SNMP TimeTicks.
COMBO *field1 field2*	Combination pseudo field.

2.1.2 DISPLAY-HINT "FormatString"

This attribute is for specifying how the value of the FIELD is displayed. The currently supported formats are:

Numx	Print as a num byte hexidecimal number.
Numd	Print as a num byte decimal number.

Numo	Print as a num byte octal number.
Numb	Print as a num byte binary number.
Numa	Print num bytes in ASCII format.
Text	Print as ASCII text.
HexDump	Print in hexdump format.

2.1.3 LENGTH "Expression"

This attribute defines an expression for determining the FIELD's length. Expressions are arithmetic and can refer to the value of other FIELD's in the packet by adding a $ to the referenced field's name. For example, "($tcpHeaderLen *4) – 20" is a valid expression if tcpHeaderLen is another field defined for the current packet.

2.1.4 FLAGS FieldFlags

The attribute defines some special flags for a FIELD. The currently supported FieldFlags are:

SAMELAYER	Display field on the same layer as the previous field.
NOLABEL	Don't display the field name with the value.
NOSHOW	Decode the field but don't display it.
SWAPPED	The integer value is swapped.

2.1.5 ENCAP FieldName [, FieldName2]

This attribute defines how one packet is encapsulated inside another. Which packet is determined by the value of the FieldName field. If no packet is found using FieldName then FieldName2 is tried.

2.1.6 LOOKUP LookupType [Filename]

This attribute defines how to lookup the name for a particular FIELD value. The currently supported LookupTypes are:

SERVICE	Use getservbyport().
HOSTNAME	Use gethostbyaddr().
MACADDRESS	Use $METERFLOW/conf/mac2ip.cf.
FILE *file*	Use *file* to lookup value.

2.1.7 ENCODING EncodingType

This attribute defines how a FIELD is encoded. Currently, the only supported EncodingType is BER (for Basic Encoding Rules defined by ASN.1).

2.1.8 DEFAULT "value"

This attribute defines the default value to be used for this field when generating sample packets of this protocol.

2.1.9 DESCRIPTION "Description"

This attribute defines the description of the FIELD. It is used for informational purposes only.

2.2 GROUP Definitions

The GROUP definition is used to tie several related FIELDs together. The GROUP definition has the following format:

```
Name GROUP
LENGTH "Expression"
OPTIONAL "Condition"
SUMMARIZE "Condition" : "FormatString" [
"Condition" : "FormatString"... ]
DESCRIPTION "Description"
::= { Name=FieldOrGroup [ ,
Name=FieldOrGroup... ] }
```

Where only the GROUP and ::= lines are required. All the other lines are attribute lines, which define special characteristics for the GROUP. Attribute lines are optional and may appear in any order. Each attribute line is described in detail below:

2.2.1 LENGTH "Expression"

This attribute defines an expression for determining the GROUP's length. Expressions are

arithmetic and can refer to the value of other FIELD's in the packet by adding a $ to the referenced field's name. For example, "($tcpHeaderLen *4) – 20" is a valid expression if tcpHeaderLen is another field defined for the current packet.

2.2.2 OPTIONAL "Condition"

This attribute defines a condition for determining whether a GROUP is present or not. Valid conditions are defined in the Conditions section below.

2.2.3 SUMMARIZE "Condition" : "FormatString" ["Condition" : "FormatString"...]

This attribute defines how a GROUP will be displayed in Detail mode. A different format (FormatString) can be specified for each condition (Condition). Valid conditions are defined in the Conditions section below. Any FIELD's value can be referenced within the FormatString by proceeding the FIELD's name with a $. In addition to FIELD names there are several other special $ keywords:

$LAYER	Displays the current protocol layer.
$GROUP	Displays the entire GROUP as a table.
$LABEL	Displays the GROUP label.
$*field*	Displays the *field* value (use enumerated name if available).
$:*field*	Displays the *field* value (in raw format).

2.2.4 DESCRIPTION "Description"

This attribute defines the description of the GROUP. It is used for informational purposes only.

2.2.5 ::= { Name=FieldOrGroup [, Name=FieldOrGroup...] }

This defines the order of the fields and subgroups within the GROUP.

2.3 PROTOCOL Definitions

The PROTOCOL definition is used to define the order of the FIELDs and GROUPs within the protocol header. The PROTOCOL definition has the following format:

```
Name PROTOCOL
SUMMARIZE "Condition" : "FormatString" [
"Condition" : "FormatString"... ]
DESCRIPTION "Description"
REFERENCE "Reference"
::= { Name=FieldOrGroup [ ,
Name=FieldOrGroup... ] }
```

Where only the PROTOCOL and ::= lines are required. All the other lines are attribute lines, which define special characteristics for the PROTOCOL. Attribute lines are optional and may appear in any order. Each attribute line is described in detail below:

2.3.1 SUMMARIZE "Condition" : "FormatString" ["Condition" : "FormatString"...]

This attribute defines how a PROTOCOL will be displayed in Summary mode. A different format (FormatString) can be specified for each condition (Condition). Valid conditions are defined in the Conditions section below. Any FIELD's value can be referenced within the FormatString by proceeding the FIELD's name with a $. In addition to FIELD names there are several other special $ keywords:

$LAYER	Displays the current protocol layer.
$VARBIND	Displays the entire SNMP VarBind list.
$*field*	Displays the *field* value (use enumerated name if available).
$:*field*	Displays the *field* value (in raw format).
$#*field*	Counts all occurrences of *field.*
$**field*	Lists all occurrences of *field.*

2.3.2 DESCRIPTION "Description"

This attribute defines the description of the PROTOCOL. It is used for informational purposes only.

2.3.3 REFERENCE "Reference"

This attribute defines the reference material used to determine the protocol format. It is used for informational purposes only.

2.3.4 ::= { Name=FieldOrGroup [, Name=FieldOrGroup...] }

This defines the order of the FIELDs and GROUPs within the PROTOCOL.

2.4 FLOW Definitions

The FLOW definition is used to define a network flow by describing where the address, protocol type, and port numbers are in a packet. The FLOW definition has the following format:

```
Name FLOW
HEADER { Option [, Option...] }
DLC-LAYER { Option [, Option...] }
NET-LAYER { Option [, Option...] }
CONNECTION { Option [, Option...] }
PAYLOAD { Option [, Option...] }
CHILDREN { Option [, Option...] }
STATE-BASED
STATES "Definitions"
```

Where only the FLOW line is required. All the other lines are attribute lines, which define special characteristics for the FLOW. Attribute lines are optional and may appear in any order. However, at least one attribute line must be present. Each attribute line is described in detail below:

2.4.1 HEADER { Option [, Option...] }

This attribute is used to describe the length of the protocol header. The currently supported Options are:

LENGTH=*number*	Header is a fixed length of size *number*.
LENGTH=*field*	Header is variable length determined by value of *field*.
IN-WORDS	The units of the header length are in 32-bit words rather than bytes.

2.4.2 DLC-LAYER { Option [, Option...] }

If the protocol is a data link layer protocol, this attribute describes it. The currently supported Options are:

DESTINATION=*field*	Indicates which *field* is the DLC destination address.
SOURCE=*field*	Indicates which *field* is the DLC source address.

PROTOCOL	Indicates this is a data link layer protocol.
TUNNELING	Indicates this is a tunneling protocol.

2.4.3 NET-LAYER { Option [, Option...] }

If the protocol is a network layer protocol, then this attribute describes it. The currently supported Options are:

DESTINATION=*field*	Indicates which *field* is the network destination address.
SOURCE=*field*	Indicates which *field* is the network source address.
TUNNELING	Indicates this is a tunneling protocol.
FRAGMENTATION=*type*	Indicates this protocol supports fragmentation. There are currently two fragmentation types: IPV4 and IPV6.

2.4.4 CONNECTION { Option [, Option...] }

If the protocol is a connection-oriented protocol, then this attribute describes how connections are established and torn down. The currently supported Options are:

IDENTIFIER=*field*	Indicates the connection identifier *field*.
CONNECT-START="*flag*"	Indicates when a connection is being initiated.
CONNECT-COMPLETE="*flag*"	Indicates when a connection has been established.
DISCONNECT-START="*flag*"	Indicates when a connection is being torn down.
DISCONNECT-COMPLETE="*flag*"	Indicates when a connection has been torn down.
INHERITED	Indicates this is a connection-oriented protocol but the parent protocol is where the connection is established.

2.4.5 PAYLOAD { Option [, Option...] }

This attribute describes how much of the payload from a packet of this type should be

stored for later use during analysis. The currently supported Options are:

INCLUDE-HEADER	Indicates that the protocol header should be included.
LENGTH=*number*	Indicates how many bytes of the payload should be stored.
DATA=*field*	Indicates which *field* contains the payload.

2.4.6 CHILDREN { Option [, Option...] }

This attribute describes how children protocols are determined. The currently supported Options are:

DESTINATION=*field*	Indicates which *field* is the destination port.
SOURCE=*field*	Indicates which *field* is the source port.
LLCCHECK=*flow*	Indicates that if the DESTINATION field is less than 0x05DC then use *flow* instead of the current flow definition.

2.4.7 STATE-BASED

This attribute indicates that the flow is a state-based flow.

2.4.8 STATES "Definitions"

This attribute describes how children flows of this protocol are determined using states. See the State Definitions section below for how these states are defined.

2.5 CONDITIONS

Conditions are used with the OPTIONAL and SUMMARIZE attributes and may consist of the following:

Value1 == Value2	Value1 equals Value2. Works with string values.
Value1 != Value2	Value1 does not equal Value2. Works with string values.
Value1 <= Value2	Value1 is less than or equal to Value2.
Value1 >= Value2	Value1 is greater than or equal to Value2.

Value1 < Value2	Value1 is less than Value2.
Value1 > Value2	Value1 is greater than Value2.
Field m/regex/	Field matches the regular expression regex.

Where *Value1* and *Value2* can be either FIELD references (field names preceded by a $) or constant values. Note that compound conditional statements (using AND and OR) are not currently supported.

2.6 STATE DEFINITIONS

Many applications running over data networks utilize complex methods of classifying traffic through the use of multiple states. State definitions are used for managing and maintaining learned states from traffic derived from the network.

The basic format of a state definition is:

```
StateName:  Operand Parameters [ Operand Parameters…]
```

The various states of a particular flow are described using the following operands:

2.6.1 CHECKCONNECT, *operand*

Checks for connection. Once connected executes *operand.*

2.6.2 GOTO *state*

Goes to *state*, using the current packet.

2.6.3 NEXT *state*

Goes to *state*, using the next packet.

2.6.4 DEFAULT *operand*

Executes *operand* when all other operands fail.

2.6.5 CHILD *protocol*

Jump to child *protocol* and perform state-based processing (if any) in the child.

2.6.6 WAIT *numPackets, operand1, operand2*

Waits the specified number of packets. Executes *operand1* when the specified number of packets have been received. Executes *operand2* when a packet is received but it is less

than the number of specified packets.

2.6.7 MATCH *'string' weight offset LF-offset range LF-range, operand*

Searches for a *string* in the packet, executes *operand* if found.

2.6.8 CONSTANT *number offset range, operand*

Checks for a constant in a packet, executes *operand* if found.

2.6.9 EXTRACTIP *offset destination, operand*

Extracts an IP address from the packet and then executes *operand.*

2.6.10 EXTRACTPORT *offset destination, operand*

Extracts a port number from the packet and then executes *operand.*

2.6.11 CREATEREDIRECTEDFLOW, *operand*

Creates a redirected flow and then executes *operand.*

3. EXAMPLE PDL RULES

The following section contains several examples of PDL Rule files.

3.1 Ethernet

The following is an example of the PDL for Ethernet:

```
MacAddress     FIELD
               SYNTAX        BYTESTRING(6)
               DISPLAY-HINT  "1x:"
               LOOKUP        MACADDRESS
               DESCRIPTION
                      "MAC layer physical address"

etherType      FIELD
               SYNTAX        INT(16)
               DISPLAY-HINT  "1x:"
               LOOKUP        FILE "EtherType.cf"
               DESCRIPTION
                      "Ethernet type field"

etherData      FIELD
               SYNTAX        BYTESTRING(46..1500)
               ENCAP                etherType
               DISPLAY-HINT  "HexDump"
               DESCRIPTION
                      "Ethernet data"

ethernet       PROTOCOL
               DESCRIPTION
                   "Protocol format for an Ethernet frame"
               REFERENCE     "RFC 894"
::= { MacDest=macAddress, MacSrc=macAddress, EtherType=etherType,
       Data=etherData }

ethernet       FLOW
               HEADER { LENGTH=14 }
               DLC-LAYER {
                   SOURCE=MacSrc,
                   DESTINATION=MacDest,
                   TUNNELING,
                   PROTOCOL
               }
               CHILDREN { DESTINATION=EtherType, LLC-CHECK=llc }
```

3.2 IP Version 4

Here is an example of the PDL for the IP protocol:

```
ipAddress     FIELD
              SYNTAX        BYTESTRING(4)
              DISPLAY-HINT  "1d."
              LOOKUP        HOSTNAME
              DESCRIPTION
                  "IP address"

ipVersion     FIELD
              SYNTAX        INT(4)
              DEFAULT       "4"

ipHeaderLength      FIELD
                    SYNTAX INT(4)

ipTypeOfService     FIELD
                    SYNTAX        BITSTRING(8) { minCost(1),
                                  maxReliability(2), maxThruput(3), minDelay(4) }

ipLength      FIELD
              SYNTAX UNSIGNED INT(16)

ipFlags       FIELD
              SYNTAX        BITSTRING(3) { moreFrags(0), dontFrag(1) }

IpFragmentOffset    FIELD
                    SYNTAX        INT(13)

ipProtocol    FIELD
              SYNTAX INT(8)
              LOOKUP FILE "IpProtocol.cf"

ipData FIELD
              SYNTAX        BYTESTRING(0..1500)
              ENCAP         ipProtocol
              DISPLAY-HINT  "HexDump"

ip      PROTOCOL
        SUMMARIZE
        "$FragmentOffset != 0":
              "IPFragment ID=$Identification Offset=$FragmentOffset"
        "Default" :
              "IP Protocol=$Protocol"
        DESCRIPTION
            "Protocol format for the Internet Protocol"
        REFERENCE     "RFC 791"
::= { Version=ipVersion, HeaderLength=ipHeaderLength,
        TypeOfService=ipTypeOfService, Length=ipLength,
        Identification=UInt16, IpFlags=ipFlags,
        FragmentOffset=ipFragmentOffset, TimeToLive=Int8,
        Protocol=ipProtocol, Checksum=ByteStr2,
        IpSrc=ipAddress, IpDest=ipAddress, Options=ipOptions,
        Fragment=ipFragment, Data=ipData }

ip      FLOW
        HEADER { LENGTH=HeaderLength, IN-WORDS }
        NET-LAYER {
            SOURCE=IpSrc,
            DESTINATION=IpDest,
            FRAGMENTATION=IPV4,
            TUNNELING
        }
```

```
        CHILDREN { DESTINATION=Protocol }

    ipFragData    FIELD
                  SYNTAX        BYTESTRING(1..1500)
                  LENGTH        "ipLength - ipHeaderLength * 4"
                  DISPLAY-HINT  "HexDump"

    ipFragment    GROUP
                  OPTIONAL      "$FragmentOffset != 0"
    ::= { Data=ipFragData }

    ipOptionCode  FIELD
                  SYNTAX INT(8) { ipRR(0x07), ipTimestamp(0x44),
                                  ipLSRR(0x83), ipSSRR(0x89) }
                  DESCRIPTION
                      "IP option code"

    ipOptionLength      FIELD
                        SYNTAX UNSIGNED INT(8)
                        DESCRIPTION
                            "Length of IP option"

    ipOptionData  FIELD
                        SYNTAX        BYTESTRING(0..1500)
                        ENCAP                 ipOptionCode
                        DISPLAY-HINT  "HexDump"

    ipOptions     GROUP
                  LENGTH        "(ipHeaderLength * 4) - 20"
    ::= { Code=ipOptionCode, Length=ipOptionLength, Pointer=UInt8,
          Data=ipOptionData }
```

3.3 TCP

Here is an example of the PDL for the TCP protocol:

```
tcpPort FIELD
       SYNTAX UNSIGNED INT(16)
       LOOKUP FILE "TcpPort.cf"

tcpHeaderLen FIELD
       SYNTAX INT(4)

tcpFlags FIELD
       SYNTAX BITSTRING(12) { fin(0), syn(1), rst(2), psh(3),
                              ack(4), urg(5) }

tcpData FIELD
       SYNTAX BYTESTRING(0..1564)
       LENGTH "($ipLength-($ipHeaderLength*4))-($tcpHeaderLen*4)"
       ENCAP         tcpPort
       DISPLAY-HINT  "HexDump"

tcp    PROTOCOL
       SUMMARIZE
           "Default" :
              "TCP ACK=$Ack WIN=$WindowSize"
       DESCRIPTION
           "Protocol format for the Transmission Control Protocol"
       REFERENCE     "RFC 793"
::= { SrcPort=tcpPort, DestPort=tcpPort, SequenceNum=UInt32,
       Ack=UInt32, HeaderLength=tcpHeaderLen, TcpFlags=tcpFlags,
       WindowSize=UInt16, Checksum=ByteStr2,
       UrgentPointer=UInt16, Options=tcpOptions, Data=tcpData }

tcp    FLOW
       HEADER { LENGTH=HeaderLength, IN-WORDS }
       CONNECTION {
            IDENTIFIER=SequenceNum,
            CONNECT-START="TcpFlags:1",
            CONNECT-COMPLETE="TcpFlags:4",
            DISCONNECT-START="TcpFlags:0",
            DISCONNECT-COMPLETE="TcpFlags:4"
        }
       PAYLOAD { INCLUDE-HEADER }
       CHILDREN { DESTINATION=DestPort, SOURCE=SrcPort }

tcpOptionKind FIELD
             SYNTAX UNSIGNED INT(8) { tcpOptEnd(0), tcpNop(1),
tcpMSS(2), tcpWscale(3), tcpTimestamp(4) }
             DESCRIPTION
                 "Type of TCP option"

tcpOptionData FIELD
             SYNTAX        BYTESTRING(0..1500)
             ENCAP         tcpOptionKind
             FLAGS         SAMELAYER
             DISPLAY-HINT  "HexDump"

tcpOptions   GROUP
             LENGTH        "($tcpHeaderLen * 4) - 20"
::= { Option=tcpOptionKind, OptionLength=UInt8,
       OptionData=tcpOptionData }

tcpMSS PROTOCOL
::= { MaxSegmentSize=UInt16 }
```

3.4 HTTP (with State)

Here is an example of the PDL for the HTTP protocol:

```
httpData FIELD
   SYNTAX  BYTESTRING(1..1500)
   LENGTH     "($ipLength - ($ipHeaderLength * 4)) - ($tcpHeaderLen * 4)"
   DISPLAY-HINT      "Text"
   FLAGS             NOLABEL

http     PROTOCOL
         SUMMARIZE
             "$httpData m/^GET|^HTTP|^HEAD|^POST/" :
                 "HTTP $httpData"
             "$httpData m/^[Dd]ate|^[Ss]erver|^[Ll]ast-[Mm]odified/" :
                 "HTTP $httpData"
             "$httpData m/^[Cc]ontent-/" :
                 "HTTP $httpData"
             "$httpData m/^<HTML>/" :
                 "HTTP [HTML document]"
             "$httpData m/^GIF/" :
                 "HTTP [GIF image]"
             "Default" :
                 "HTTP [Data]"
         DESCRIPTION
             "Protocol format for HTTP."
::= { Data=httpData }

http   FLOW
HEADER { LENGTH=0 }
CONNECTION { INHERITED }
PAYLOAD { INCLUDE-HEADER, DATA=Data, LENGTH=256 }
STATES
            "S0: CHECKCONNECT, GOTO S1
                 DEFAULT NEXT S0

             S1: WAIT 2, GOTO S2, NEXT S1
                 DEFAULT NEXT S0

             S2: MATCH
                   '\n\r\n'          900 0 0 255 0, NEXT S3
                   '\n\n'            900 0 0 255 0, NEXT S3
                   'POST /tds?'       50 0 0 127 1, CHILD sybaseWebsql
                   '.hts HTTP/1.0'    50 4 0 127 1, CHILD sybaseJdbc
                   'jdbc:sybase:Tds'  50 4 0 127 1, CHILD sybaseTds
                   'PCN-The Poin'    500 4 1 255 0, CHILD pointcast
                   't: BW-C-'        100 4 1 255 0, CHILD backweb
                   DEFAULT NEXT S3

             S3: MATCH
                   '\n\r\n'           50 0 0   0 0, NEXT S3
                   '\n\n'             50 0 0   0 0, NEXT S3
                   'Content-Type:'   800 0 0 255 0, CHILD mime
                   'PCN-The Poin'    500 4 1 255 0, CHILD pointcast
                   't: BW-C-'        100 4 1 255 0, CHILD backweb
                   DEFAULT NEXT S0"

sybaseWebsql   FLOW
               STATE-BASED

sybaseJdbc     FLOW
               STATE-BASED

sybaseTds      FLOW
               STATE-BASED
```

```
pointcast       FLOW
                STATE-BASED

backweb         FLOW
                STATE-BASED

mime            FLOW
                STATE-BASED
                STATES
                  "S0:    MATCH
'application' 900 0 0    1 0, CHILD mimeApplication
'audio'       900 0 0    1 0, CHILD mimeAudio
'image'        50 0 0    1 0, CHILD mimeImage
'text'         50 0 0    1 0, CHILD mimeText
'video'        50 0 0    1 0, CHILD mimeVideo
'x-world'     500 4 1 255 0, CHILD mimeXworld
DEFAULT GOTO S0"

mimeApplication FLOW
                STATE-BASED

mimeAudio   FLOW
            STATE-BASED
            STATES
             "S0: MATCH
                  'basic'                100 0 0 1 0, CHILD pdBasicAudio
                  'midi'                 100 0 0 1 0, CHILD pdMidi
                  'mpeg'                 100 0 0 1 0, CHILD pdMpeg2Audio
                  'vnd.rn-realaudio'     100 0 0 1 0, CHILD pdRealAudio
                  'wav'                  100 0 0 1 0, CHILD pdWav
                  'x-aiff'               100 0 0 1 0, CHILD pdAiff
                  'x-midi'               100 0 0 1 0, CHILD pdMidi
                  'x-mpeg'               100 0 0 1 0, CHILD pdMpeg2Audio
                  'x-mpgurl'             100 0 0 1 0, CHILD pdMpeg3Audio
                  'x-pn-realaudio'       100 0 0 1 0, CHILD pdRealAudio
                  'x-wav'                100 0 0 1 0, CHILD pdWav
                  DEFAULT GOTO S0"

mimeImage   FLOW
            STATE-BASED

mimeText    FLOW
            STATE-BASED

mimeVideo   FLOW
            STATE-BASED

mimeXworld  FLOW
            STATE-BASED

pdBasicAudio FLOW
             STATE-BASED

pdMidi       FLOW
             STATE-BASED

pdMpeg2Audio FLOW
             STATE-BASED

pdMpeg3Audio FLOW
             STATE-BASED

pdRealAudio  FLOW
             STATE-BASED

pdWav        FLOW
```

STATE-BASED

pdAiff FLOW
STATE-BASED

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.

APPENDIX: SOME PDL FILES.

The following pages include some PDL files as examples. Included herein are the PDL contents of the following files. A reference to PDL is also included herein. Note that any contents on any line following two hyphen (--) are ignored by the compiler. That is, they are comments.

common.pdl;

flows.pdl;

virtual.pdl;

ethernet.pdl;

IEEE8032.pdl and IEEE8033.pdl (ethertype files);

IP.pdl;

TCP.pdl and UDP.pdl;

RPC.pdl;

NFS.pdl; and

HTTP.pdl.

```
--------------------------------------------------------------------------
--
--  Common.pdl - Common protocol definitions
--
--  Description:
--     This file contains some field definitions for commonly used fields
--     in various network protocols.
--
--  Copyright:
--     Copyright (c) 1996-1999 Apptitude, Inc.
--       (formerly Technically Elite, Inc.)
--     All rights reserved.
--
--  RCS:
--     $Id: Common.pdl,v 1.7 1999/04/13 15:47:56 skip Exp $
--
--------------------------------------------------------------------------
Int4    FIELD
        SYNTAX INT(4)

Int8    FIELD
        SYNTAX INT(8)

Int16   FIELD
        SYNTAX INT(16)

Int24   FIELD
        SYNTAX INT(24)

Int32   FIELD
        SYNTAX INT(32)

Int64   FIELD
        SYNTAX INT(64)

UInt8   FIELD
        SYNTAX UNSIGNED INT(8)

UInt16 FIELD
        SYNTAX UNSIGNED INT(16)

UInt24 FIELD
        SYNTAX UNSIGNED INT(24)

UInt32 FIELD
        SYNTAX UNSIGNED INT(32)

UInt64 FIELD
        SYNTAX UNSIGNED INT(64)

SInt16 FIELD
        SYNTAX INT(16)
        FLAGS  SWAPPED

SUInt16       FIELD
        SYNTAX UNSIGNED INT(16)
        FLAGS  SWAPPED

SInt32 FIELD
        SYNTAX INT(32)
        FLAGS  SWAPPED

ByteStr1      FIELD
              SYNTAX BYTESTRING(1)

ByteStr2      FIELD
              SYNTAX BYTESTRING(2)
```

```
ByteStr4        FIELD
                SYNTAX BYTESTRING(4)

Pad1            FIELD
                SYNTAX BYTESTRING(1)
                FLAGS  NOSHOW

Pad2            FIELD
                SYNTAX BYTESTRING(2)
                FLAGS  NOSHOW

Pad3            FIELD
                SYNTAX BYTESTRING(3)
                FLAGS  NOSHOW

Pad4            FIELD
                SYNTAX BYTESTRING(4)
                FLAGS  NOSHOW

Pad5            FIELD
                SYNTAX BYTESTRING(5)
                FLAGS  NOSHOW

macAddress      FIELD
        SYNTAX          BYTESTRING(6)
        DISPLAY-HINT    "1x:"
        LOOKUP          MACADDRESS
        DESCRIPTION
            "MAC layer physical address"

ipAddress       FIELD
        SYNTAX          BYTESTRING(4)
        DISPLAY-HINT    "1d."
        LOOKUP          HOSTNAME
        DESCRIPTION
            "IP address"

ipv6Address     FIELD
        SYNTAX          BYTESTRING(16)
        DISPLAY-HINT    "1d."
        DESCRIPTION
            "IPV6 address"
```

```
--------------------------------------------------------------------------------
--
--  Flows.pdl - General FLOW definitions
--
--  Description:
--     This file contains general flow definitions.
--
--  Copyright:
--     Copyright (c) 1998-1999 Apptitude, Inc.
--       (formerly Technically Elite, Inc.)
--     All rights reserved.
--
--  RCS:
--     $Id: Flows.pdl,v 1.12 1999/04/13 15:47:57 skip Exp $
--
--------------------------------------------------------------------------------

chaosnet  FLOW

spanningTree FLOW

sna       FLOW

oracleTNS FLOW
          PAYLOAD { INCLUDE-HEADER, LENGTH=256 }

ciscoOUI  FLOW

------------------------------------------------------------
-- IP Protocols
------------------------------------------------------------

igmp      FLOW

GGP       FLOW

ST        FLOW

UCL       FLOW

egp       FLOW

igp       FLOW

BBN-RCC-MON    FLOW

NVP2     FLOW

PUP      FLOW

ARGUS    FLOW

EMCON    FLOW

XNET     FLOW

MUX      FLOW

DCN-MEAS FLOW

HMP      FLOW

PRM      FLOW

TRUNK1   FLOW

TRUNK2   FLOW
```

```
LEAF1   FLOW
LEAF2   FLOW
RDP     FLOW
IRTP    FLOW
ISO-TP4         FLOW
NETBLT  FLOW
MFE-NSP         FLOW
MERIT-INP       FLOW
SEP     FLOW
PC3     FLOW
IDPR    FLOW
XTP     FLOW
DDP     FLOW
IDPR-CMTP       FLOW
TPPlus  FLOW
IL      FLOW
SIP     FLOW
SDRP    FLOW
SIP-SR  FLOW
SIP-FRAG FLOW
IDRP    FLOW
RSVP    FLOW
MHRP    FLOW
BNA     FLOW
SIPP-ESP FLOW
SIPP-AH         FLOW
INLSP   FLOW
SWIPE   FLOW
NHRP    FLOW
CFTP    FLOW
SAT-EXPAK       FLOW
KRYPTOLAN       FLOW
RVD     FLOW
```

```
IPPC    FLOW
SAT-MON         FLOW
VISA    FLOW
IPCV    FLOW
CPNX    FLOW
CPHB    FLOW
WSN     FLOW
PVP     FLOW
BR-SAT-MON      FLOW
SUN-ND  FLOW
WB-MON  FLOW
WB-EXPAK FLOW
ISO-IP  FLOW
VMTP    FLOW
SECURE-VMTP     FLOW
TTP     FLOW
NSFNET-IGP      FLOW
DGP     FLOW
TCF     FLOW
IGRP    FLOW
OSPFIGP         FLOW
Sprite-RPC      FLOW
LARP    FLOW
MTP     FLOW
AX25    FLOW
IPIP    FLOW
MICP    FLOW
SCC-SP  FLOW
ETHERIP         FLOW
encap   FLOW
GMTP    FLOW

------------------------------------------------------------
-- UDP Protocols
------------------------------------------------------------
compressnet   FLOW
```

```
rje     FLOW

echo    FLOW

discard         FLOW

systat FLOW

daytime         FLOW

qotd    FLOW

msp     FLOW

chargen         FLOW

biff    FLOW

who     FLOW

syslog FLOW

loadav FLOW

notify FLOW

acmaint_dbd    FLOW

acmaint_transd       FLOW

puparp FLOW

applix FLOW

ock     FLOW


------------------------------------------------------------
-- TCP Protocols
------------------------------------------------------------
tcpmux FLOW

telnet FLOW
        CONNECTION { INHERITED }

privMail       FLOW

nsw-fe FLOW

msg-icp        FLOW

msg-auth       FLOW

dsp     FLOW

privPrint      FLOW

time    FLOW

rap     FLOW

rlp     FLOW

graphics       FLOW

nameserver     FLOW
```

```
nicname         FLOW
mpm-flags       FLOW
mpm     FLOW
mpm-snd         FLOW
ni-ftp FLOW
auditd FLOW
finger FLOW
re-mail-ck      FLOW
la-maint        FLOW
xns-time        FLOW
xns-ch FLOW
isi-gl FLOW
xns-auth        FLOW
privTerm        FLOW
xns-mail        FLOW
privFile        FLOW
ni-mail         FLOW
acas    FLOW
covia   FLOW
tacacs-ds       FLOW
sqlnet FLOW
gopher FLOW
netrjs-1        FLOW
netrjs-2        FLOW
netrjs-3        FLOW
netrjs-4        FLOW
privDial        FLOW
deos    FLOW
privRJE         FLOW
vettcp FLOW
hosts2-ns       FLOW
xfer    FLOW
ctf     FLOW
```

```
mit-ml-dev    FLOW
mfcobol       FLOW
kerberos      FLOW
su-mit-tg     FLOW
dnsix  FLOW
mit-dov       FLOW
npp    FLOW
dcp    FLOW
objcall       FLOW
supdup FLOW
dixie  FLOW
swift-rvf     FLOW
tacnews       FLOW
metagram      FLOW
newacct       FLOW
hostname      FLOW
iso-tsap      FLOW
gppitnp       FLOW
csnet-ns      FLOW
threeCom-tsmux       FLOW
rtelnet       FLOW
snagas FLOW
mcidas FLOW
auth   FLOW
audionews     FLOW
sftp   FLOW
ansanotify    FLOW
uucp-path     FLOW
sqlserv       FLOW
cfdptkt       FLOW
erpc   FLOW
smakynet      FLOW
ntp    FLOW
ansatrader    FLOW
```

```
locus-map       FLOW
unitary         FLOW
locus-con       FLOW
gss-xlicen      FLOW
pwdgen FLOW
cisco-fna       FLOW
cisco-tna       FLOW
cisco-sys       FLOW
statsrv         FLOW
ingres-net      FLOW
loc-srv         FLOW
profile         FLOW
emfis-data      FLOW
emfis-cntl      FLOW
bl-idm FLOW
imap2  FLOW
news   FLOW
uaac   FLOW
iso-tp0         FLOW
iso-ip FLOW
cronus FLOW
aed-512         FLOW
sql-net         FLOW
hems   FLOW
bftp   FLOW
sgmp   FLOW
netsc-prod      FLOW
netsc-dev       FLOW
sqlsrv FLOW
knet-cmp        FLOW
pcmail-srv      FLOW
nss-routing     FLOW
sgmp-traps      FLOW
```

```
cmip-man        FLOW
cmip-agent      FLOW
xns-courier     FLOW
s-net   FLOW
namp    FLOW
rsvd    FLOW
send    FLOW
print-srv       FLOW
multiplex       FLOW
cl-1    FLOW
xyplex-mux      FLOW
mailq   FLOW
vmnet   FLOW
genrad-mux      FLOW
xdmcp   FLOW
nextstep        FLOW
bgp     FLOW
ris     FLOW
unify   FLOW
audit   FLOW
ocbinder        FLOW
ocserver        FLOW
remote-kis      FLOW
kis     FLOW
aci     FLOW
mumps   FLOW
qft     FLOW
gacp    FLOW
prospero        FLOW
osu-nms         FLOW
srmp    FLOW
irc     FLOW
dn6-nlm-aud     FLOW
dn6-smm-red     FLOW
```

```
dls     FLOW
dls-mon         FLOW
smux    FLOW
src     FLOW
at-rtmp         FLOW
at-nbp FLOW
at-3    FLOW
at-echo         FLOW
at-5    FLOW
at-zis FLOW
at-7    FLOW
at-8    FLOW
tam     FLOW
z39-50 FLOW
anet    FLOW
vmpwscs         FLOW
softpc FLOW
atls    FLOW
dbase   FLOW
mpp     FLOW
uarps   FLOW
imap3   FLOW
fln-spx         FLOW
rsh-spx         FLOW
cdc     FLOW
sur-meas        FLOW
link    FLOW
dsp3270         FLOW
pdap    FLOW
pawserv         FLOW
zserv   FLOW
fatserv         FLOW
csi-sgwp        FLOW
```

```
clearcase     FLOW
ulistserv     FLOW
legent-1      FLOW
legent-2      FLOW
hassle FLOW
nip     FLOW
tnETOS FLOW
dsETOS FLOW
is99c  FLOW
is99s  FLOW
hp-collector  FLOW
hp-managed-node       FLOW
hp-alarm-mgr  FLOW
arns    FLOW
ibm-app        FLOW
asa     FLOW
aurp    FLOW
unidata-ldm   FLOW
ldap    FLOW
uis     FLOW
synotics-relay        FLOW
synotics-broker       FLOW
dis     FLOW
embl-ndt        FLOW
netcp  FLOW
netware-ip      FLOW
mptn    FLOW
kryptolan       FLOW
work-sol        FLOW
ups     FLOW
genie   FLOW
decap   FLOW
nced    FLOW
ncld    FLOW
```

```
imsp    FLOW
timbuktu        FLOW
prm-sm FLOW
prm-nm FLOW
decladebug      FLOW
rmt     FLOW
synoptics-trap          FLOW
smsp    FLOW
infoseek        FLOW
bnet    FLOW
silverplatter FLOW
onmux   FLOW
hyper-g         FLOW
ariell FLOW
smpte   FLOW
ariel2 FLOW
ariel3 FLOW
opc-job-start FLOW
opc-job-track FLOW
icad-el         FLOW
smartsdp        FLOW
svrloc FLOW
ocs_cmu         FLOW
ocs_amu         FLOW
utmpsd FLOW
utmpcd FLOW
iasd    FLOW
nnsp    FLOW
mobileip-agent          FLOW
mobilip-mn      FLOW
dna-cml         FLOW
comscm FLOW
dsfgw   FLOW
```

```
dasp    FLOW
sgcp    FLOW
decvms-sysmgt FLOW
cvc_hostd     FLOW
https   FLOW
        CONNECTION { INHERITED }
snpp    FLOW
microsoft-ds  FLOW
ddm-rdb       FLOW
ddm-dfm       FLOW
ddm-byte      FLOW
as-servermap  FLOW
tserver       FLOW
exec    FLOW
        CONNECTION { INHERITED }
login   FLOW
        CONNECTION { INHERITED }
cmd     FLOW
        CONNECTION { INHERITED }
printer       FLOW
        CONNECTION { INHERITED }
talk    FLOW
        CONNECTION { INHERITED }
ntalk   FLOW
        CONNECTION { INHERITED }
utime   FLOW
efs     FLOW
timed   FLOW
tempo   FLOW
courier       FLOW
conference    FLOW
netnews       FLOW
netwall       FLOW
apertus-ldp   FLOW
uucp    FLOW
uucp-rlogin   FLOW
klogin FLOW
```

```
kshell FLOW
new-rwho      FLOW
dsf    FLOW
remotefs      FLOW
rmonitor      FLOW
monitor       FLOW
chshell       FLOW
p9fs   FLOW
whoami FLOW
meter  FLOW
ipcserver     FLOW
urm    FLOW
nqs    FLOW
sift-uft      FLOW
npmp-trap     FLOW
npmp-local    FLOW
npmp-gui      FLOW
ginad  FLOW
doom   FLOW
mdqs   FLOW
elcsd  FLOW
entrustmanager      FLOW
netviewdm1    FLOW
netviewdm2    FLOW
netviewdm3    FLOW
netgw  FLOW
netrcs FLOW
flexlm FLOW
fujitsu-dev   FLOW
ris-cm FLOW
kerberos-adm  FLOW
rfile  FLOW
pump   FLOW
qrh    FLOW
```

```
rrh     FLOW
tell    FLOW
nlogin FLOW
con     FLOW
ns      FLOW
rxe     FLOW
quotad FLOW
cycleserv       FLOW
omserv FLOW
webster         FLOW
phonebook       FLOW
vid     FLOW
cadlock         FLOW
rtip    FLOW
cycleserv2      FLOW
submit FLOW
rpasswd         FLOW
entomb FLOW
wpages FLOW
wpgs    FLOW
concert         FLOW
mdbs_daemon     FLOW
device FLOW
xtreelic        FLOW
maitrd FLOW
busboy FLOW
garcon FLOW
puprouter       FLOW
socks   FLOW
```

```
--------------------------------------------------------------------------
--
--  Virtual.pdl - Virtual Layer definition
--
--  Description:
--     This file contains the definition for the VirtualBase layer used
--     by the embodiment.
--
--  Copyright:
--     Copyright (c) 1998-1999 Apptitude, Inc.
--       (formerly Technically Elite, Inc.)
--     All rights reserved.
--
--  RCS:
--     $Id: Virtual.pdl,v 1.13 1999/04/13 15:48:03 skip Exp $
--
--------------------------------------------------------------------------
-- This includes two things: the flow signature (called FLOWKEY) that the
-- system that is going to use.
--
-- note that not all elements are in the HASH. Reason is that these non-HASHED
 -- elements may be varied without the HASH changing, whihc allows the system
 -- to look up multiple buckets with a single HASH. That is, the MeyMatchFlag,
 -- StateStatus Flag and MuliPacketID may be varied.
 --

FLOWKEY {
    KeyMatchFlags, -- to tell the system which of the in-HASH elements have to
 -- match for the this particular flow record.
                      -- Flows for which complete signatures may not yet have
                      -- been generated may then be stored in the system
--
StateStatusFlags,

    GroupId1              IN-HASH, -- user defined
    GroupId2              IN-HASH, -- user defined

    DLCProtocol           IN-HASH, , -- data link protocol - lowest level we
                                     -- evaluate. It is the type for the
-- Ethernet V 2
    NetworkProtocol       IN-HASH,    -- IP, etc.
    TunnelProtocol        IN-HASH,    -- IP over IPX, etc.
    TunnelTransport       IN-HASH,
    TransportProtocol     IN-HASH,
    ApplicationProtocol   IN-HASH,

    DLCAddresses(8)       IN-HASH, -- lowest level address
    NetworkAddresses(16)  IN-HASH,
    TunnelAddresses(16)   IN-HASH,
    ConnectionIds         IN-HASH,

    MultiPacketId                  -- used for fragmentaion purposes
}
-- now define all of the children. In this example, only one virtual
-- child - Ethernet.
--
virtualChildren     FIELD
              SYNTAX INT(8) { ethernet(1) }

-- now define the base for the children. In this case, it is the same as
-- for the overall system. There may be multiples.
--

VirtualBase   PROTOCOL
::= { VirtualChildren=virtualChildren }

--
```

```
-- The following is the header that every packet has to have and
-- that is placed into the system by the packet acquisition system.
--

VirtualBase   FLOW
              HEADER { LENGTH=8 }
              CHILDREN { DESTINATION=VirtualChildren } -- this will be
-- Ethernet for this example.
--
-- the VirtualBAse will be 01 for these packets.
```

```
--------------------------------------------------------------------------
--
--  Ethernet.pdl - Ethernet frame definition
--
--  Description:
--     This file contains the definition for the Ethernet frame.  In this
--  PDL file, the decision on EtherType vs. IEEE is made.  If this is
--  EtherType, the selection is made from this file.  It would be possible
--  to move the EtherType selection to another file, if that would assist
--  in the modularity.
--
--  Copyright:
--     Copyright (c) 1994-1998 Apptitude, Inc.
--       (formerly Technically Elite, Inc.)
--     All rights reserved.
--
--  RCS:
--     $Id: Ethernet.pdl,v 1.13 1999/01/26 15:15:57 skip Exp $
--
--------------------------------------------------------------------------

--
-- Enumerated type of a 16 bit integer that contains all of the
-- possible values of interest in the etherType field of an
-- Ethernet V2 packet.
--
etherType     FIELD
              SYNTAX        INT(16) { xns(0x0600), ip(0x0800),
                            chaosnet(0x0804), arp(0x0806),
                            vines(0xbad),
                            vinesLoop(0x0bae), vinesLoop(0x80c4),
                            vinesEcho(0xbaf), vinesEcho(0x80c5),
                            netbios(0x3c00), netbios(0x3c01),
                            netbios(0x3c02), netbios(0x3c03),
                            netbios(0x3c04), netbios(0x3c05),
                            netbios(0x3c06), netbios(0x3c07),
                            netbios(0x3c08), netbios(0x3c09),
                            netbios(0x3c0a), netbios(0x3c0b),
                            netbios(0x3c0c), netbios(0x3c0d),
                            dec(0x6000), mop(0x6001), mop2(0x6002),
                            drp(0x6003), lat(0x6004), decDiag(0x6005),
                            lavc(0x6007), rarp(0x8035), appleTalk(0x809b),
                            sna(0x80d5), aarp(0x80f3), ipx(0x8137),
                            snmp(0x814c), ipv6(0x86dd), loopback(0x9000) }
              DISPLAY-HINT  "1x:"
              LOOKUP        FILE "EtherType.cf"
              DESCRIPTION
                  "Ethernet type field"

--
-- The unformatted data field in and Ethernet V2 type frame
--
etherData     FIELD
              SYNTAX        BYTESTRING(46..1500)
              ENCAP         etherType
              DISPLAY-HINT  "HexDump"
              DESCRIPTION
                  "Ethernet data"

--
-- The layout and structure of an Ethernet V2 type frame with
-- the address and protocol fields in the correct offset position
--
ethernet      PROTOCOL
              DESCRIPTION
                  "Protocol format for an Ethernet frame"
              REFERENCE     "RFC 894"
```

```
    ::= { MacDest=macAddress, MacSrc=macAddress, EtherType=etherType,
          Data=etherData }

    --
    -- The elements from this Ethernet frame used to build a flow key
    -- to classify and track the traffic.  Notice that the total length
    -- of the header for this type of packet is fixed and at 14 bytes or
    -- octets in length.  The special field, LLC-CHECK, is specific to
    -- Ethernet frames for the decoding of the base Ethernet type value.
    -- If it is NOT LLC, the protocol field in the flow is set to the
    -- EtherType value decoded from the packet.
    --
    ethernet      FLOW
                  HEADER { LENGTH=14 }
                  DLC-LAYER {
                      SOURCE=MacSrc,
                      DESTINATION=MacDest,
                      TUNNELING,
                      PROTOCOL
                  }
                  CHILDREN { DESTINATION=EtherType, LLC-CHECK=llc }
```

```
-------------------------------------------------------------------------
--
--  IEEE8022.pdl - IEEE 802.2 frame definitions
--
--  Description:
--     This file contains the definition for the IEEE 802.2 Link Layer
--     protocols including the SNAP (Sub-network Access Protocol).
--
--  Copyright:
--     Copyright (c) 1994-1998 Apptitude, Inc.
--       (formerly Technically Elite, Inc.)
--     All rights reserved.
--
--  RCS:
--     $Id: IEEE8022.pdl,v 1.18 1999/01/26 15:15:58 skip Exp $
--
-------------------------------------------------------------------------

--
-- IEEE 802.2 LLC
--
llcSap FIELD
       SYNTAX        INT(16) { ipx(0xFFFF), ipx(0xE0E0), isoNet(0xFEFE),
                     netbios(0xF0F0), vsnap(0xAAAA), ip(0x0606),
                     vines(0xBCBC), xns(0x8080), spanningTree(0x4242),
                     sna(0x0c0c), sna(0x0808), sna(0x0404) }
       DISPLAY-HINT "1x:"
       DESCRIPTION
           "Service Access Point"

llcControl    FIELD
       -- This is a special field. When the decoder encounters this field, it
       -- invokes the hard-coded LLC decoder to decode the rest of the packet.
       -- This is necessary because LLC decoding requires the ability to
       -- handle forward references which the current PDL format does not
       -- support at this time.
       SYNTAX        UNSIGNED INT(8)
       DESCRIPTION
           "Control field"

llcPduType    FIELD
       SYNTAX BITSTRING(2) { llcInformation(0), llcSupervisory(1),
              llcInformation(2), llcUnnumbererd(3) }

llcData       FIELD
       SYNTAX        BYTESTRING(38..1492)
       ENCAP         llcPduType
       FLAGS         SAMELAYER
       DISPLAY-HINT  "HexDump"

llc    PROTOCOL
       SUMMARIZE
           "$llcPduType == llcUnnumbered" :
              "LLC ($SAP) $Modifier"
           "$llcPduType == llcSupervisory" :
              "LLC ($SAP) $Function N(R)=$NR"
           "$llcPduType == 0|2" :
              "LLC ($SAP) N(R)=$NR N(S)=$NS"
           "Default" :
              "LLC ($SAP) $llcPduType"
       DESCRIPTION
           "IEEE 802.2 LLC frame format"
::= { SAP=llcSap, Control=llcControl, Data=llcData }

llc     FLOW
       HEADER { LENGTH=3 }
       DLC-LAYER { PROTOCOL }
```

```
        CHILDREN { DESTINATION=SAP }

    llcUnnumberedData FIELD
        SYNTAX        BYTESTRING(0..1500)
        ENCAP         llcSap
        DISPLAY-HINT  "HexDump"

    llcUnnumbered PROTOCOL
        SUMMARIZE
            "Default" :
              "LLC ($SAP) $Modifier"
    ::= { Data=llcUnnumberedData }

    llcSupervisoryData  FIELD
        SYNTAX        BYTESTRING(0..1500)
        DISPLAY-HINT  "HexDump"

    llcSupervisory      PROTOCOL
        SUMMARIZE
            "Default" :
              "LLC ($SAP) $Function N(R)=$NR"
    ::= { Data=llcSupervisoryData }

    llcInformationData  FIELD
        SYNTAX        BYTESTRING(0..1500)
        ENCAP         llcSap
        DISPLAY-HINT  "HexDump"

    llcInformation      PROTOCOL
        SUMMARIZE
            "Default" :
              "LLC ($SAP) N(R)=$NR N(S)=$NS"
    ::= { Data=llcInformationData }

    --
    -- SNAP
    --
    snapOrgCode  FIELD
        SYNTAX        BYTESTRING(3) { snap("00:00:00"), ciscoOUI("00:00:0C"),
                      appleOUI("08:00:07") }
        DESCRIPTION
            "Protocol ID or Organizational Code"

    vsnapData    FIELD
        SYNTAX        BYTESTRING(46..1500)
        ENCAP         snapOrgCode
        FLAGS         SAMELAYER
        DISPLAY-HINT  "HexDump"
        DESCRIPTION
            "SNAP LLC data"

    vsnap  PROTOCOL
        DESCRIPTION
            "SNAP LLC Frame"
    ::= { OrgCode=snapOrgCode, Data=vsnapData }

    vsnap       FLOW
                HEADER { LENGTH=3 }
                DLC-LAYER { PROTOCOL }
                CHILDREN { DESTINATION=OrgCode }

    snapType     FIELD
                SYNTAX  INT(16) { xns(0x0600), ip(0x0800), arp(0x0806),
                        vines(0xbad),
                        mop(0x6001), mop2(0x6002), drp(0x6003),
                        lat(0x6004), decDiag(0x6005), lavc(0x6007),
                        rarp(0x8035), appleTalk(0x809B), sna(0x80d5),
```

```
            aarp(0x80F3), ipx(0x8137), snmp(0x814c), ipv6(0x86dd) }
        DISPLAY-HINT    "1x:"
        LOOKUP          FILE "EtherType.cf"
        DESCRIPTION
          "SNAP type field"

snapData      FIELD
       SYNTAX        BYTESTRING(46..1500)
       ENCAP         snapType
       DISPLAY-HINT  "HexDump"
       DESCRIPTION
           "SNAP data"

snap   PROTOCOL
       SUMMARIZE
           "$OrgCode == 00:00:00" :
             "SNAP Type=$SnapType"
           "Default" :
             "VSNAP Org=$OrgCode Type=$SnapType"
       DESCRIPTION
           "SNAP Frame"
::= { SnapType=snapType, Data=snapData }

snap    FLOW
       HEADER { LENGTH=2 }
       DLC-LAYER { PROTOCOL }
       CHILDREN { DESTINATION=SnapType }
```

```
--------------------------------------------------------------------------
--
--  IEEE8023.pdl - IEEE 802.3 frame definitions
--
--  Description:
--     This file contains the definition for the IEEE 802.3 (Ethernet)
--     protocols.
--
--  Copyright:
--     Copyright (c) 1994-1998 Apptitude, Inc.
--       (formerly Technically Elite, Inc.)
--     All rights reserved.
--
--  RCS:
--     $Id: IEEE8023.pdl,v 1.7 1999/01/26 15:15:58 skip Exp $
--
--------------------------------------------------------------------------


--
-- IEEE 802.3
--
ieee8023Length      FIELD
       SYNTAX UNSIGNED INT(16)

ieee8023Data FIELD
       SYNTAX        BYTESTRING(38..1492)
       ENCAP         =llc
       LENGTH        "$ieee8023Length"
       DISPLAY-HINT  "HexDump"

ieee8023      PROTOCOL
       DESCRIPTION
           "IEEE 802.3 (Ethernet) frame"
       REFERENCE     "RFC 1042"
::= { MacDest=macAddress, MacSrc=macAddress, Length=ieee8023Length,
       Data=ieee8023Data }
```

```
--------------------------------------------------------------------------------
--
--  IP.pdl - Internet Protocol (IP) definitions
--
--  Description:
--     This file contains the packet definitions for the Internet
--     Protocol.  These elements are all of the fields, templates and
--  processes required to recognize, decode and classify IP datagrams
--  found within packets.
--
--  Copyright:
--     Copyright (c) 1994-1998 Apptitude, Inc.
--       (formerly Technically Elite, Inc.)
--     All rights reserved.
--
--  RCS:
--     $Id: IP.pdl,v 1.14 1999/01/26 15:15:58 skip Exp $
--
--------------------------------------------------------------------------------

--
-- The following are the fields that make up an IP datagram.
-- Some of these fields are used to recognize datagram elements, build
-- flow signatures and determine the next layer in the decode process.
--
ipVersion     FIELD
              SYNTAX INT(4)
              DEFAULT       "4"

ipHeaderLength      FIELD
              SYNTAX INT(4)

ipTypeOfService     FIELD
              SYNTAX BITSTRING(8) { minCost(1), maxReliability(2),
                     maxThruput(3), minDelay(4) }

ipLength      FIELD
              SYNTAX UNSIGNED INT(16)

--
-- This field will tell us if we need to do special processing to support
-- the payload of the datagram existing in multiple packets.
--
ipFlags             FIELD
              SYNTAX BITSTRING(3) { moreFrags(0), dontFrag(1) }

ipFragmentOffset FIELD
              SYNTAX INT(13)

--
-- This field is used to determine the children or next layer of the
-- datagram.
--
ipProtocol    FIELD
              SYNTAX INT(8)
              LOOKUP FILE "IpProtocol.cf"

ipData        FIELD
              SYNTAX        BYTESTRING(0..1500)
              ENCAP         ipProtocol
              DISPLAY-HINT  "HexDump"

--
-- Detailed packet layout for the IP datagram.  This includes all fields
-- and format.  All offsets are relative to the beginning of the header.
--
ip     PROTOCOL
```

```
        SUMMARIZE
             "$FragmentOffset != 0":
                 "IPFragment ID=$Identification Offset=$FragmentOffset"
             "Default" :
                 "IP Protocol=$Protocol"
        DESCRIPTION
            "Protocol format for the Internet Protocol"
        REFERENCE     "RFC 791"
    ::= { Version=ipVersion, HeaderLength=ipHeaderLength,
        TypeOfService=ipTypeOfService, Length=ipLength,
        Identification=UInt16, IpFlags=ipFlags,
        FragmentOffset=ipFragmentOffset, TimeToLive=Int8,
        Protocol=ipProtocol, Checksum=ByteStr2,
        IpSrc=ipAddress, IpDest=ipAddress, Options=ipOptions,
        Fragment=ipFragment, Data=ipData }

    --
    -- This is the description of the signature elements required to build a flow
    -- that includes the IP network layer protocol.  Notice that the flow builds on
    -- the lower layers.  Only the fields required to complete IP are included.
    -- This flow requires the support of the fragmentation engine as well as the
    -- potential of having a tunnel.  The child field is found from the IP
    -- protocol field.
    --
    ip      FLOW
            HEADER { LENGTH=HeaderLength, IN-WORDS }
            NET-LAYER {
                SOURCE=IpSrc,
                DESTINATION=IpDest,
                FRAGMENTATION=IPV4,
                TUNNELING
            }
            CHILDREN { DESTINATION=Protocol }

    ipFragData    FIELD
                  SYNTAX        BYTESTRING(1..1500)
                  LENGTH        "$ipLength - $ipHeaderLength * 4"
                  DISPLAY-HINT  "HexDump"

    ipFragment    GROUP
                  OPTIONAL      "$FragmentOffset != 0"
    ::= { Data=ipFragData }

    ipOptionCode  FIELD
                  SYNTAX INT(8) { ipRR(0x07), ipTimestamp(0x44),
                         ipLSRR(0x83), ipSSRR(0x89) }
                  DESCRIPTION
                      "IP option code"

    ipOptionLength      FIELD
                  SYNTAX UNSIGNED INT(8)
                  DESCRIPTION
                      "Length of IP option"

    ipOptionData  FIELD
                  SYNTAX        BYTESTRING(0..1500)
                  ENCAP         ipOptionCode
                  DISPLAY-HINT  "HexDump"

    ipOptions     GROUP
                  LENGTH        "($ipHeaderLength * 4) - 20"
    ::= { Code=ipOptionCode, Length=ipOptionLength, Pointer=UInt8,
          Data=ipOptionData }
```

```
-------------------------------------------------------------------------
--
--  TCP.pdl - Transmission Control Protocol (TCP) definitions
--
--  Description:
--     This file contains the packet definitions for the Transmission
--     Control Protocol.  This protocol is a transport service for
--  the IP protocol.  In addition to extracting the protocol information
--  the TCP protocol assists in the process of identification of connections
--  for the processing of states.
--
--  Copyright:
--     Copyright (c) 1994-1998 Apptitude, Inc.
--       (formerly Technically Elite, Inc.)
--     All rights reserved.
--
--  RCS:
--     $Id: TCP.pdl,v 1.9 1999/01/26 15:16:02 skip Exp $
--
-------------------------------------------------------------------------
--
-- This is the 16 bit field where the child protocol is located for
-- the next layer beyond TCP.
--
tcpPort FIELD
       SYNTAX UNSIGNED INT(16)
       LOOKUP FILE "TcpPort.cf"

tcpHeaderLen FIELD
       SYNTAX INT(4)

tcpFlags FIELD
       SYNTAX BITSTRING(12) { fin(0), syn(1), rst(2), psh(3), ack(4), urg(5) }

tcpData FIELD
       SYNTAX        BYTESTRING(0..1564)
       LENGTH        "($ipLength - ($ipHeaderLength * 4)) - ($tcpHeaderLen * 4)"
       ENCAP         tcpPort
       DISPLAY-HINT  "HexDump"

--
-- The layout of the TCP datagram found in a packet.  Offset based on the
-- beginning of the header for TCP.
--
tcp PROTOCOL
       SUMMARIZE
           "Default" :
              "TCP ACK=$Ack WIN=$WindowSize"
       DESCRIPTION
           "Protocol format for the Transmission Control Protocol"
       REFERENCE     "RFC 793"
::= { SrcPort=tcpPort, DestPort=tcpPort, SequenceNum=UInt32,
       Ack=UInt32, HeaderLength=tcpHeaderLen, TcpFlags=tcpFlags,
       WindowSize=UInt16, Checksum=ByteStr2,
       UrgentPointer=UInt16, Options=tcpOptions, Data=tcpData }

--
-- The flow elements required to build a key for a TCP datagram.
-- Noticed that this FLOW description has a CONNECTION section. This is
-- used to describe what connection state is reached for each setting
-- of the TcpFlags field.
--
tcp      FLOW
        HEADER { LENGTH=HeaderLength, IN-WORDS }
        CONNECTION {
            IDENTIFIER=SequenceNum,
            CONNECT-START="TcpFlags:1",
```

```
            CONNECT-COMPLETE="TcpFlags:4",
            DISCONNECT-START="TcpFlags:0",
            DISCONNECT-COMPLETE="TcpFlags:4"
        }
        PAYLOAD { INCLUDE-HEADER }
        CHILDREN { DESTINATION=DestPort, SOURCE=SrcPort }

tcpOptionKind FIELD
             SYNTAX UNSIGNED INT(8) { tcpOptEnd(0), tcpNop(1), tcpMSS(2),
                          tcpWscale(3), tcpTimestamp(4) }
             DESCRIPTION
                 "Type of TCP option"

tcpOptionData FIELD
             SYNTAX       BYTESTRING(0..1500)
             ENCAP        tcpOptionKind
             FLAGS        SAMELAYER
             DISPLAY-HINT "HexDump"

tcpOptions   GROUP
             LENGTH       "($tcpHeaderLen * 4) - 20"
--           SUMMARIZE
--               "Default" :
--                  "Option=$Option, Len=$OptionLength, $OptionData"
::= { Option=tcpOptionKind, OptionLength=UInt8, OptionData=tcpOptionData }

tcpMSS       PROTOCOL
::= { MaxSegmentSize=UInt16 }
```

```
--------------------------------------------------------------------------------
--
--  UDP.pdl - User Datagram Protocol (UDP) definitions
--
--  Description:
--     This file contains the packet definitions for the User Datagram
--     Protocol.
--
--  Copyright:
--     Copyright (c) 1994-1998 Apptitude, Inc.
--       (formerly Technically Elite, Inc.)
--     All rights reserved.
--
--  RCS:
--     $Id: UDP.pdl,v 1.9 1999/01/26 15:16:02 skip Exp $
--
--------------------------------------------------------------------------------
udpPort       FIELD
       SYNTAX UNSIGNED INT(16)
       LOOKUP FILE "UdpPort.cf"

udpLength FIELD
       SYNTAX        UNSIGNED INT(16)

udpData FIELD
       SYNTAX        BYTESTRING(0..1500)
       ENCAP         udpPort
       DISPLAY-HINT  "HexDump"

udp    PROTOCOL
       SUMMARIZE
           "Default" :
              "UDP Dest=$DestPort Src=$SrcPort"
       DESCRIPTION
           "Protocol format for the User Datagram Protocol."
       REFERENCE     "RFC 768"
::= { SrcPort=udpPort, DestPort=udpPort, Length=udpLength,
       Checksum=ByteStr2, Data=udpData }

udp    FLOW
       HEADER { LENGTH=8 }
       CHILDREN { DESTINATION=DestPort, SOURCE=SrcPort }
```

```
-------------------------------------------------------------------------
--
--  RPC.pdl - Remote Procedure Calls (RPC) definitions
--
--  Description:
--     This file contains the packet definitions for Remote Procedure
--     Calls.
--
--  Copyright:
--     Copyright (c) 1994-1999 Apptitude, Inc.
--       (formerly Technically Elite, Inc.)
--     All rights reserved.
--
--  RCS:
--     $Id: RPC.pdl,v 1.7 1999/01/26 15:16:01 skip Exp $
--
-------------------------------------------------------------------------
rpcType       FIELD
       SYNTAX UNSIGNED INT(32) { rpcCall(0), rpcReply(1) }

rpcData       FIELD
       SYNTAX         BYTESTRING(0..100)
       ENCAP          rpcType
       FLAGS          SAMELAYER
       DISPLAY-HINT   "HexDump"

rpc     PROTOCOL
        SUMMARIZE
            "$Type == rpcCall" :
               "RPC $Program"
            "$ReplyStatus == rpcAcceptedReply" :
               "RPC Reply Status=$Status"
            "$ReplyStatus == rpcDeniedReply" :
               "RPC Reply Status=$Status, AuthStatus=$AuthStatus"
            "Default" :
               "RPC $Program"
        DESCRIPTION
            "Protocol format for RPC"
        REFERENCE
            "RFC 1057"
::= { XID=UInt32, Type=rpcType, Data=rpcData }

rpc    FLOW
       HEADER { LENGTH=0 }
       PAYLOAD { DATA=XID, LENGTH=256 }

-------------
-- RPC Call
-------------
rpcProgram FIELD
       SYNTAX UNSIGNED INT(32) { portMapper(100000), nfs(100003),
              mount(100005), lockManager(100021), statusMonitor(100024) }

rpcProcedure GROUP
       SUMMARIZE
           "Default" :
              "Program=$Program, Version=$Version, Procedure=$Procedure"
::= { Program=rpcProgram, Version=UInt32, Procedure=UInt32 }

rpcAuthFlavor FIELD
       SYNTAX UNSIGNED INT(32) { null(0), unix(1), short(2) }

rpcMachine    FIELD
       SYNTAX LSTRING(4)

rpcGroup      GROUP
       LENGTH "$NumGroups * 4"
```

```
::= { Gid=Int32 }

rpcCredentials      GROUP
      LENGTH "$CredentialLength"
::= { Stamp=UInt32, Machine=rpcMachine, Uid=Int32, Gid=Int32,
      NumGroups=UInt32, Groups=rpcGroup }

rpcVerifierData     FIELD
      SYNTAX        BYTESTRING(0..400)
      LENGTH        "$VerifierLength"

rpcEncap      FIELD
      SYNTAX COMBO Program Procedure
      LOOKUP FILE "RPC.cf"

rpcCallData   FIELD
      SYNTAX        BYTESTRING(0..100)
      ENCAP         rpcEncap
      DISPLAY-HINT  "HexDump"

rpcCall       PROTOCOL
      DESCRIPTION
          "Protocol format for RPC call"
::= { RPCVersion=UInt32, Procedure=rpcProcedure,
      CredentialAuthFlavor=rpcAuthFlavor, CredentialLength=UInt32,
      Credentials=rpcCredentials,
      VerifierAuthFlavor=rpcAuthFlavor, VerifierLength=UInt32,
      Verifier=rpcVerifierData, Encap=rpcEncap, Data=rpcCallData }

-------------
-- RPC Reply
-------------
rpcReplyStatus      FIELD
      SYNTAX INT(32) { rpcAcceptedReply(0), rpcDeniedReply(1) }

rpcReplyData  FIELD
      SYNTAX        BYTESTRING(0..40000)
      ENCAP         rpcReplyStatus
      FLAGS         SAMELAYER
      DISPLAY-HINT  "HexDump"

rpcReply      PROTOCOL
      DESCRIPTION
          "Protocol format for RPC reply"
::= { ReplyStatus=rpcReplyStatus, Data=rpcReplyData }

rpcAcceptStatus     FIELD
      SYNTAX INT(32) { Success(0), ProgUnavail(1), ProgMismatch(2),
            ProcUnavail(3), GarbageArgs(4), SystemError(5) }

rpcAcceptEncap      FIELD
      SYNTAX BYTESTRING(0)
      FLAGS  NOSHOW

rpcAcceptData FIELD
      SYNTAX        BYTESTRING(0..40000)
      ENCAP         rpcAcceptEncap
      DISPLAY-HINT  "HexDump"

rpcAcceptedReply PROTOCOL
::= { VerifierAuthFlavor=rpcAuthFlavor, VerifierLength=UInt32,
      Verifier=rpcVerifierData, Status=rpcAcceptStatus,
      Encap=rpcAcceptEncap, Data=rpcAcceptData }

rpcDeniedStatus     FIELD
      SYNTAX INT(32) { rpcVersionMismatch(0), rpcAuthError(1) }
```

```
--------------------------------------------------------------------------------
--
--  NFS.pdl - Network File System (NFS) definitions
--
--  Description:
--     This file contains the packet definitions for the Network File
--     System.
--
--  Copyright:
--     Copyright (c) 1994-1998 Apptitude, Inc.
--       (formerly Technically Elite, Inc.)
--     All rights reserved.
--
--  RCS:
--     $Id: NFS.pdl,v 1.3 1999/01/26 15:15:59 skip Exp $
--
--------------------------------------------------------------------------------
nfsString     FIELD
        SYNTAX LSTRING(4)

nfsHandle     FIELD
        SYNTAX        BYTESTRING(32)
        DISPLAY-HINT  "16x\n          "

nfsData               FIELD
        SYNTAX        BYTESTRING(0..100)
        DISPLAY-HINT  "HexDump"

nfsAccess     PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS Access $Filename"
::= { Handle=nfsHandle, Filename=nfsString }

nfsStatus     FIELD
        SYNTAX INT(32) { OK(0), NoSuchFile(2) }

nfsAccessReply       PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS AccessReply $Status"
::= { Status=nfsStatus }

nfsMode               FIELD
        SYNTAX UNSIGNED INT(32)
        DISPLAY-HINT  "4o"

nfsCreate     PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS Create $Filename"
::= { Handle=nfsHandle, Filename=nfsString, Filler=Int8, Mode=nfsMode,
        Uid=Int32, Gid=Int32, Size=Int32, AccessTime=Int64, ModTime=Int64 }

nfsFileType   FIELD
        SYNTAX INT(32) { Regular(1), Directory(2) }

nfsCreateReply       PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS CreateReply $Status"
::= { Status=nfsStatus, Handle=nfsHandle, FileType=nfsFileType,
        Mode=nfsMode, Links=UInt32, Uid=Int32, Gid=Int32, Size=Int32,
        BlockSize=Int32, NumBlocks=Int64, FileSysId=UInt32, FileId=UInt32,
        AccessTime=Int64, ModTime=Int64, InodeChangeTime=Int64 }

nfsRead       PROTOCOL
```

```
        SUMMARIZE
            "Default" :
              "NFS Read Offset=$Offset Length=$Length"
    ::= { Length=Int32, Handle=nfsHandle, Offset=UInt64, Count=Int32 }

nfsReadReply PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS ReadReply $Status"
    ::= { Status=nfsStatus, FileType=nfsFileType,
        Mode=nfsMode, Links=UInt32, Uid=Int32, Gid=Int32, Size=Int32,
        BlockSize=Int32, NumBlocks=Int64, FileSysId=UInt32, FileId=UInt32,
        AccessTime=Int64, ModTime=Int64, InodeChangeTime=Int64 }

nfsWrite PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS Write Offset=$Offset"
    ::= { Handle=nfsHandle, Offset=Int32, Data=nfsData }

nfsWriteReply PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS WriteReply $Status"
    ::= { Status=nfsStatus, FileType=nfsFileType,
        Mode=nfsMode, Links=UInt32, Uid=Int32, Gid=Int32, Size=Int32,
        BlockSize=Int32, NumBlocks=Int64, FileSysId=UInt32, FileId=UInt32,
        AccessTime=Int64, ModTime=Int64, InodeChangeTime=Int64 }

nfsReadDir   PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS ReadDir"
    ::= { Handle=nfsHandle, Cookie=Int32, Count=Int32 }

nfsReadDirReply     PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS ReadDirReply $Status"
    ::= { Status=nfsStatus, Data=nfsData }

nfsGetFileAttr      PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS GetAttr"
    ::= { Handle=nfsHandle }

nfsGetFileAttrReply PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS GetAttrReply $Status $FileType"
    ::= { Status=nfsStatus, FileType=nfsFileType,
        Mode=nfsMode, Links=UInt32, Uid=Int32, Gid=Int32, Size=Int32,
        BlockSize=Int32, NumBlocks=Int64, FileSysId=UInt32, FileId=UInt32,
        AccessTime=Int64, ModTime=Int64, InodeChangeTime=Int64 }

nfsReadLink  PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS ReadLink"
    ::= { Handle=nfsHandle }

nfsReadLinkReply PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS ReadLinkReply Path=$Path"
    ::= { Status=nfsStatus, Path=nfsString }
```

```
nfsMount      PROTOCOL
       SUMMARIZE
           "Default" :
              "NFS Mount $Path"
::= { Path=nfsString }

nfsMountReply PROTOCOL
       SUMMARIZE
           "Default" :
              "NFS MountReply $MountStatus"
::= { MountStatus=nfsStatus, Handle=nfsHandle }

nfsStatFs     PROTOCOL
       SUMMARIZE
           "Default" :
              "NFS StatFs"
::= { Handle=nfsHandle }

nfsStatFsReply      PROTOCOL
       SUMMARIZE
           "Default" :
              "NFS StatFsReply $Status"
::= { Status=nfsStatus, TransferSize=UInt32, BlockSize=UInt32,
      TotalBlocks=UInt32, FreeBlocks=UInt32, AvailBlocks=UInt32 }

nfsRemoveDir  PROTOCOL
       SUMMARIZE
           "Default" :
              "NFS RmDir $Name"
::= { Handle=nfsHandle, Name=nfsString }

nfsRemoveDirReply PROTOCOL
       SUMMARIZE
           "Default" :
              "NFS RmDirReply $Status"
::= { Status=nfsStatus }

nfsMakeDir    PROTOCOL
       SUMMARIZE
           "Default" :
              "NFS MkDir $Name"
::= { Handle=nfsHandle, Name=nfsString }

nfsMakeDirReply PROTOCOL
       SUMMARIZE
           "Default" :
              "NFS MkDirReply $Status"
::= { Status=nfsStatus }

nfsRemove     PROTOCOL
       SUMMARIZE
           "Default" :
              "NFS Remove $Name"
::= { Handle=nfsHandle, Name=nfsString }

nfsRemoveReply PROTOCOL
       SUMMARIZE
           "Default" :
              "NFS RemoveReply $Status"
::= { Status=nfsStatus }
```

```
--------------------------------------------------------------------------
--
--  HTTP.pdl - Hypertext Transfer Protocol (HTTP) definitions
--
--  Description:
--      This file contains the packet definitions for the Hypertext Transfer
--      Protocol.
--
--  Copyright:
--      Copyright (c) 1994-1999 Apptitude, Inc.
--      (formerly Technically Elite, Inc.)
--      All rights reserved.
--
--  RCS:
--      $Id: HTTP.pdl,v 1.13 1999/04/13 15:47:57 skip Exp $
--
--------------------------------------------------------------------------
httpData FIELD
         SYNTAX          BYTESTRING(1..1500)
         LENGTH            "($ipLength - ($ipHeaderLength * 4)) - ($tcpHeaderLen
* 4)"
         DISPLAY-HINT    "Text"
         FLAGS           NOLABEL

http     PROTOCOL
         SUMMARIZE
             "$httpData m/^GET|^HTTP|^HEAD|^POST/" :
                 "HTTP $httpData"
             "$httpData m/^[Dd]ate|^[Ss]erver|^[Ll]ast-[Mm]odified/" :
                 "HTTP $httpData"
             "$httpData m/^[Cc]ontent-/" :
                 "HTTP $httpData"
             "$httpData m/^<HTML>/" :
                 "HTTP [HTML document]"
             "$httpData m/^GIF/" :
                 "HTTP [GIF image]"
             "Default" :
                 "HTTP [Data]"
         DESCRIPTION
             "Protocol format for HTTP."
::= { Data=httpData }

http     FLOW
         CONNECTION { INHERITED }
         PAYLOAD { INCLUDE-HEADER, DATA=Data, LENGTH=256 }
         STATES
            "S0: CHECKCONNECT, GOTO S1
                 DEFAULT NEXT S0

             S1: WAIT 2, GOTO S2, NEXT S1
                 DEFAULT NEXT S0

             S2: MATCH
                   '\n\r\n'          900 0 0 255 0, NEXT S3
                   '\n\n'            900 0 0 255 0, NEXT S3
                   'POST /tds?'       50 0 0 127 1, CHILD sybaseWebsql
                   '.hts HTTP/1.0'    50 4 0 127 1, CHILD sybaseJdbc
                   'jdbc:sybase:Tds'  50 4 0 127 1, CHILD sybaseTds
                   'PCN-The Poin'    500 4 1 255 0, CHILD pointcast
                   't: BW-C-'        100 4 1 255 0, CHILD backweb
                   DEFAULT NEXT S3

             S3: MATCH
                   '\n\r\n'           50 0 0   0 0, NEXT S3
                   '\n\n'             50 0 0   0 0, NEXT S3
                   'Content-Type:'   800 0 0 255 0, CHILD mime
                   'PCN-The Poin'    500 4 1 255 0, CHILD pointcast
```

```
                    't: BW-C-'        100 4 1 255 0, CHILD backweb
                    DEFAULT NEXT S0"

sybaseWebsql    FLOW
                STATE-BASED

sybaseJdbc      FLOW
                STATE-BASED

sybaseTds       FLOW
                STATE-BASED

pointcast       FLOW
                STATE-BASED

backweb         FLOW
                STATE-BASED

mime            FLOW
                STATE-BASED
                STATES
                  "S0:   MATCH
                           'application' 900 0 0   1 0, CHILD mimeApplication
                           'audio'       900 0 0   1 0, CHILD mimeAudio
                           'image'        50 0 0   1 0, CHILD mimeImage
                           'text'         50 0 0   1 0, CHILD mimeText
                           'video'        50 0 0   1 0, CHILD mimeVideo
                           'x-world'     500 4 1 255 0, CHILD mimeXworld
                         DEFAULT GOTO S0"

mimeApplication FLOW
                STATE-BASED

mimeAudio   FLOW
            STATE-BASED
            STATES
             "S0: MATCH
                  'basic'             100 0 0 1 0, CHILD pdBasicAudio
                  'midi'              100 0 0 1 0, CHILD pdMidi
                  'mpeg'              100 0 0 1 0, CHILD pdMpeg2Audio
                  'vnd.rn-realaudio'  100 0 0 1 0, CHILD pdRealAudio
                  'wav'               100 0 0 1 0, CHILD pdWav
                  'x-aiff'            100 0 0 1 0, CHILD pdAiff
                  'x-midi'            100 0 0 1 0, CHILD pdMidi
                  'x-mpeg'            100 0 0 1 0, CHILD pdMpeg2Audio
                  'x-mpgurl'          100 0 0 1 0, CHILD pdMpeg3Audio
                  'x-pn-realaudio'    100 0 0 1 0, CHILD pdRealAudio
                  'x-wav'             100 0 0 1 0, CHILD pdWav
                 DEFAULT GOTO S0"

mimeImage   FLOW
            STATE-BASED

mimeText    FLOW
            STATE-BASED

mimeVideo   FLOW
            STATE-BASED

mimeXworld FLOW
            STATE-BASED

pdBasicAudio FLOW
             STATE-BASED

pdMidi        FLOW
              STATE-BASED
```

```
pdMpeg2Audio FLOW
             STATE-BASED

pdMpeg3Audio FLOW
             STATE-BASED

pdRealAudio  FLOW
             STATE-BASED

pdWav        FLOW
             STATE-BASED

pdAiff       FLOW
             STATE-BASED
```

CLAIMS

What is claimed is:

1. A method of performing protocol specific operations on a packet passing through a connection point on a computer network, the method comprising:

(a) receiving the packet;

(b) receiving a set of protocol descriptions for a plurality of protocols that conform to a layered model, a protocol description for a particular protocol at a particular layer level including:

(i) the none or more child protocols of the particular protocol, the packet including for any particular child protocol of the particular protocol information at one or more locations in the packet related to the particular child protocol,

(ii) the one or more locations in the packet where information is stored related to any child protocol of the particular protocol, and

(iii) the none or more protocol specific operations to be performed on the packet for the particular protocol at the particular layer level; and

(c) performing the protocol specific operations on the packet specified by the set of protocol descriptions based on the base protocol of the packet and the children of the protocols used in the packet.

2. A method according to claim 1, wherein step (c) of performing protocol specific operations is performed recursively for any children of the children.

3. A method according to claim 1, wherein which protocol specific operations are performed is step (c) depends on the contents of the packet such that the method adapts to different protocols according to the contents of the packet.

4. A method according to claim 1, further comprising:

storing a database in a memory, the database generated from the set of protocol descriptions and including a data structure containing information on the possible protocols and organized for locating the child protocol related information for any protocol, the data structure contents indexed by a set of one or more indices, the database entry indexed by a particular set of index values including an indication of validity,

wherein the child protocol related information includes a child recognition pattern,

wherein step (c) of performing the protocol specific operations includes, at any particular protocol layer level starting from the base level, searching the packet at the particular protocol for the child field, the searching including indexing the data structure until a valid entry is found, and

whereby the data structure is configured for rapid searches using the index set.

5. A method according to claim 4, wherein the protocol descriptions are provided in a protocol description language, the method further comprising:

compiling the PDL descriptions to produce the database.

6. A method according to claim 4, wherein the data structure comprises a set of arrays, each array identified by a first index, at least one array for each protocol, each array further indexed by a second index being the location in the packet where the child protocol related information is stored, such that finding a valid entry in the data structure provides the location in the packet for finding the child recognition pattern for an identified protocol.

7. A method according to claim 6, wherein each array is further indexed by a third index being the size of the region in the packet where the child protocol related information is stored, such that finding a valid entry in the data structure provides the location and the size of the region in the packet for finding the child recognition pattern.

8. A method according to claim 7, wherein the data structure is compressed according to a compression scheme that takes advantage of the sparseness of valid entries in the data structure.

9. A method according to claim 8, wherein the compression scheme combines two or more arrays that have no conflicting common entries.

10. A method according to claim 4, wherein the data structure includes a set of tables, each table identified by a first index, at least one table for each protocol, each table further indexed by a second index being the child recognition pattern, the data structure further including a table that for each protocol provides the location in the packet where the child protocol related information is stored, such that finding a valid entry in the data structure provides the location in the packet for finding the child recognition pattern for an identified protocol.

11. A method according to claim 10, wherein the data structure is compressed according to a compression scheme that takes advantage of the sparseness of valid entries in the set of tables.

12. A method according to claim 11, wherein the compression scheme combines two or more tables that have no conflicting common entries.

13. A method according to claim 1, wherein the protocol specific operations include one or more parsing and extraction operations on the packet to extract selected portions of the packet to form a function of the selected portions for identifying the packet as belonging to a conversational flow.

14. A method according to claim 1, wherein the protocol descriptions are provided in a protocol description language.

15. A method according to claim 14, further comprising:

compiling the PDL descriptions to produce a database and store the database in a memory, the database generated from the set of protocol descriptions and including a data structure containing information on the possible protocols and organized for locating the child protocol related information for any protocol, the data structure contents indexed by a set of one or more indices, the database entry indexed by a particular set of index values including an indication of validity,

wherein the child protocol related information includes a child recognition pattern, and

wherein the step of performing the protocol specific operations includes, at any particular protocol layer level starting from the base level, searching the packet at the particular protocol for the child field, the searching including indexing the data structure until a valid entry is found,

whereby the data structure is configured for rapid searches using the index set.

16. A method according to claim 13, further comprising:

looking up a flow-entry database comprising none or more flow-entries, at least one flow-entry for each previously encountered conversational flow, the looking up using at least some of the selected packet portions and determining if the packet matches an existing flow-entry;

if the packet is of an existing flow, classifying the packet as belonging to the found existing flow; and

if the packet is of a new flow, storing 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,

wherein the parsing and extraction operations depend on the contents of none or more packet headers.

17. A method according to claim 13, wherein the protocol specific operations further include one or more state processing operations that are a function of the state of the flow of the packet.

18. A method according to claim 1, wherein the protocol specific operations include one or more state processing operations that are a function of the state of the flow of the packet.

ABSTRACT

A method of performing protocol specific operations on a packet passing through a connection point on a computer network. The packet contents conform to protocols of a layered model wherein the protocol at a at a particular layer level may include one or a set of child protocols defined for that level. The method includes receiving the packet and receiving a set of protocol descriptions for protocols may be used in the packet. A protocol description for a particular protocol at a particular layer level includes any child protocols of the particular protocol, and for any child protocol, where in the packet information related to the particular child protocol may be found. A protocol description also includes any protocol specific operations to be performed on the packet for the particular protocol at the particular layer level. The method includes performing the protocol specific operations on the packet specified by the set of protocol descriptions based on the base protocol of the packet and the children of the protocols used in the packet. A particular embodiment includes providing the protocol descriptions in a high-level protocol description language, and compiling to the descriptions into a data structure. The compiling may further include compressing the data structure into a compressed data structure. The protocol specific operations may include parsing and extraction operations to extract identifying information. The protocol specific operations may also include state processing operations defined for a particular state of a conversational flow of the packet.

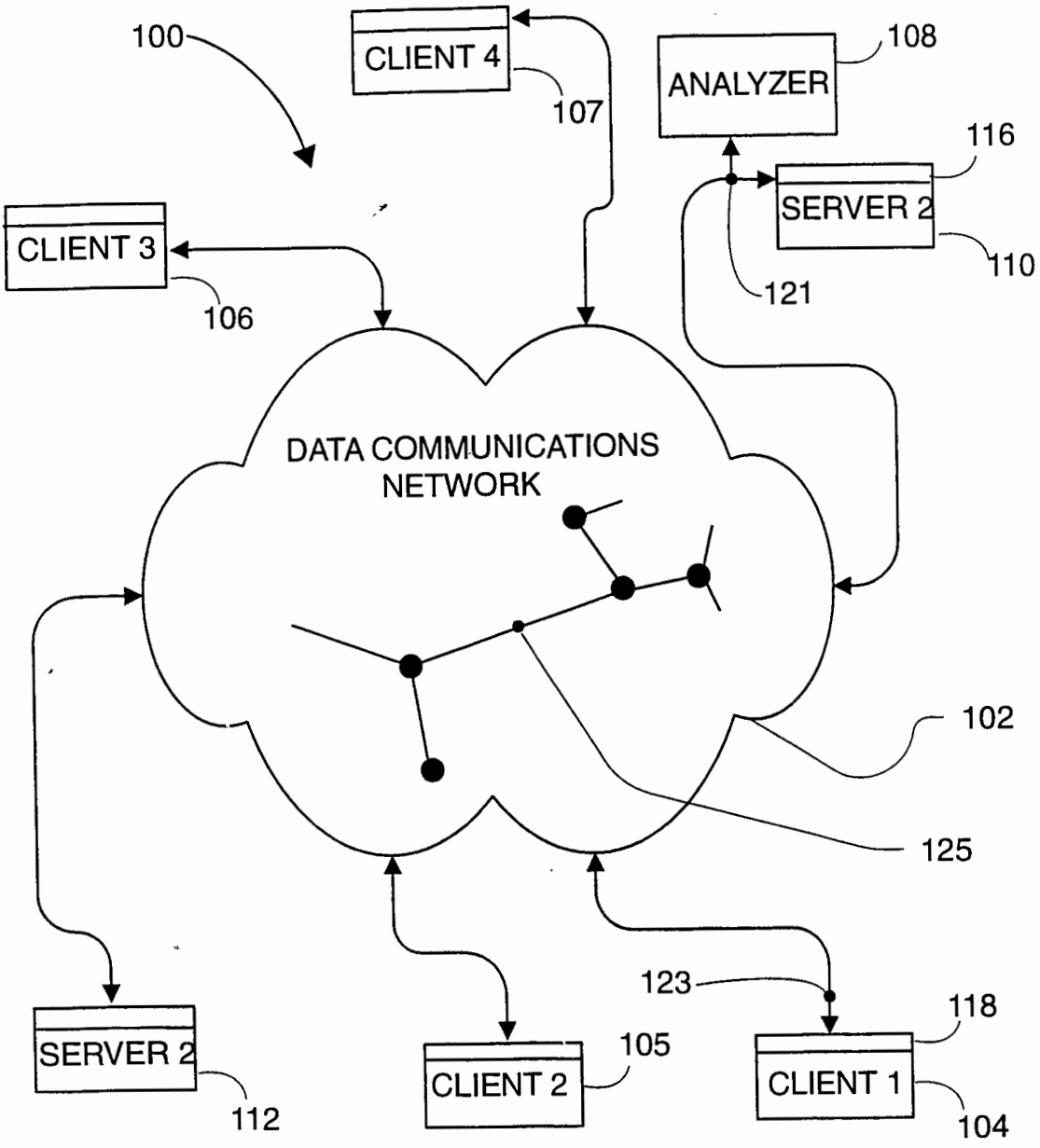


FIG. 1

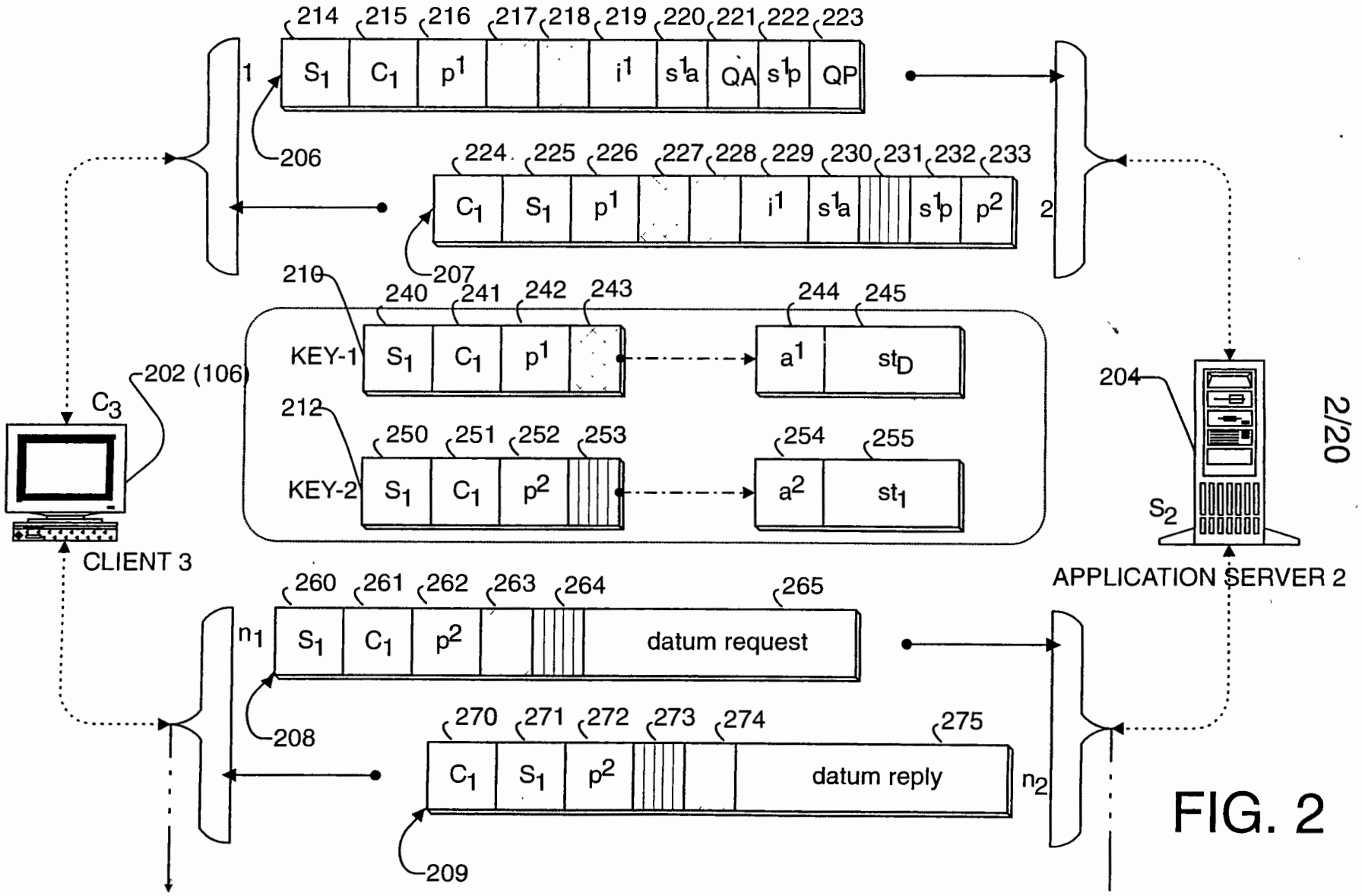


FIG. 2

2/20

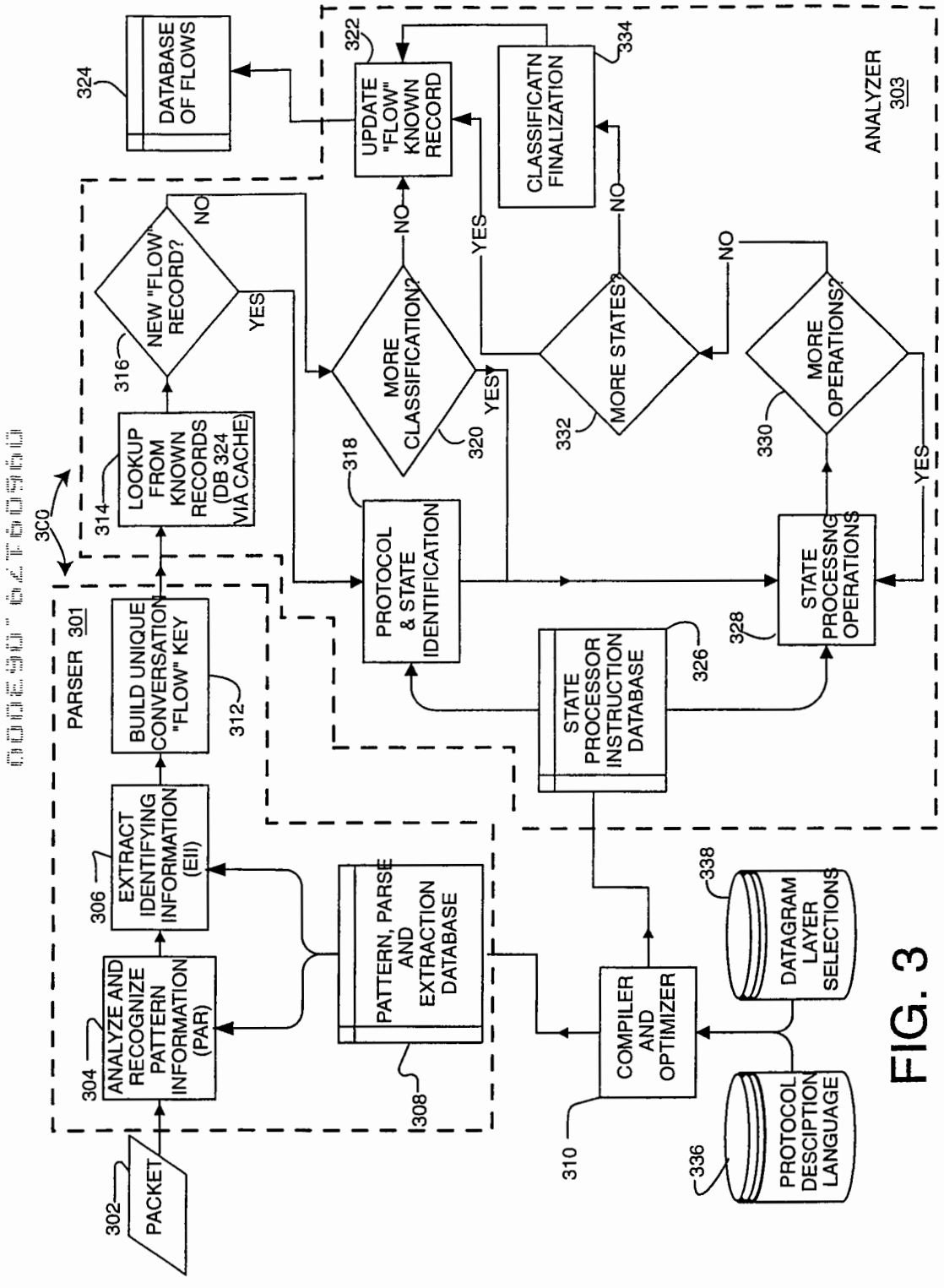


FIG. 3

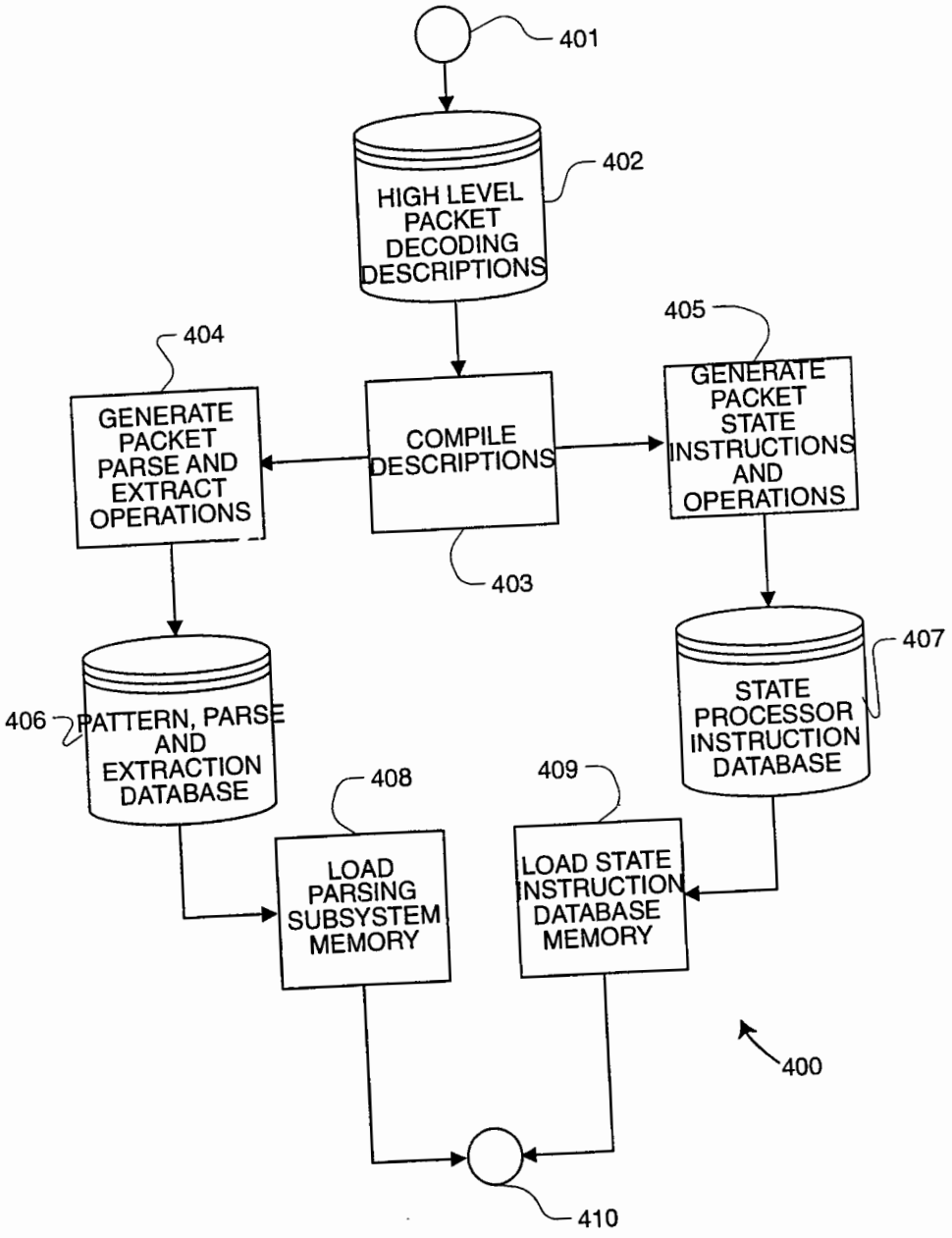


FIG. 4

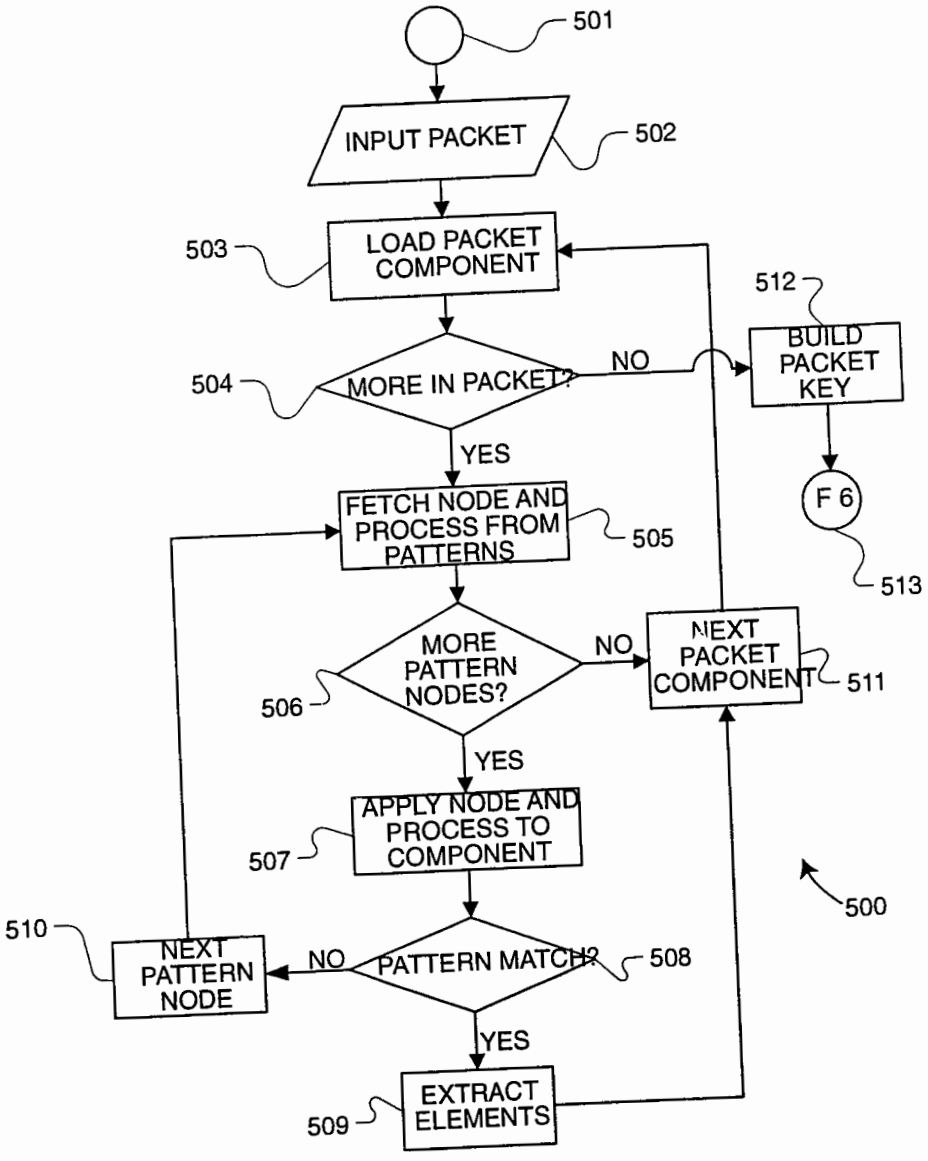


FIG. 5

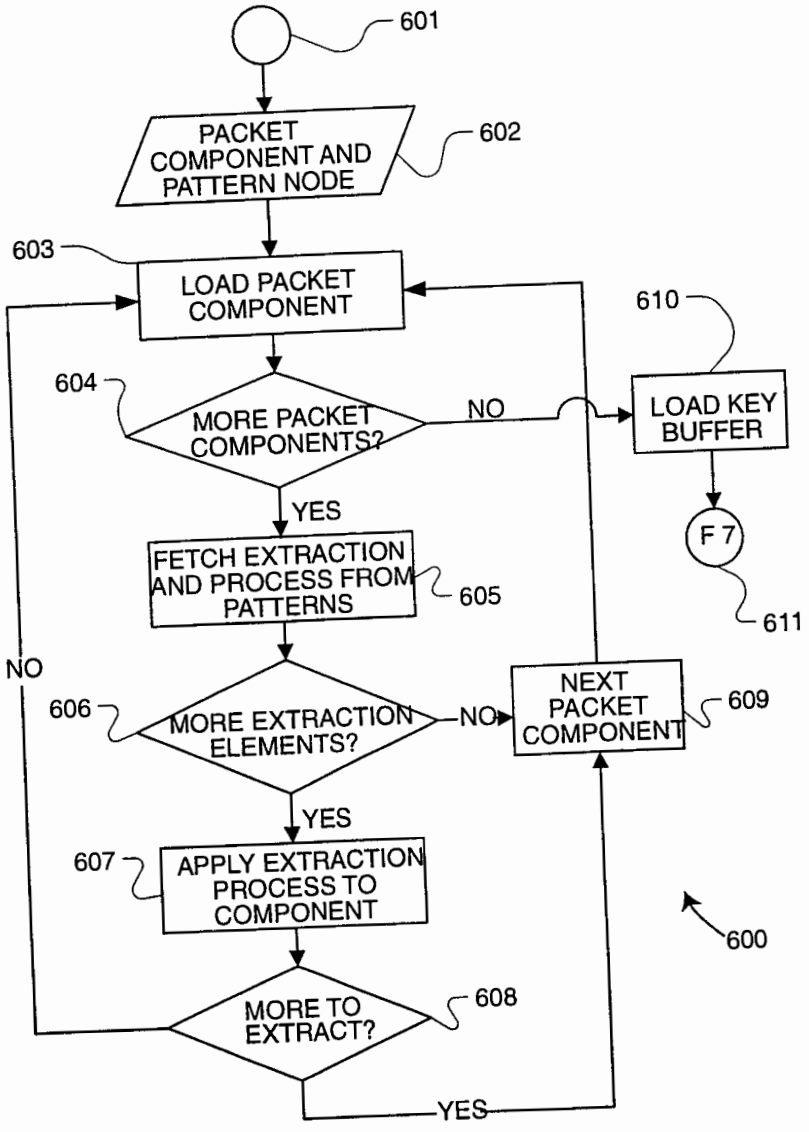


FIG. 6

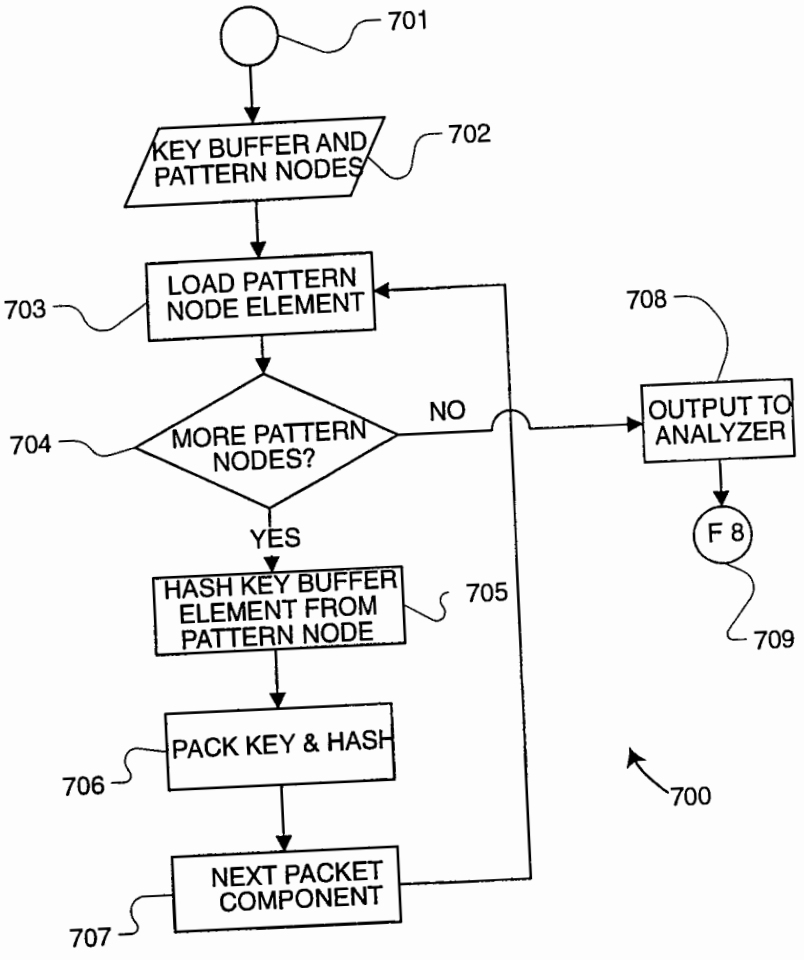


FIG. 7

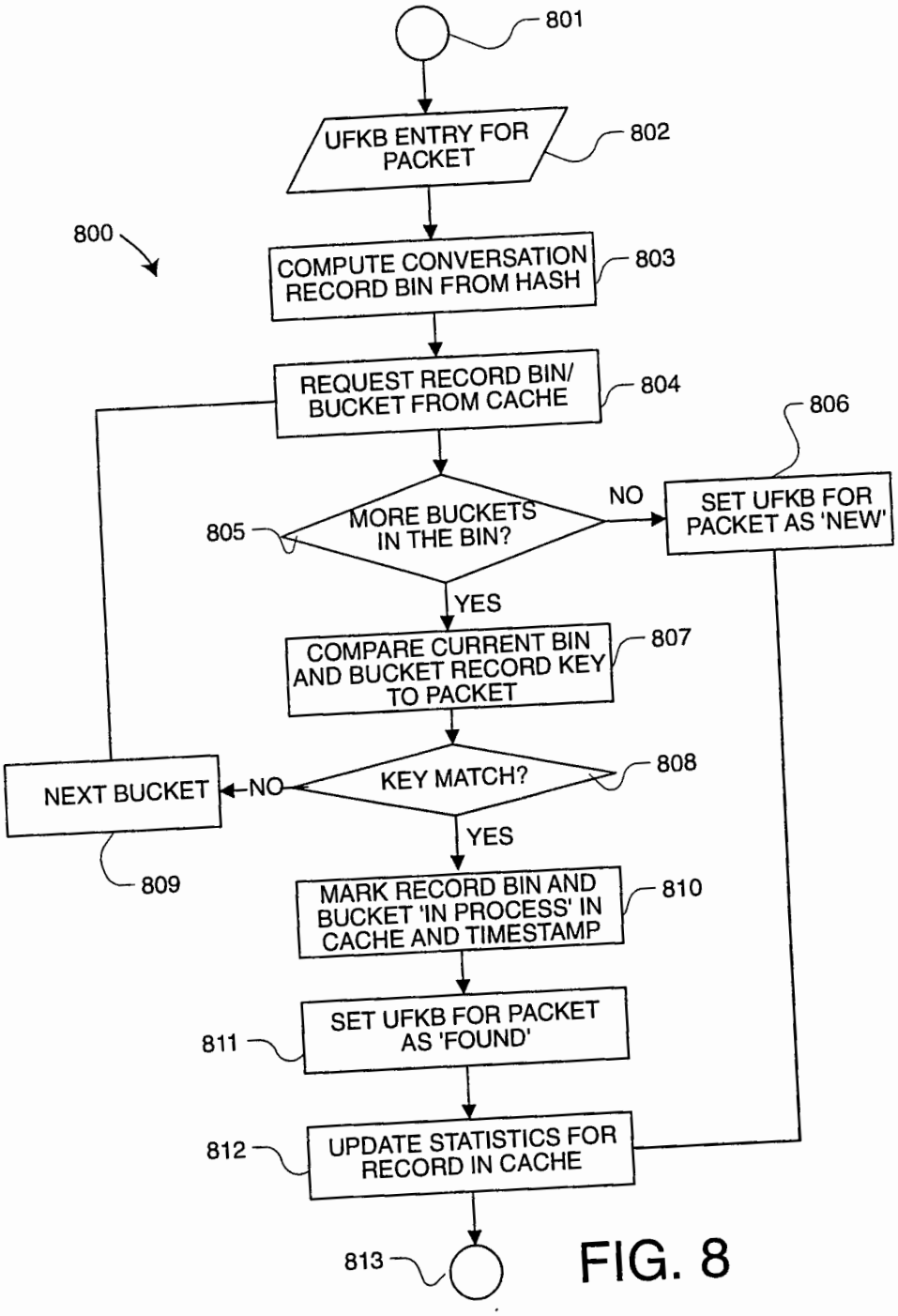


FIG. 8

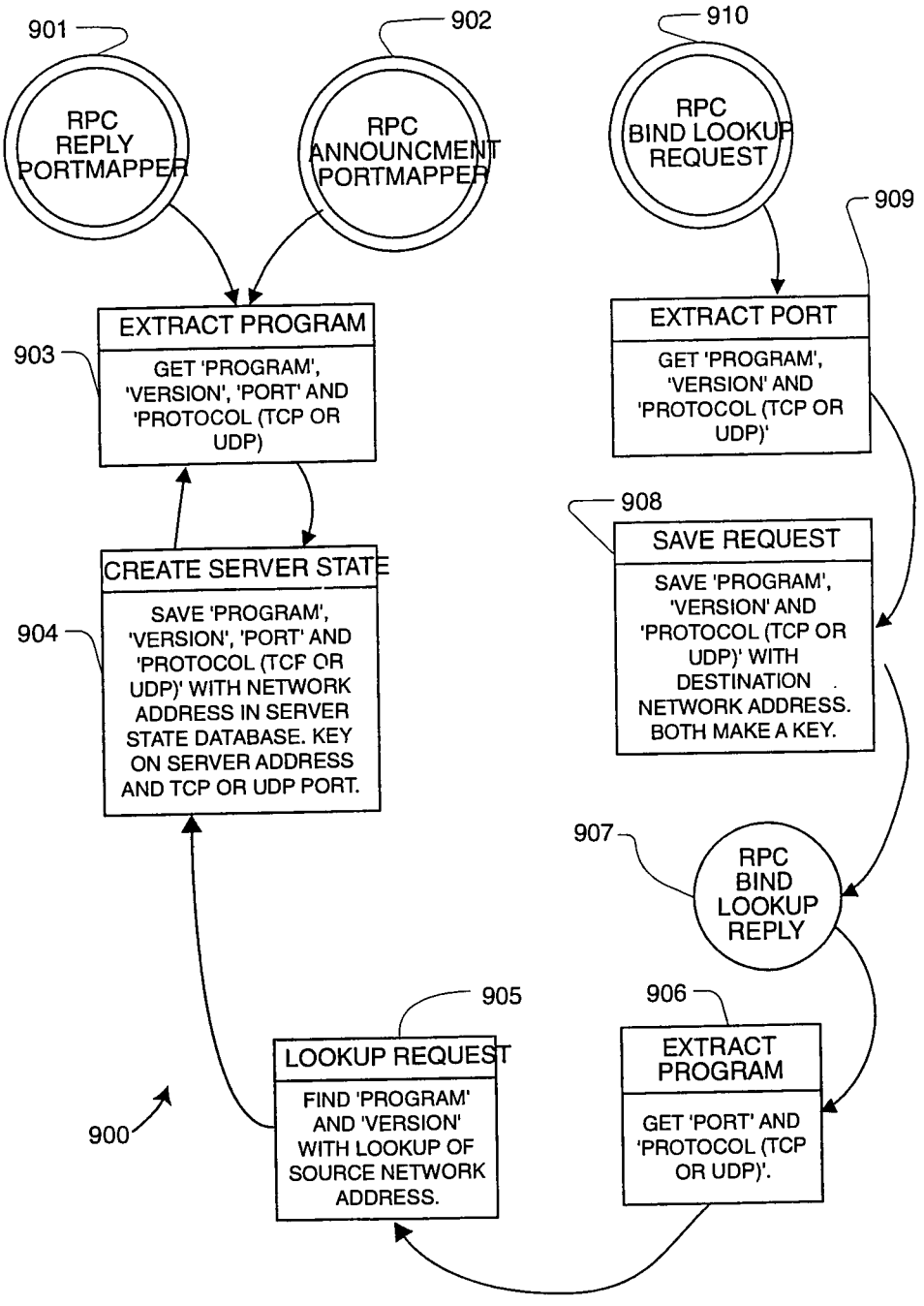


FIG. 9

PRINT OF DRAWINGS
AS ORIGINALLY FILED

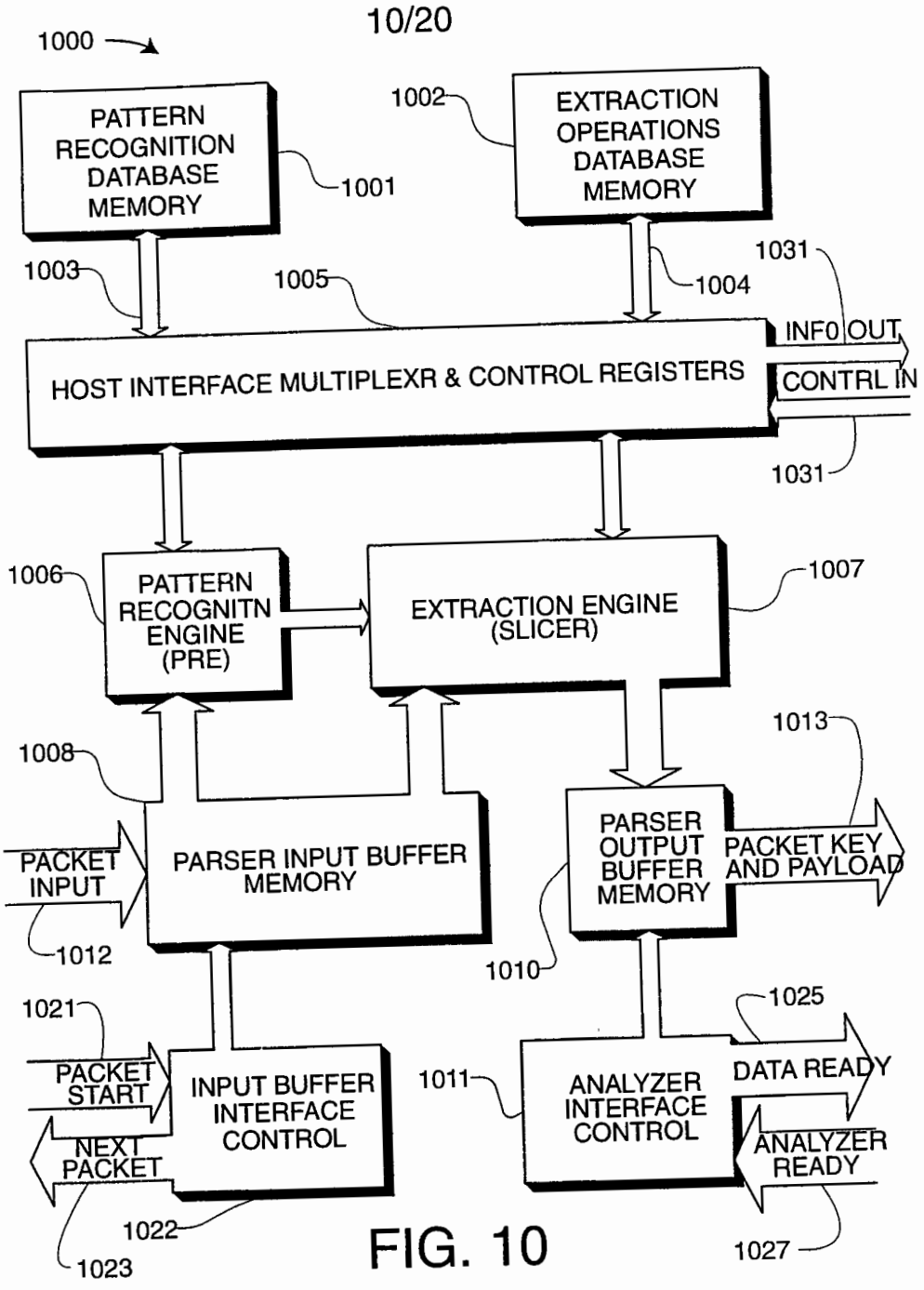


FIG. 10

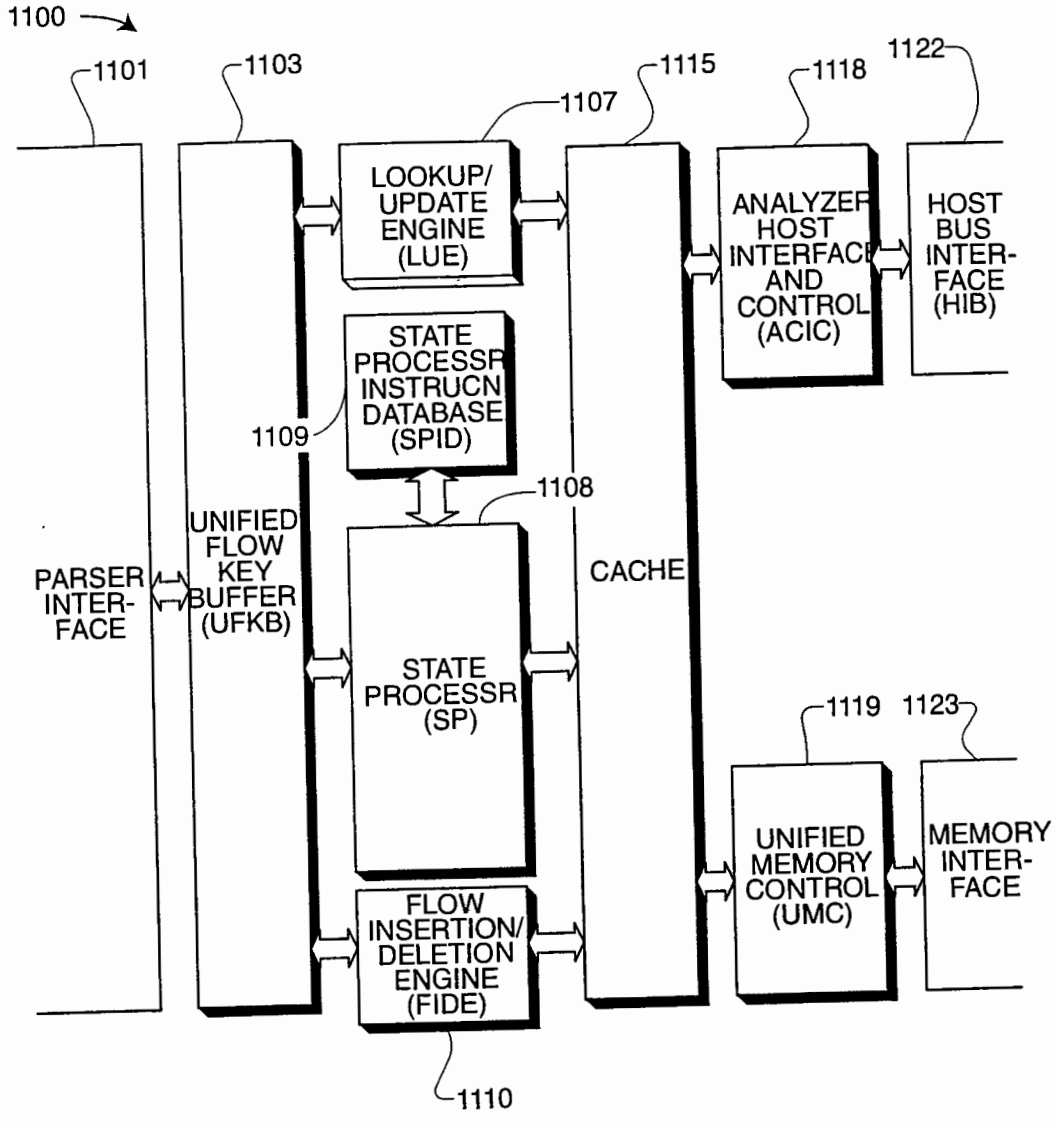


FIG. 11

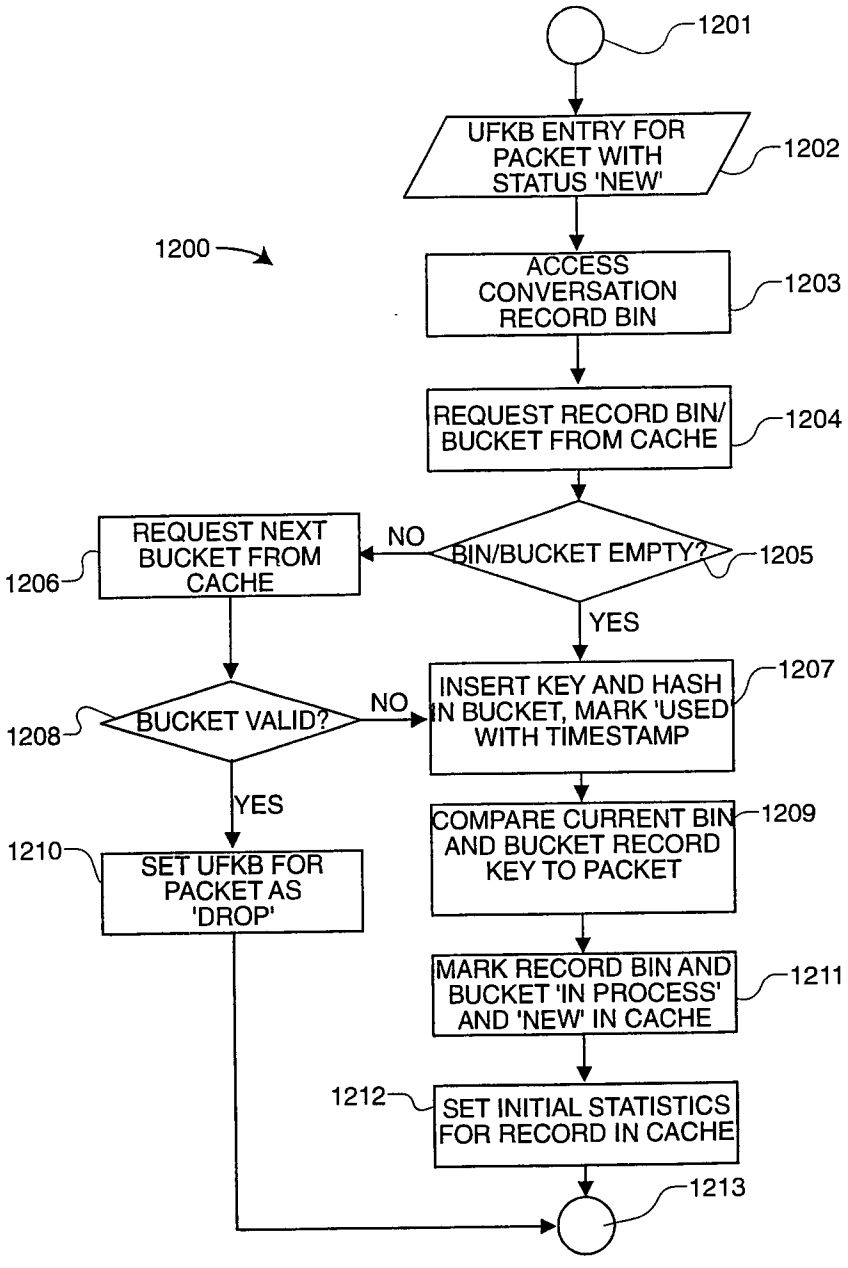

1200
1201
UFKB ENTRY FOR PACKET WITH STATUS 'NEW'
1202
ACCESS CONVERSATION RECORD BIN
1203
REQUEST RECORD BIN/ BUCKET FROM CACHE
1204
BIN/BUCKET EMPTY?
1205
NO
REQUEST NEXT BUCKET FROM CACHE
1206
YES
BUCKET VALID?
1208
NO
INSERT KEY AND HASH IN BUCKET, MARK 'USED' WITH TIMESTAMP
1207
YES
SET UFKB FOR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

FIG. 12

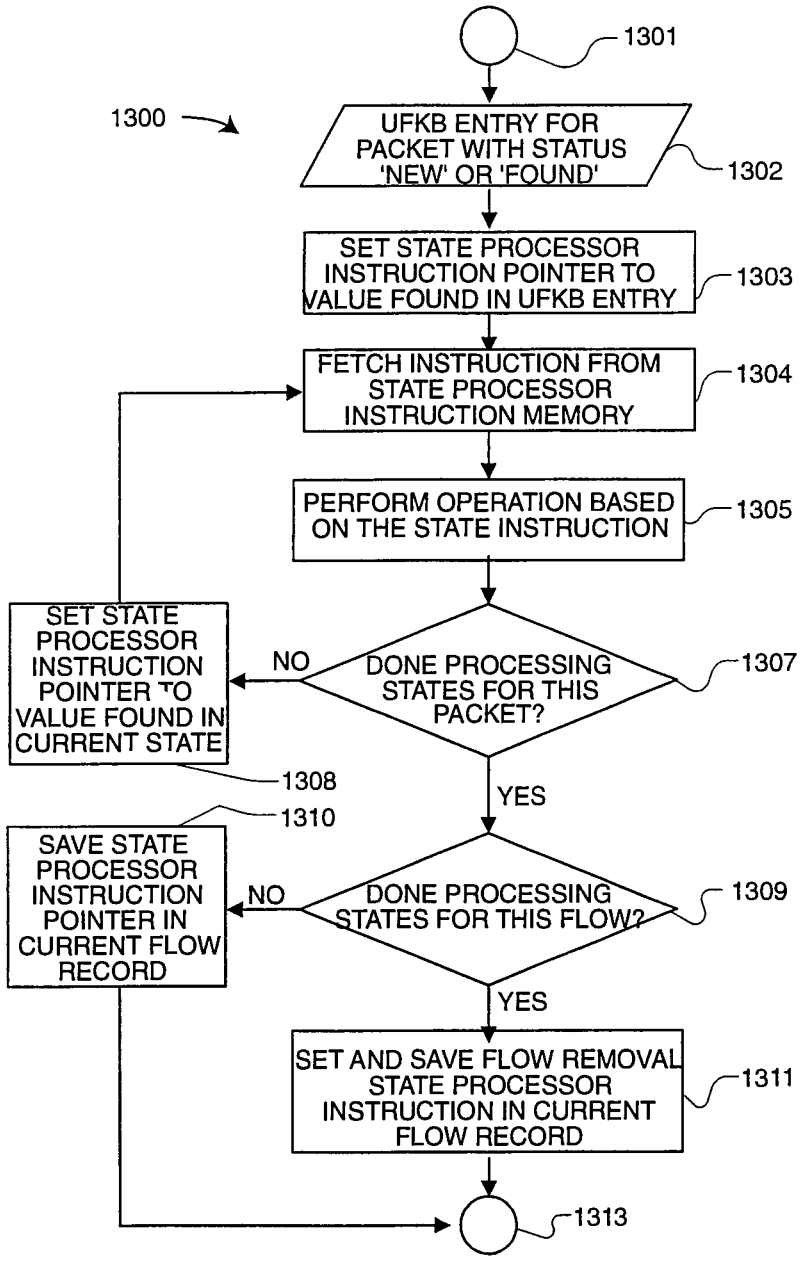


FIG. 13

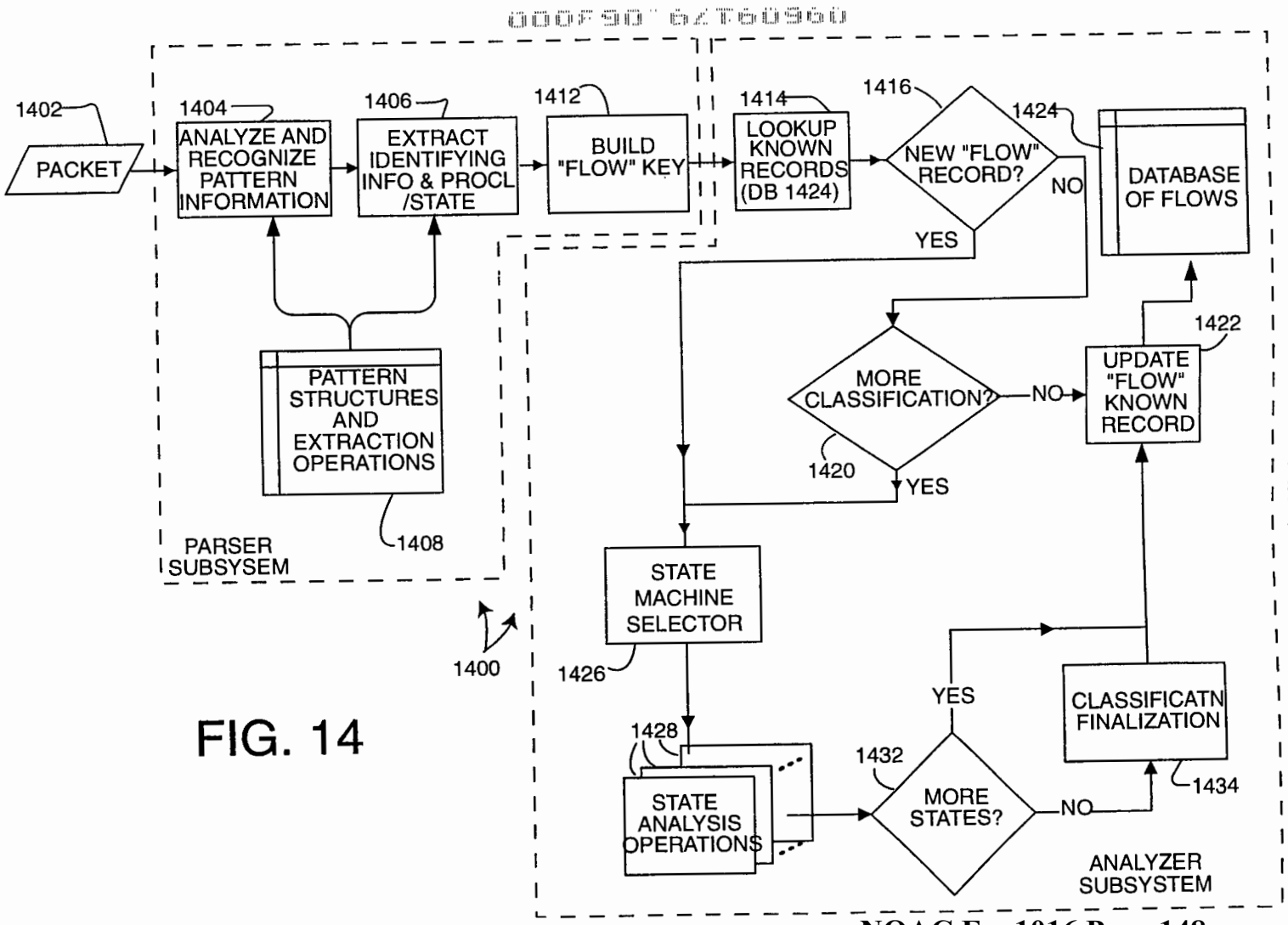


FIG. 14

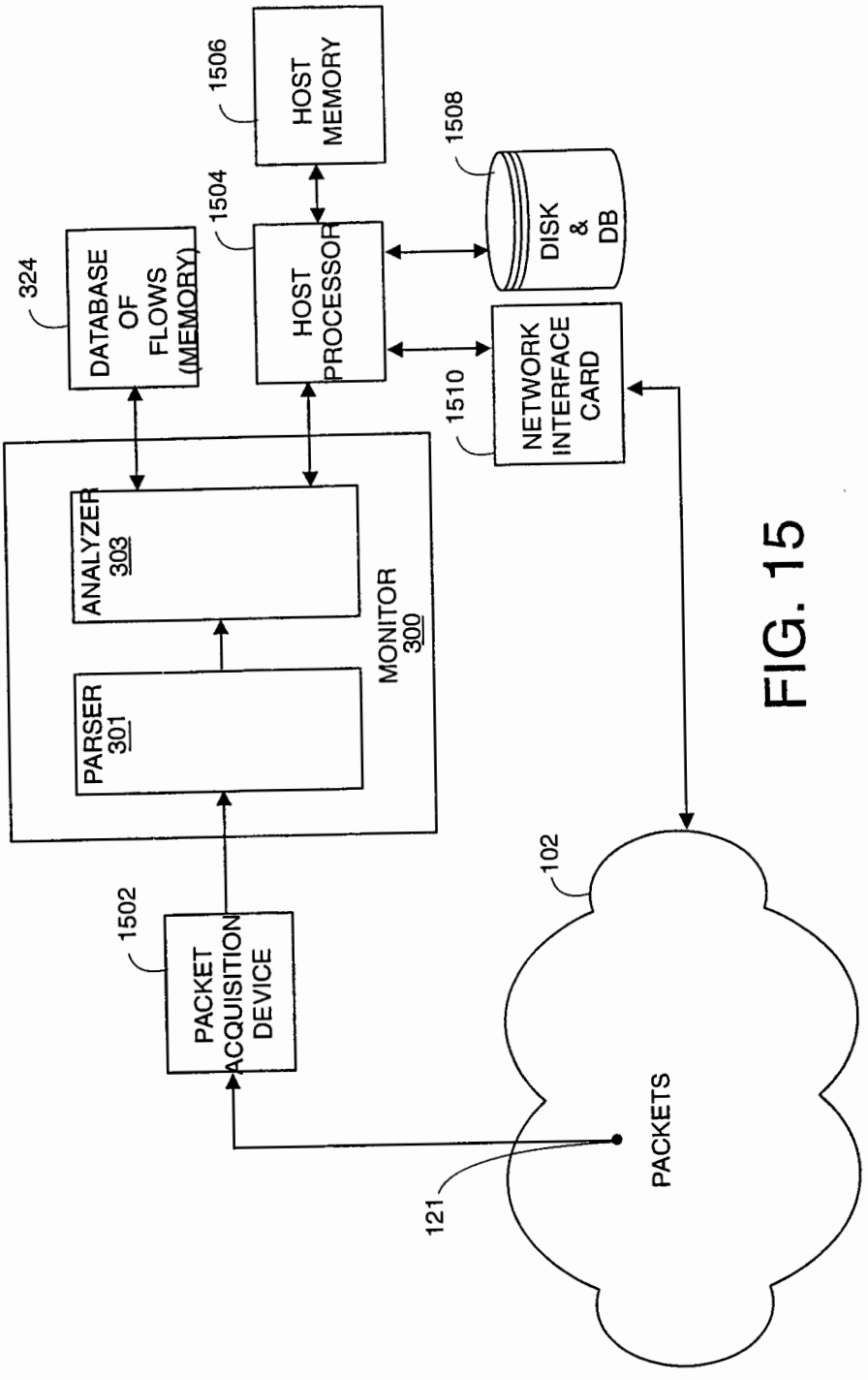


FIG. 15

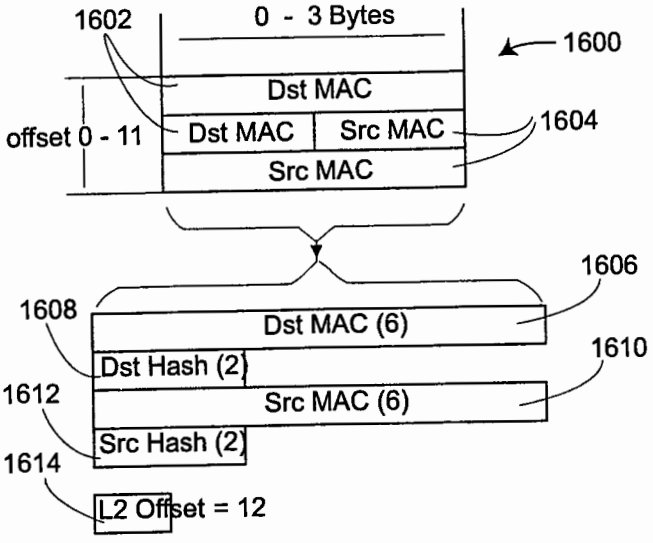


FIG. 16

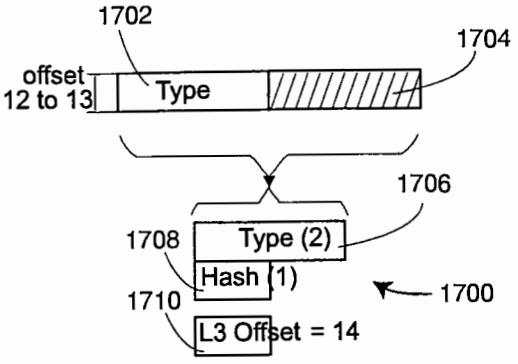


FIG. 17A

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

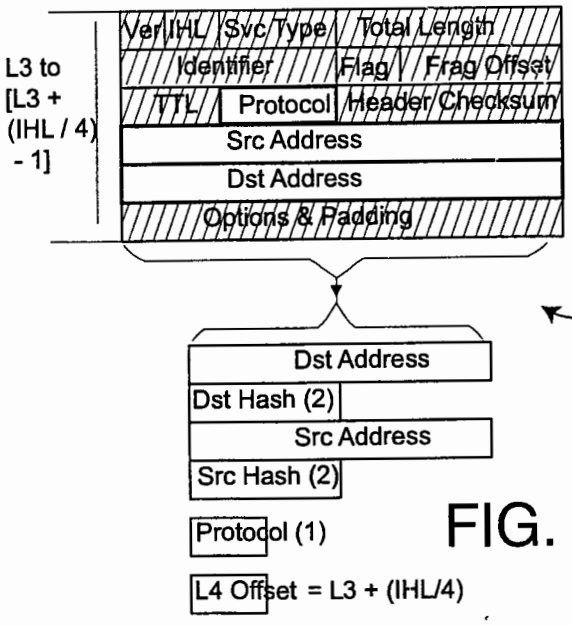


FIG. 17B

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

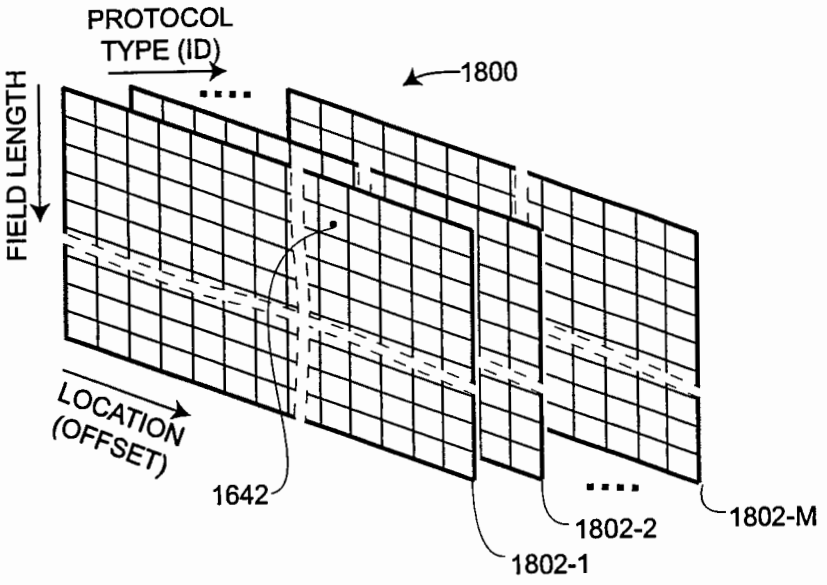


FIG. 18A

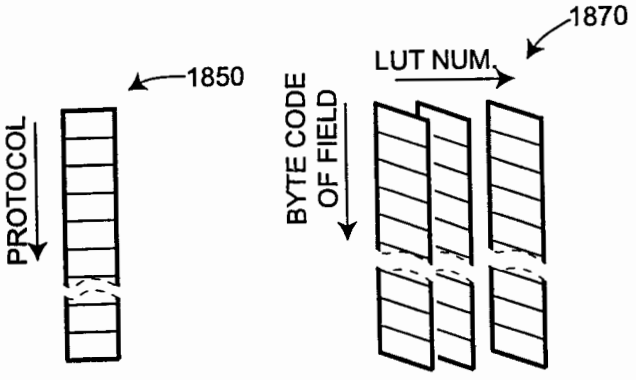


FIG. 18B

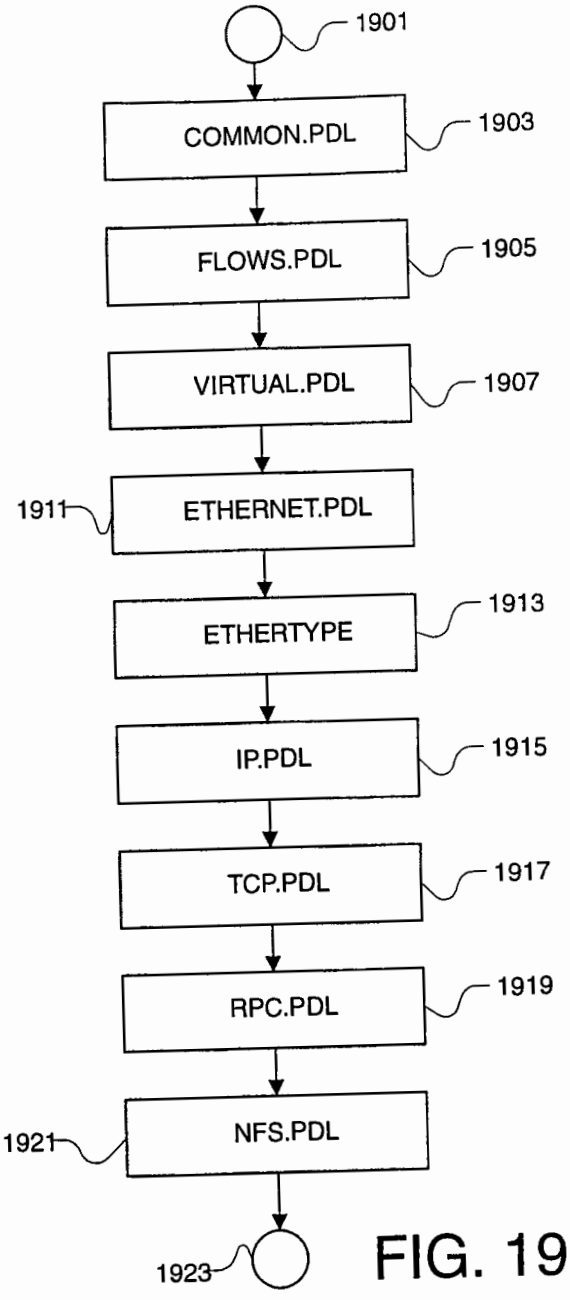


FIG. 19

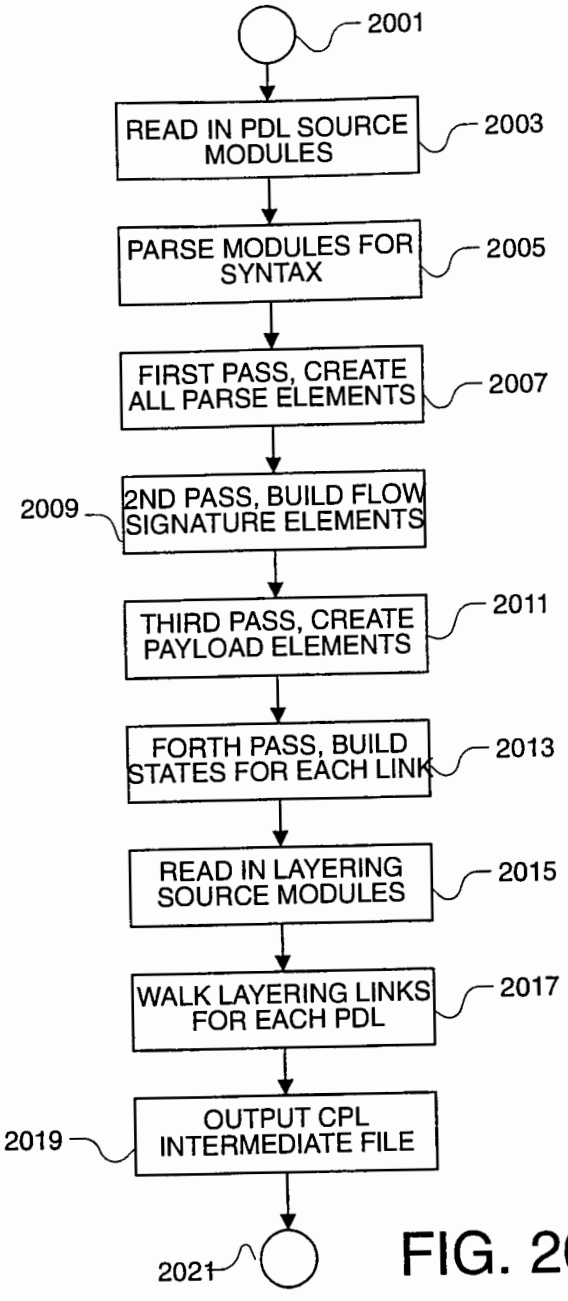


FIG. 20

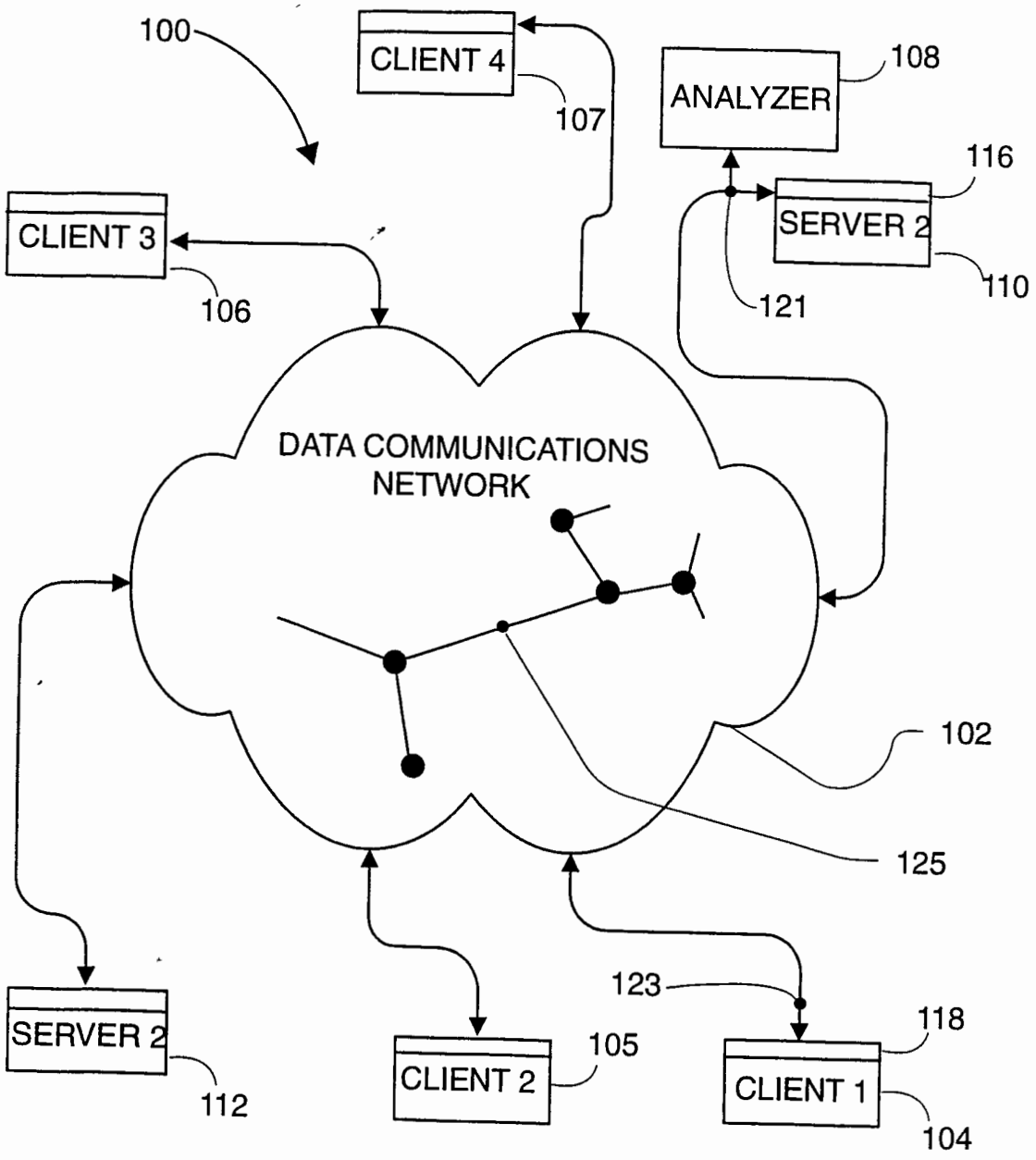


FIG. 1

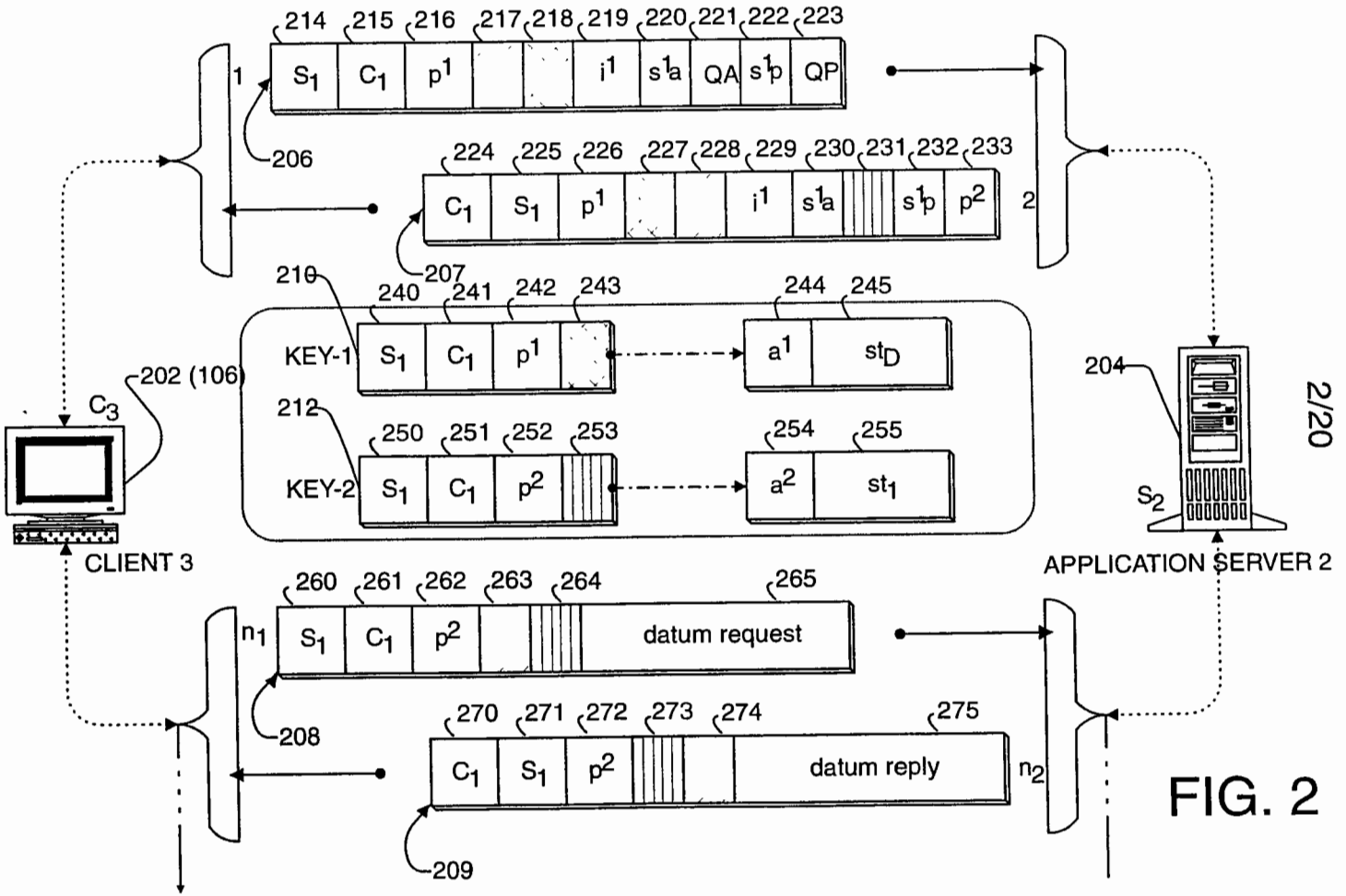


FIG. 2

2/20

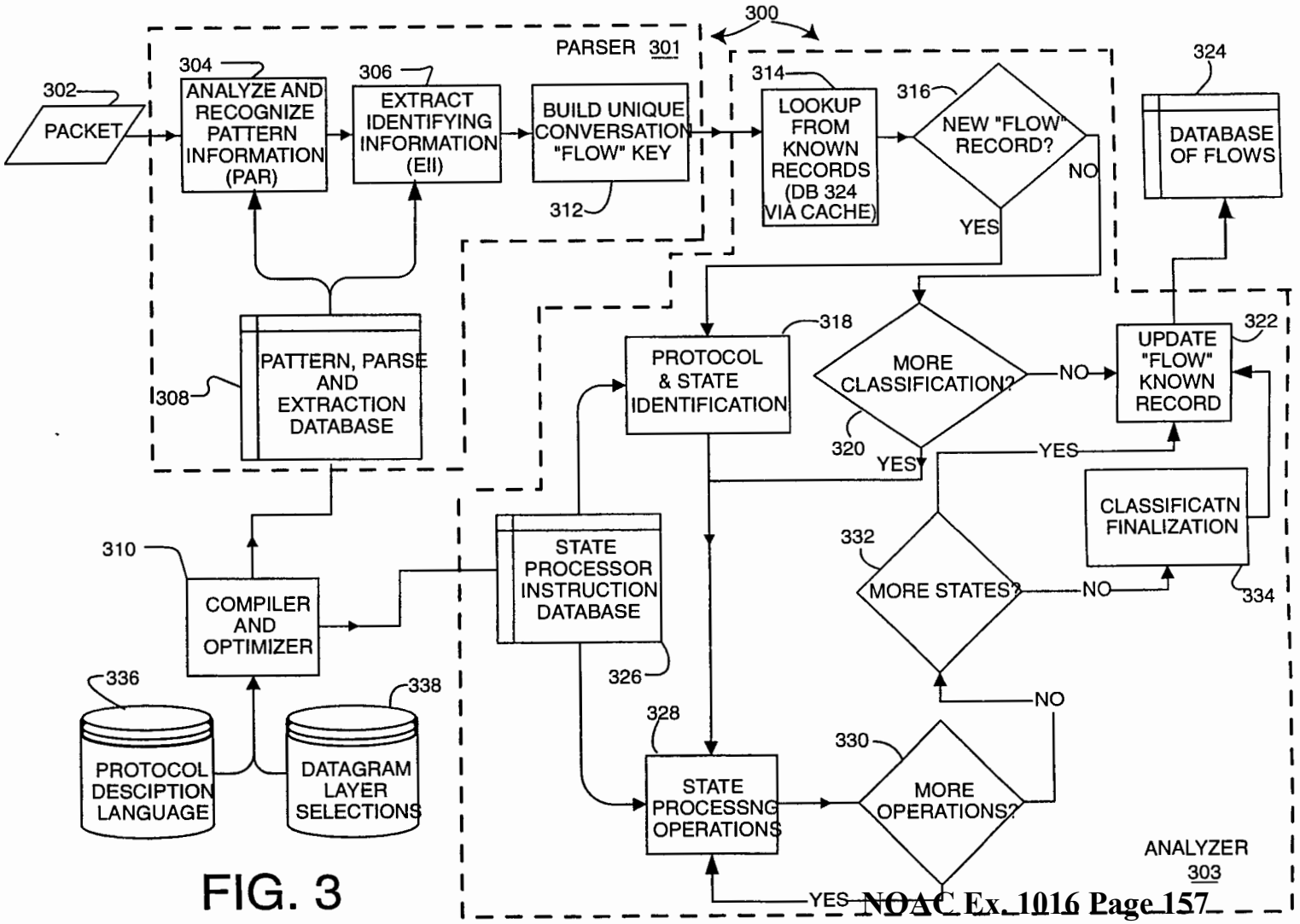


FIG. 3

3/20

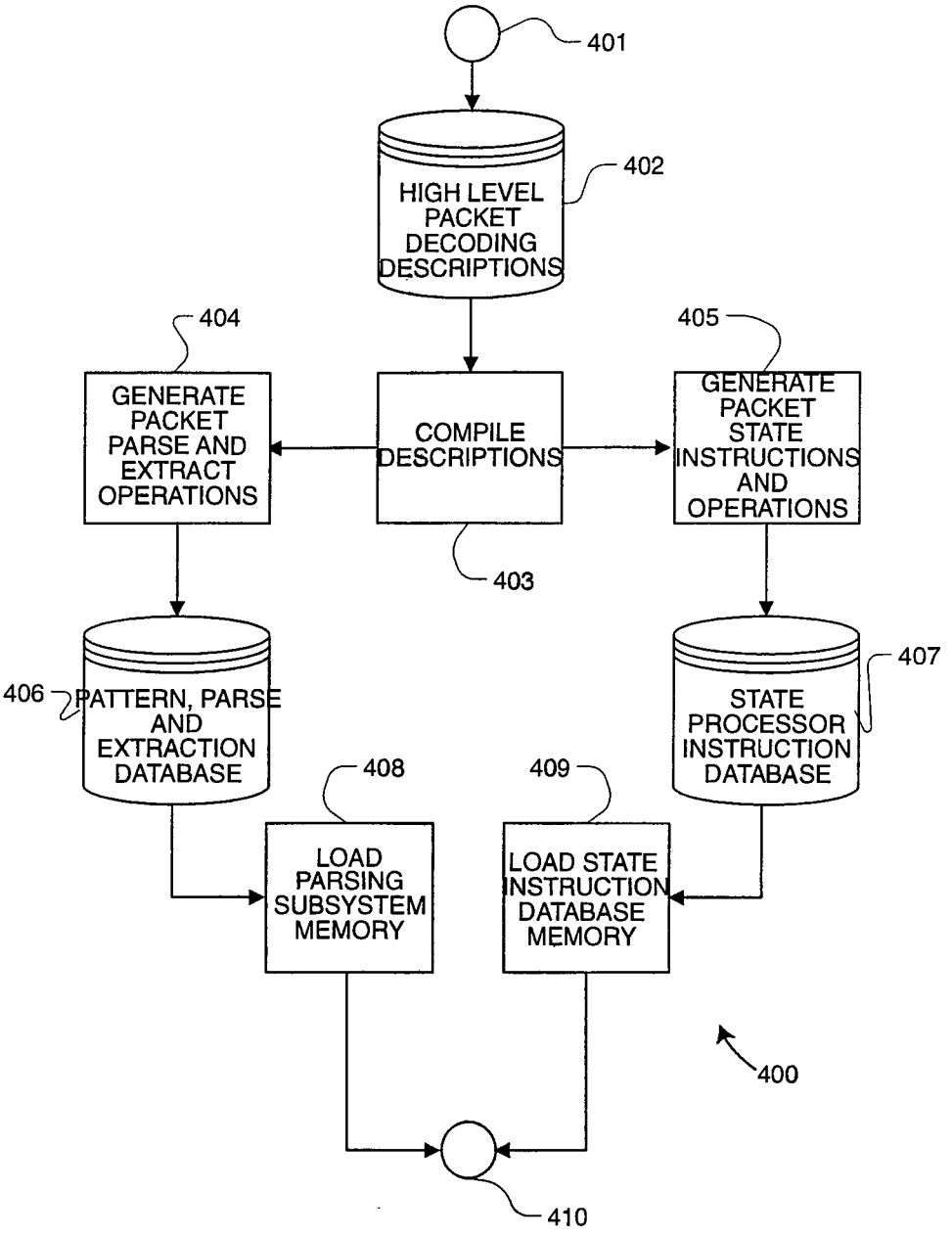


FIG. 4

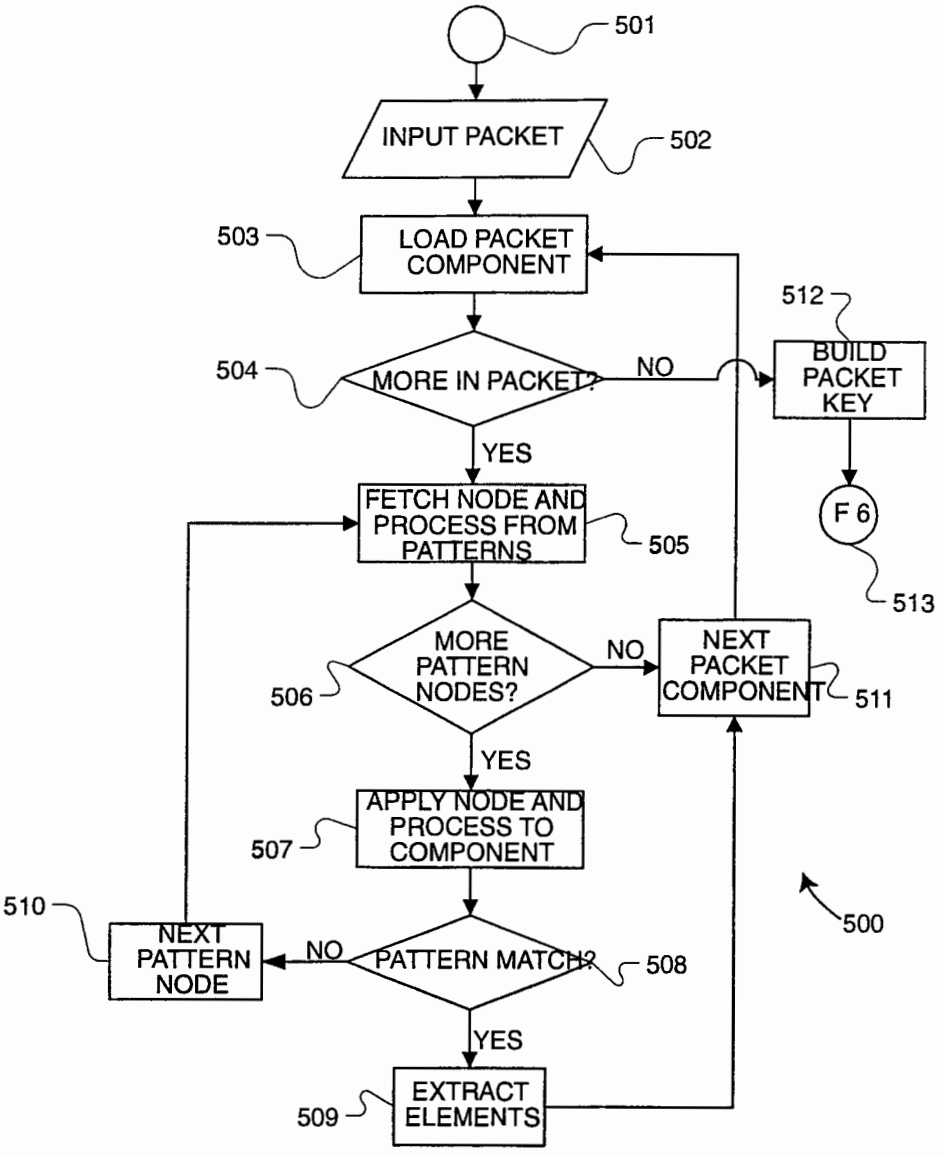


FIG. 5

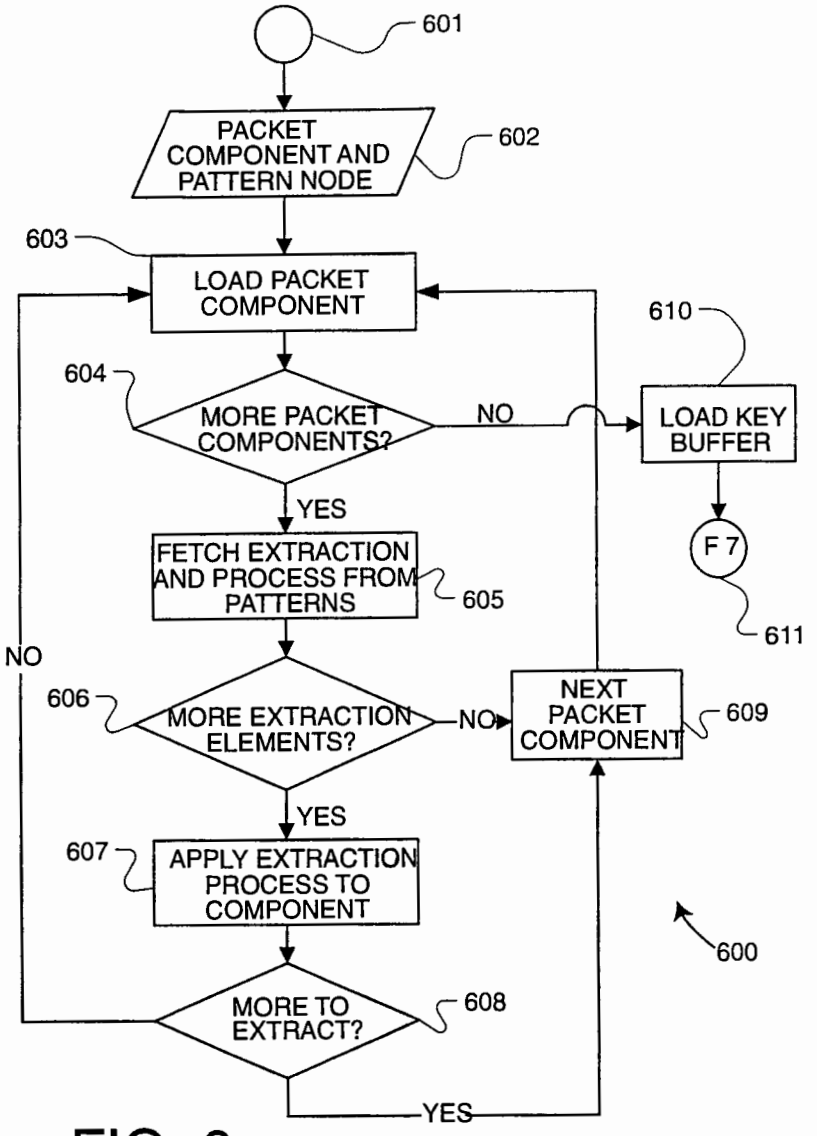


FIG. 6

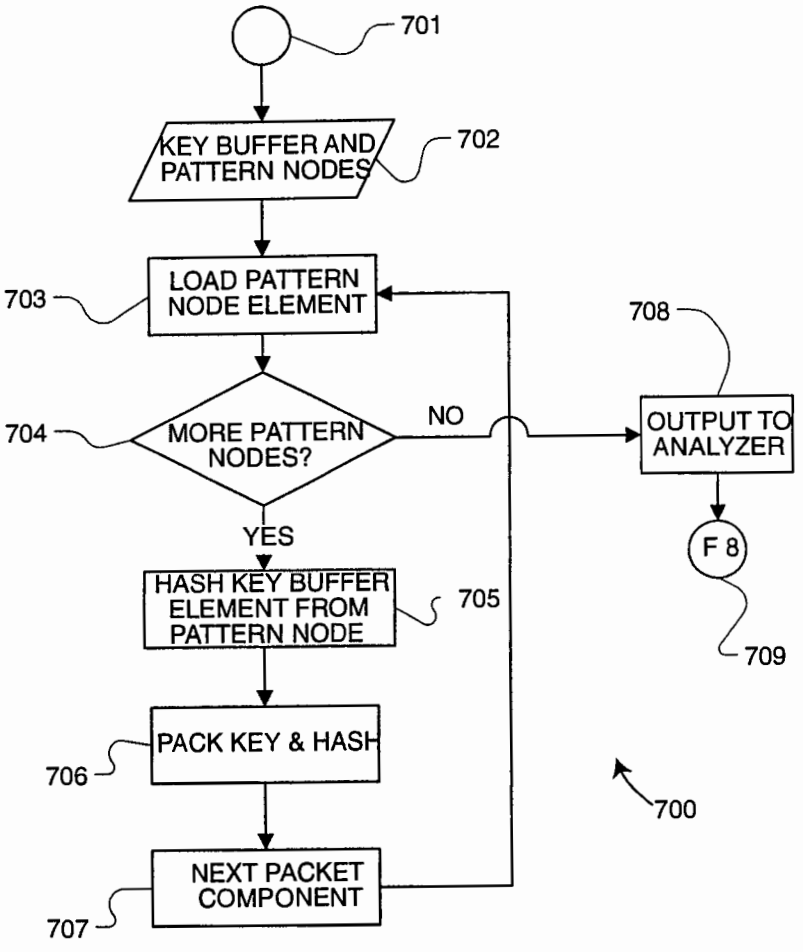


FIG. 7

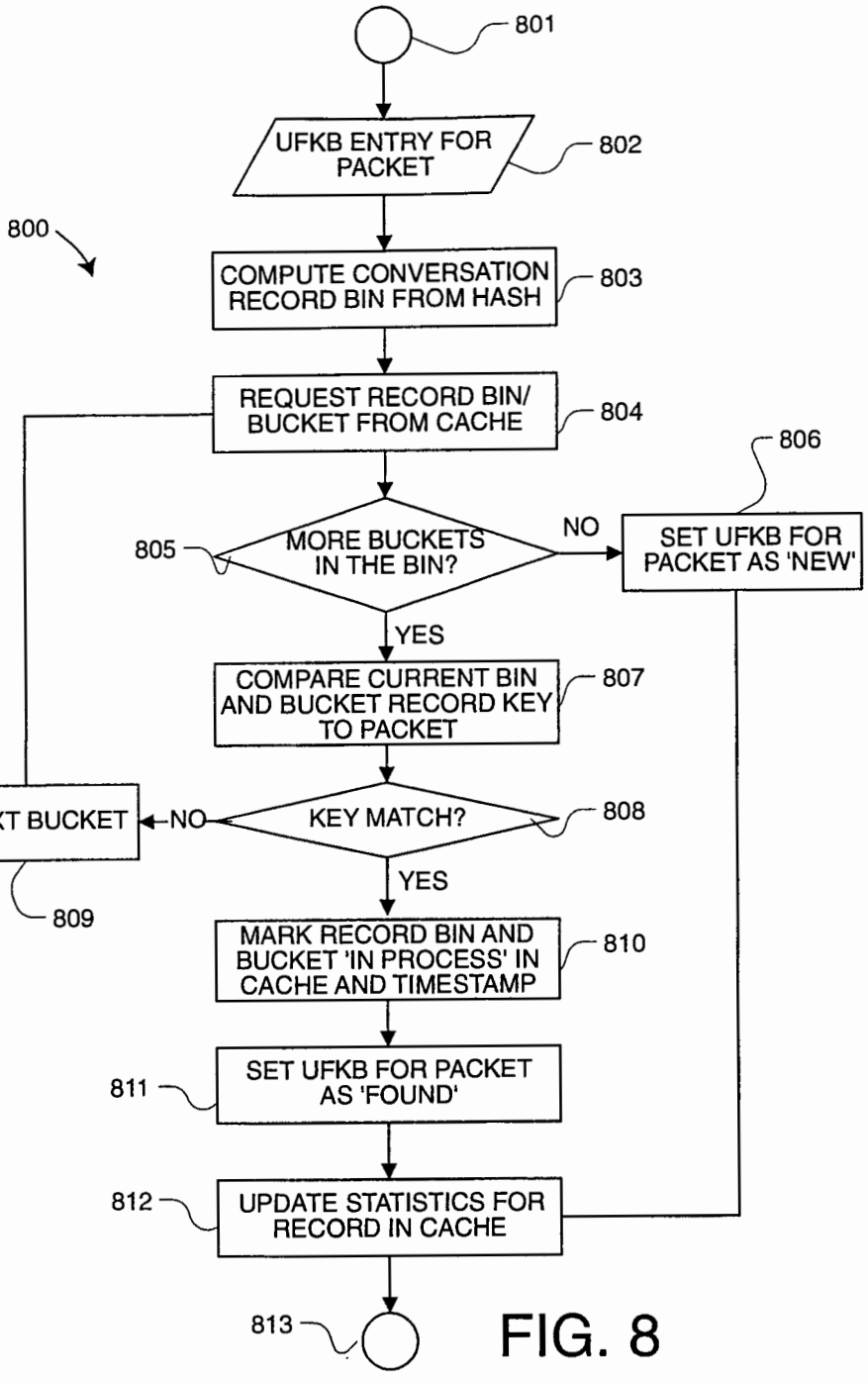


FIG. 8

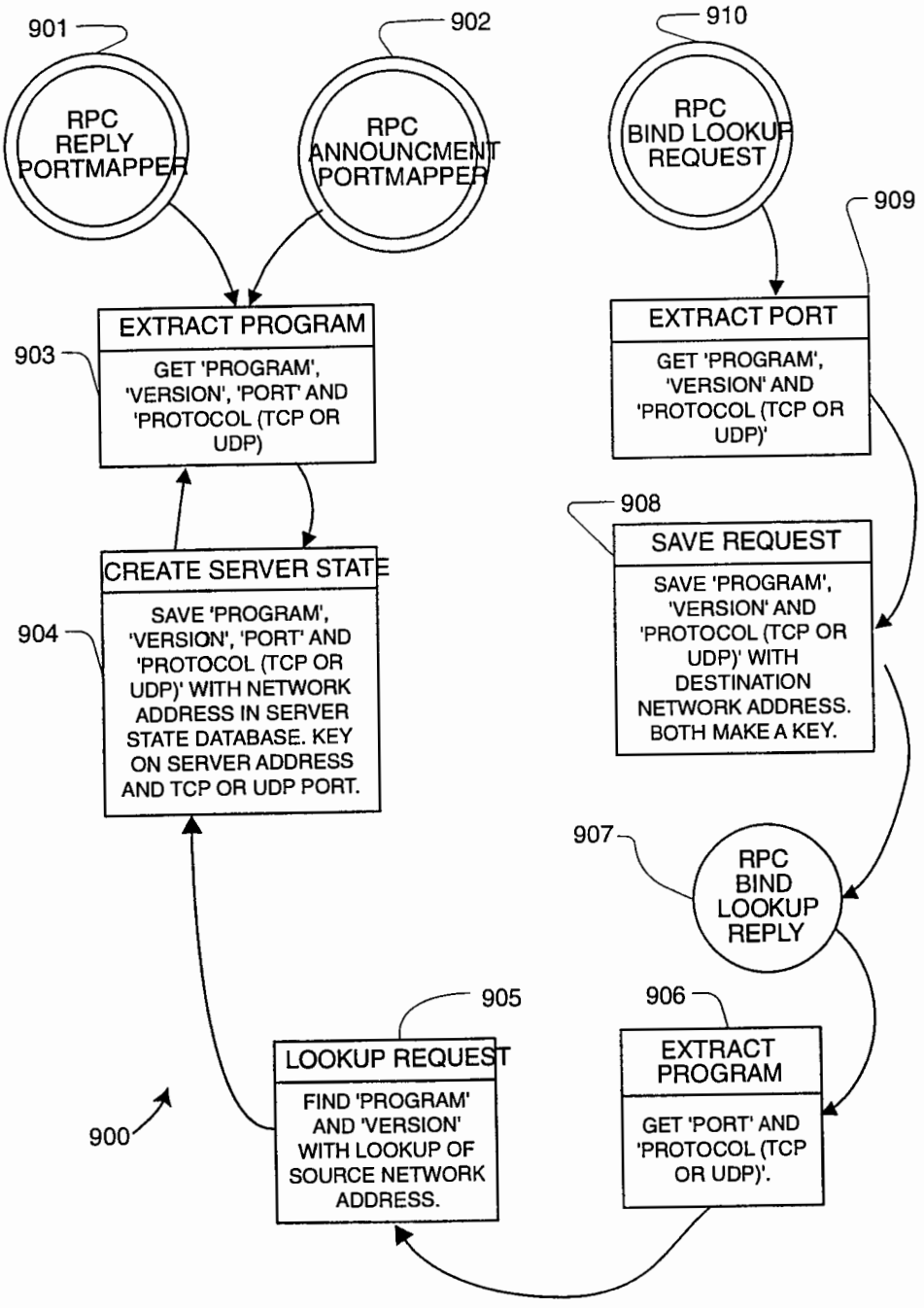


FIG. 9

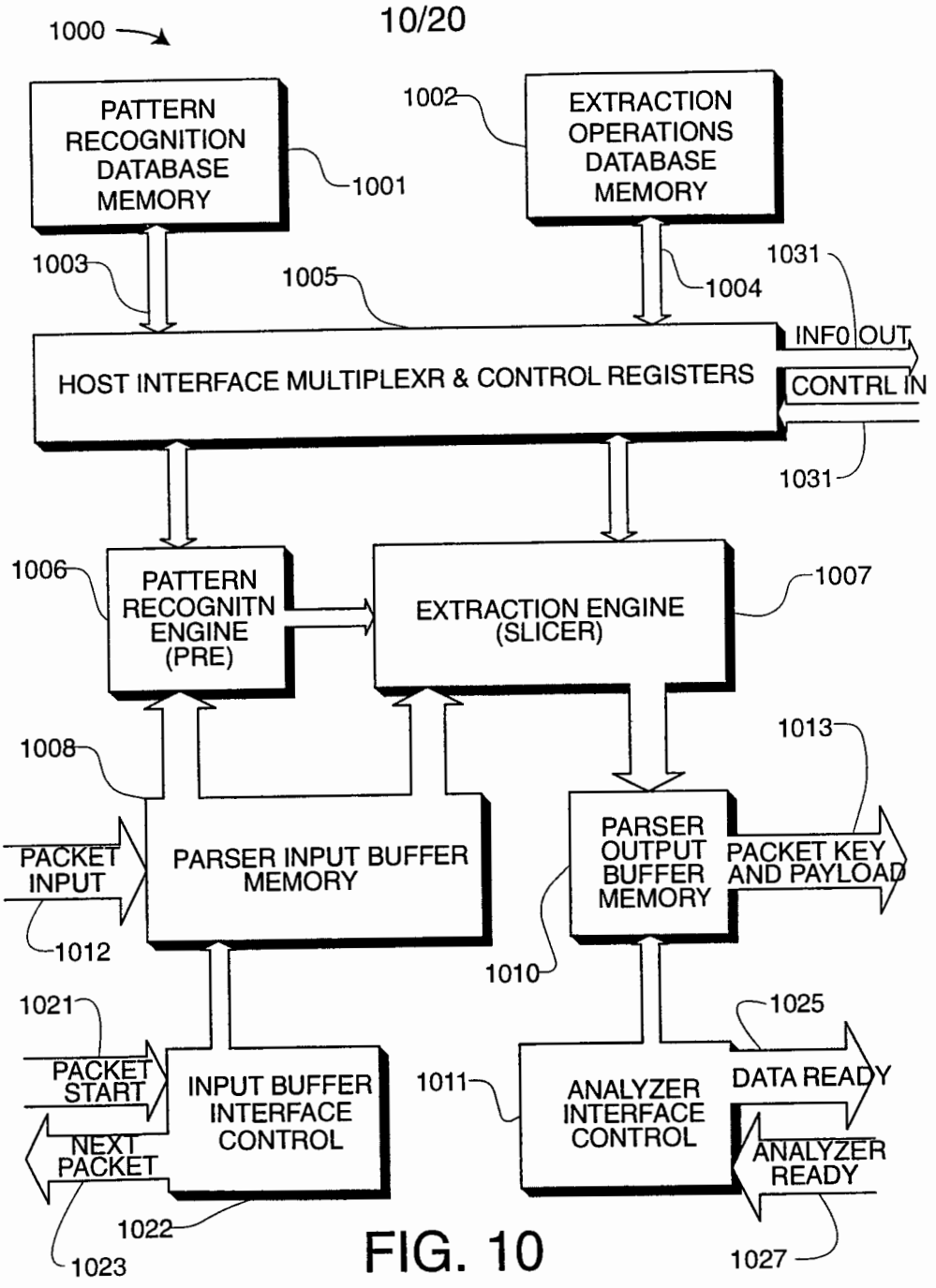


FIG. 10

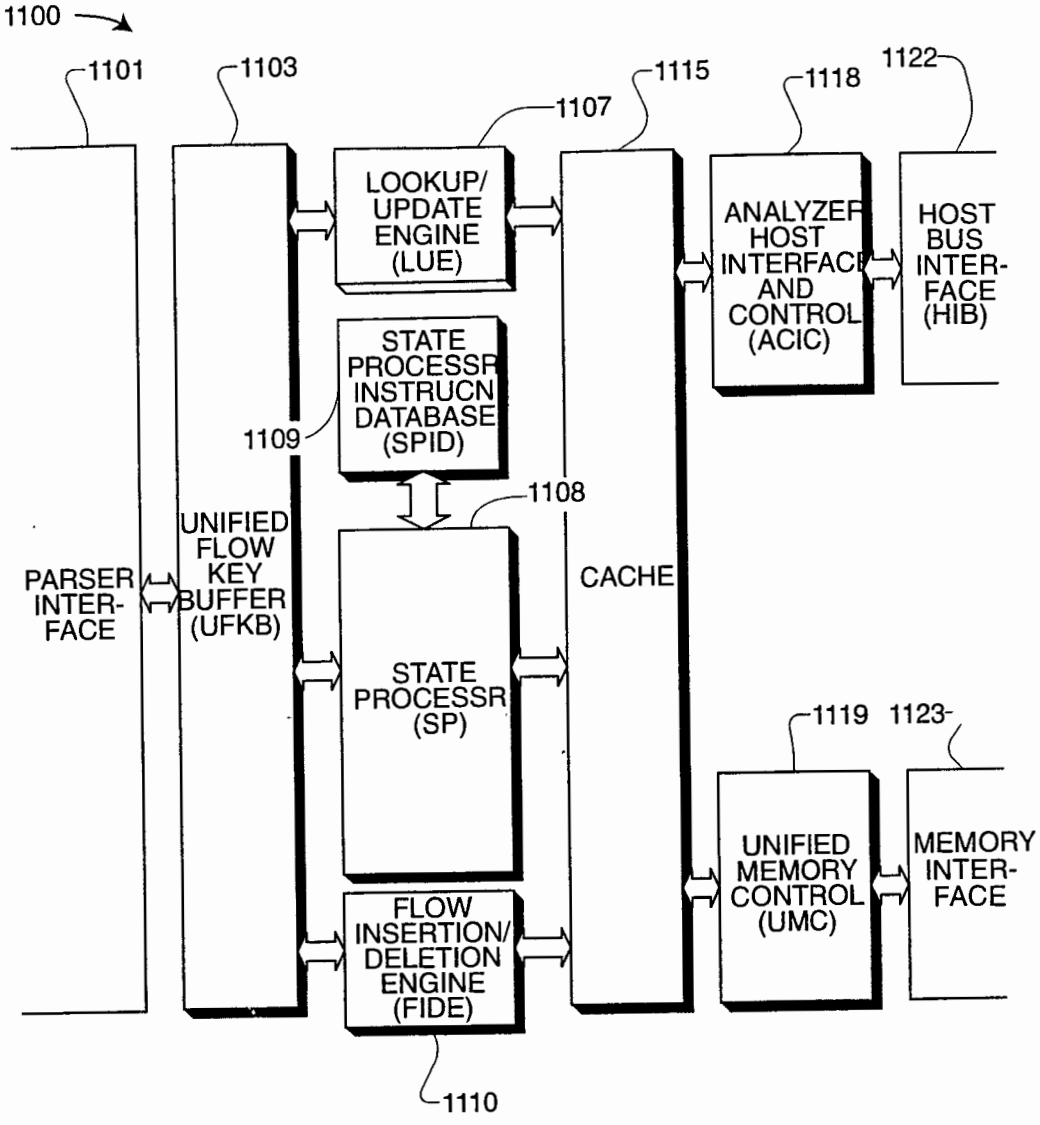


FIG. 11

PRINT OF DRAWINGS
AS ORIGINALLY FILE

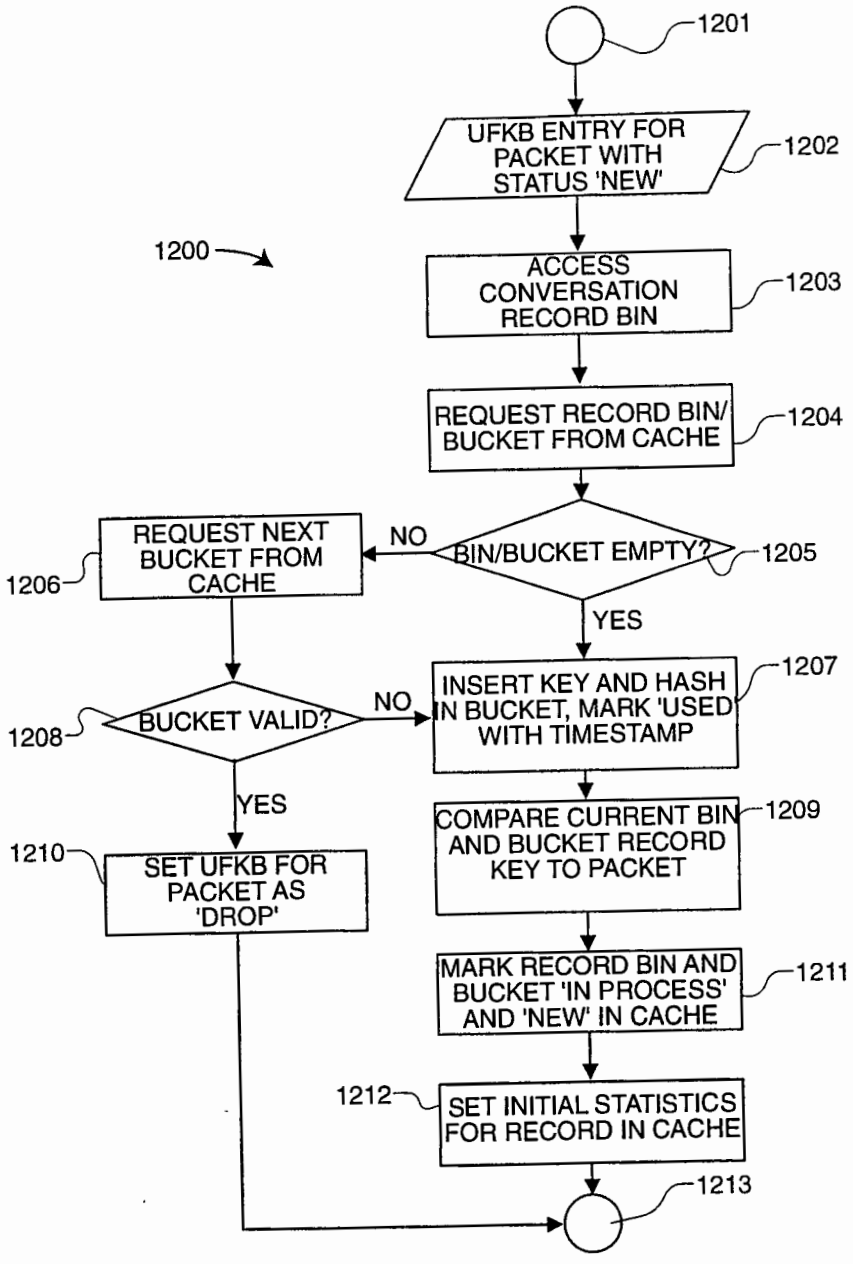


FIG. 12

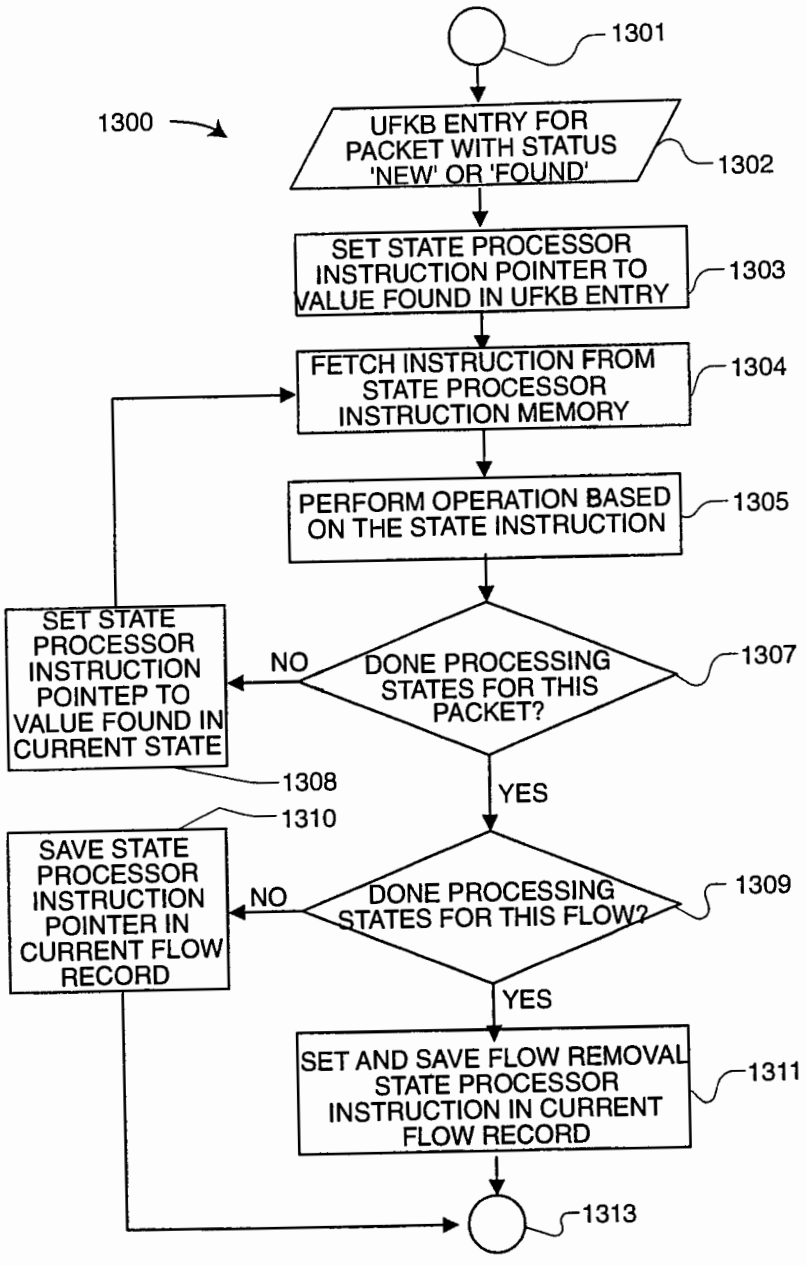


FIG. 13

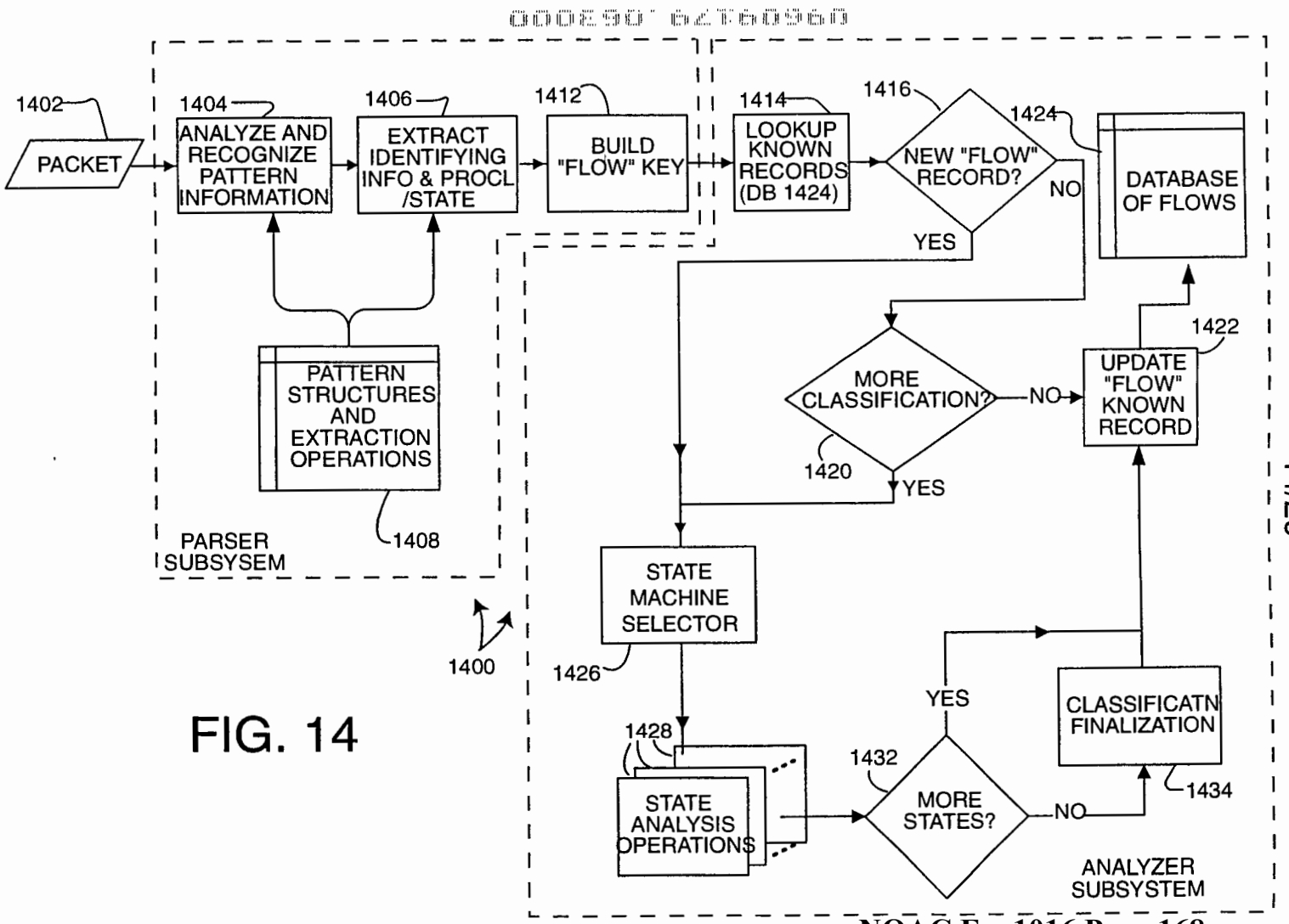


FIG. 14

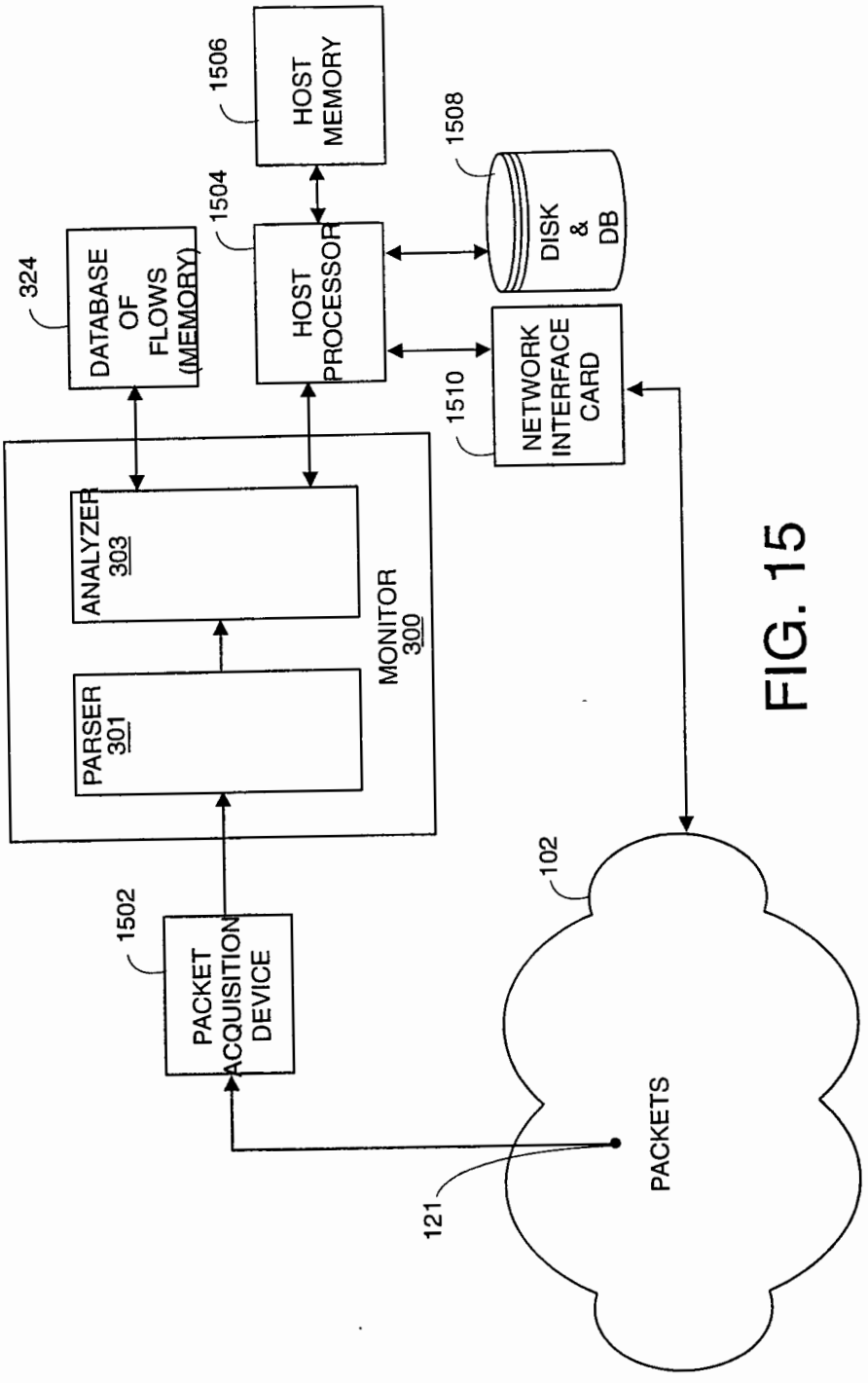


FIG. 15

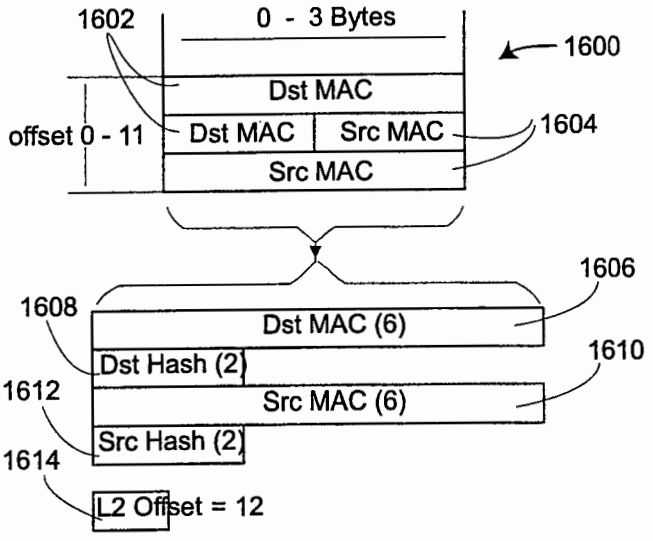


FIG. 16

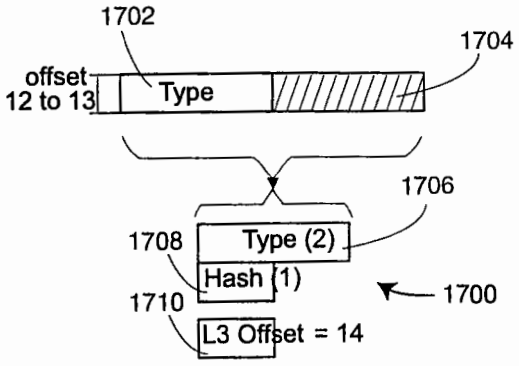


FIG. 17A

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

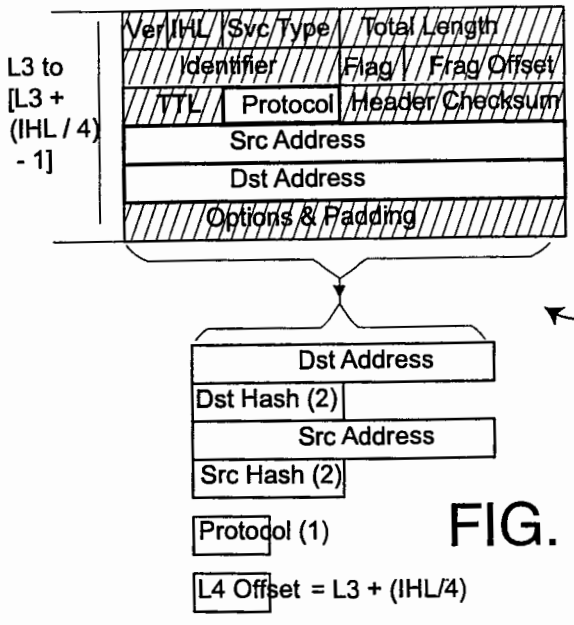


FIG. 17B

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

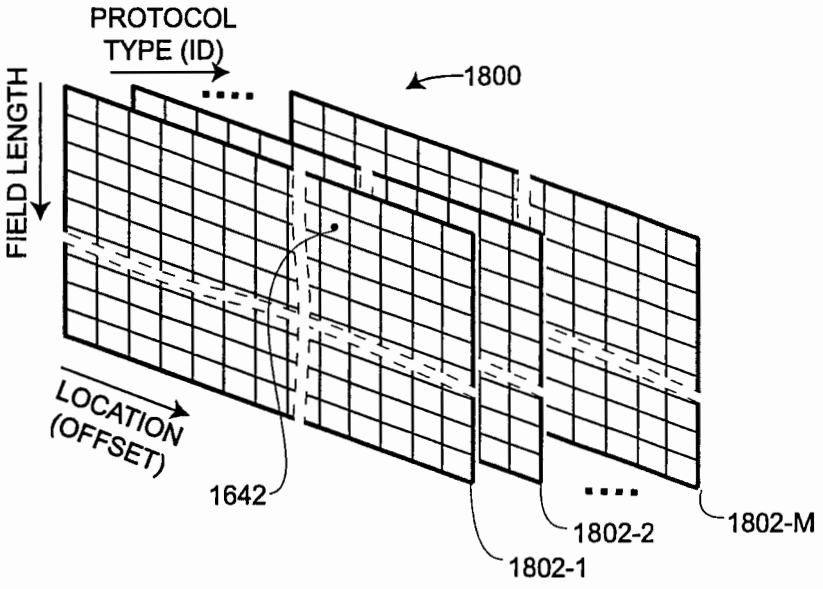


FIG. 18A

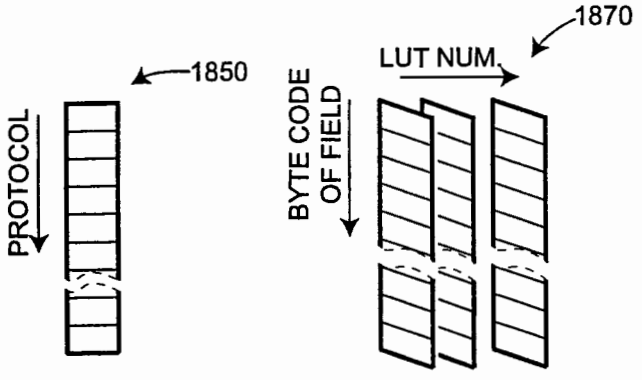


FIG. 18B

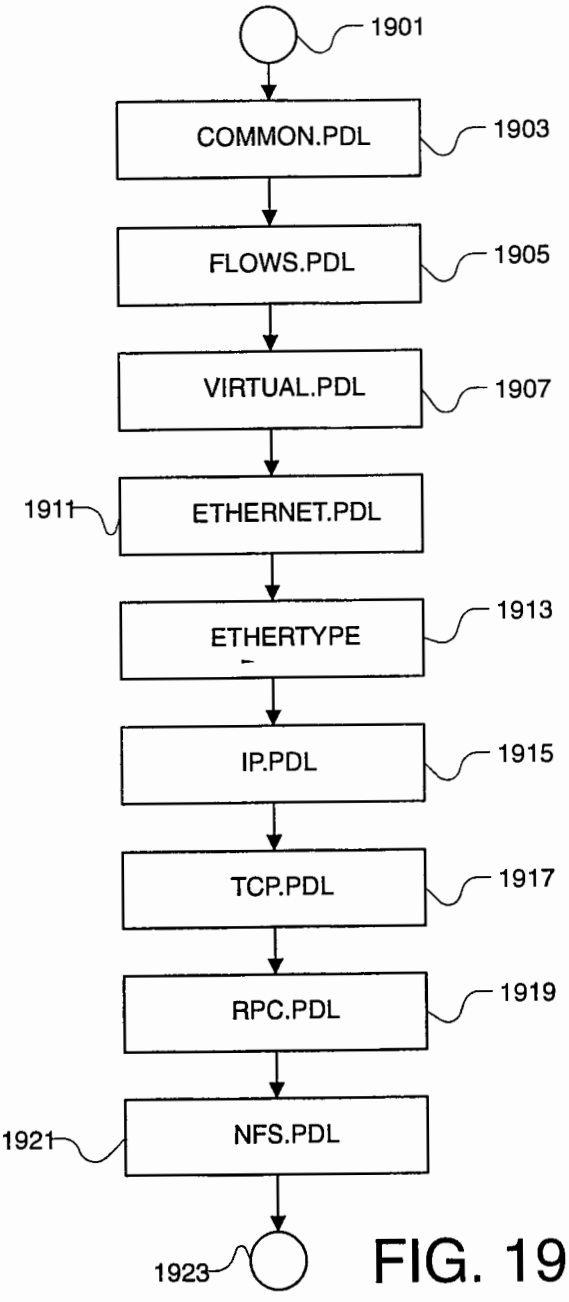


FIG. 19

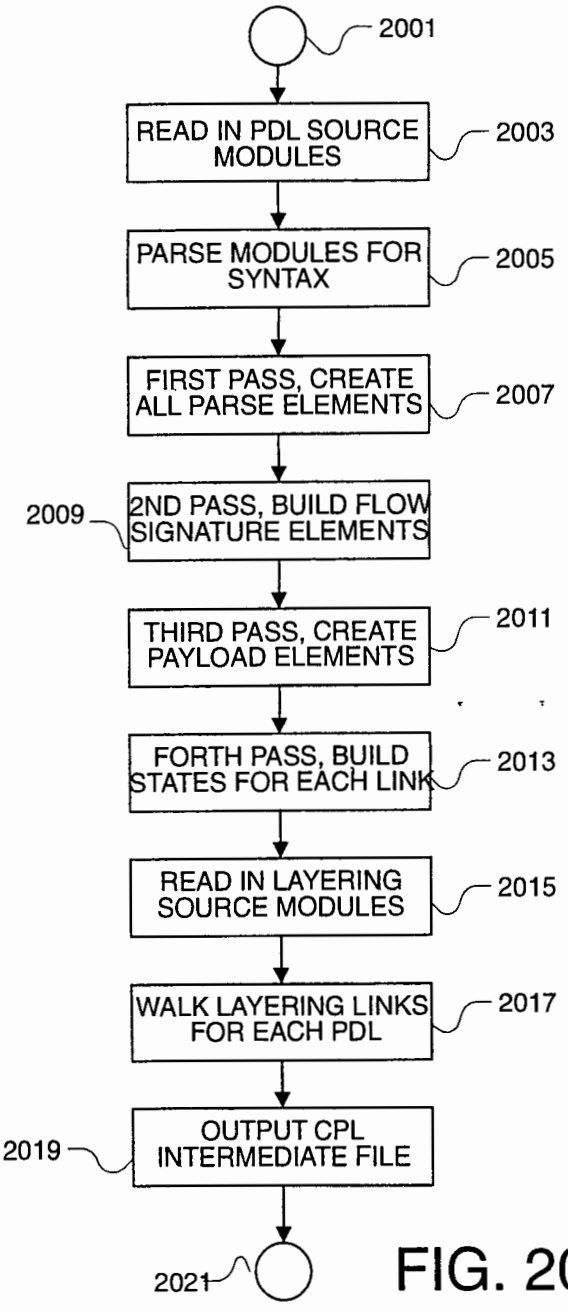


FIG. 20

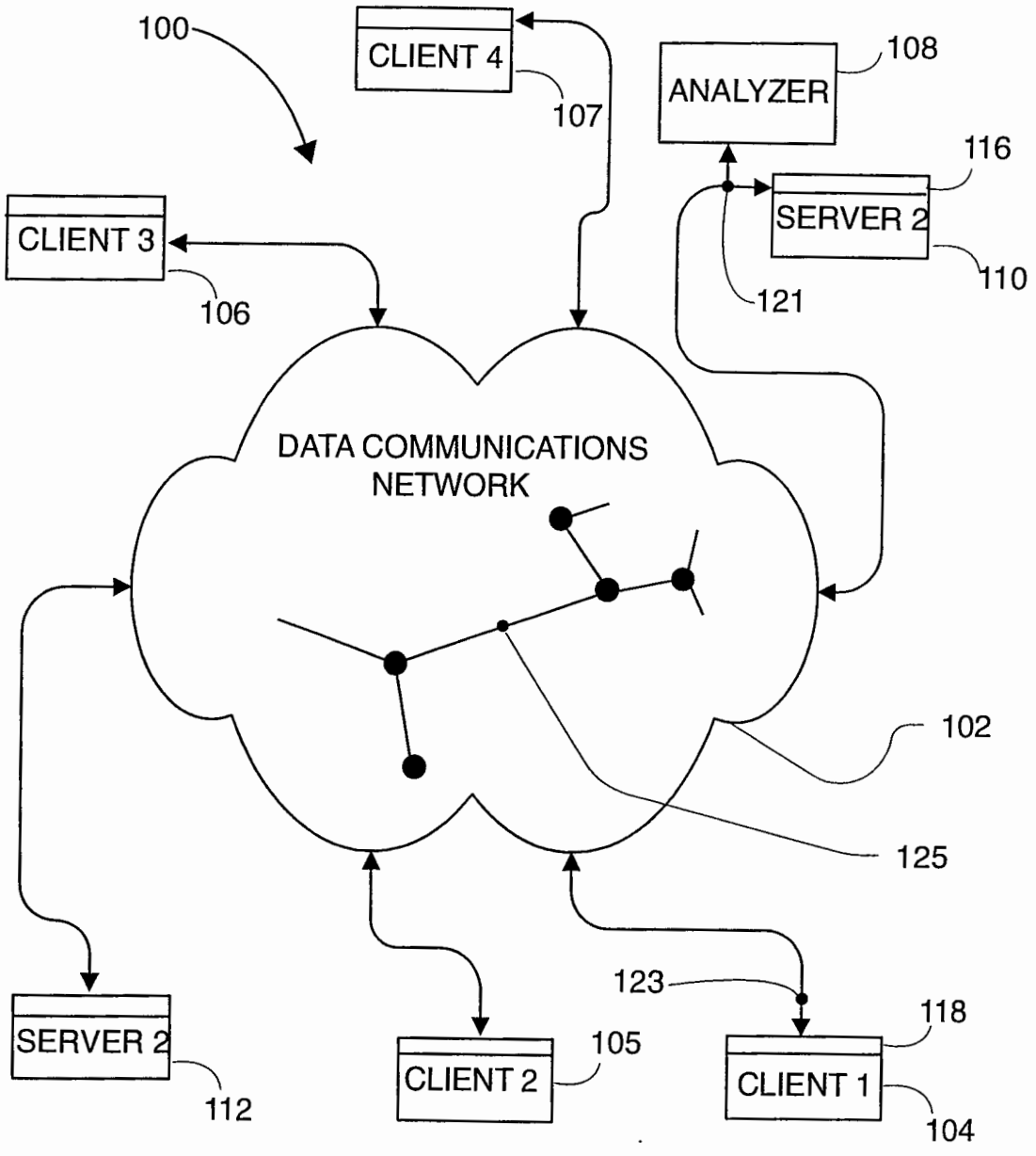

100
CLIENT 4
107
ANALYZER
108
116
SERVER 2
110
121
CLIENT 3
106
DATA COMMUNICATIONS NETWORK
102
125
123
SERVER 2
112
CLIENT 2
105
CLIENT 1
118
104

FIG. 1

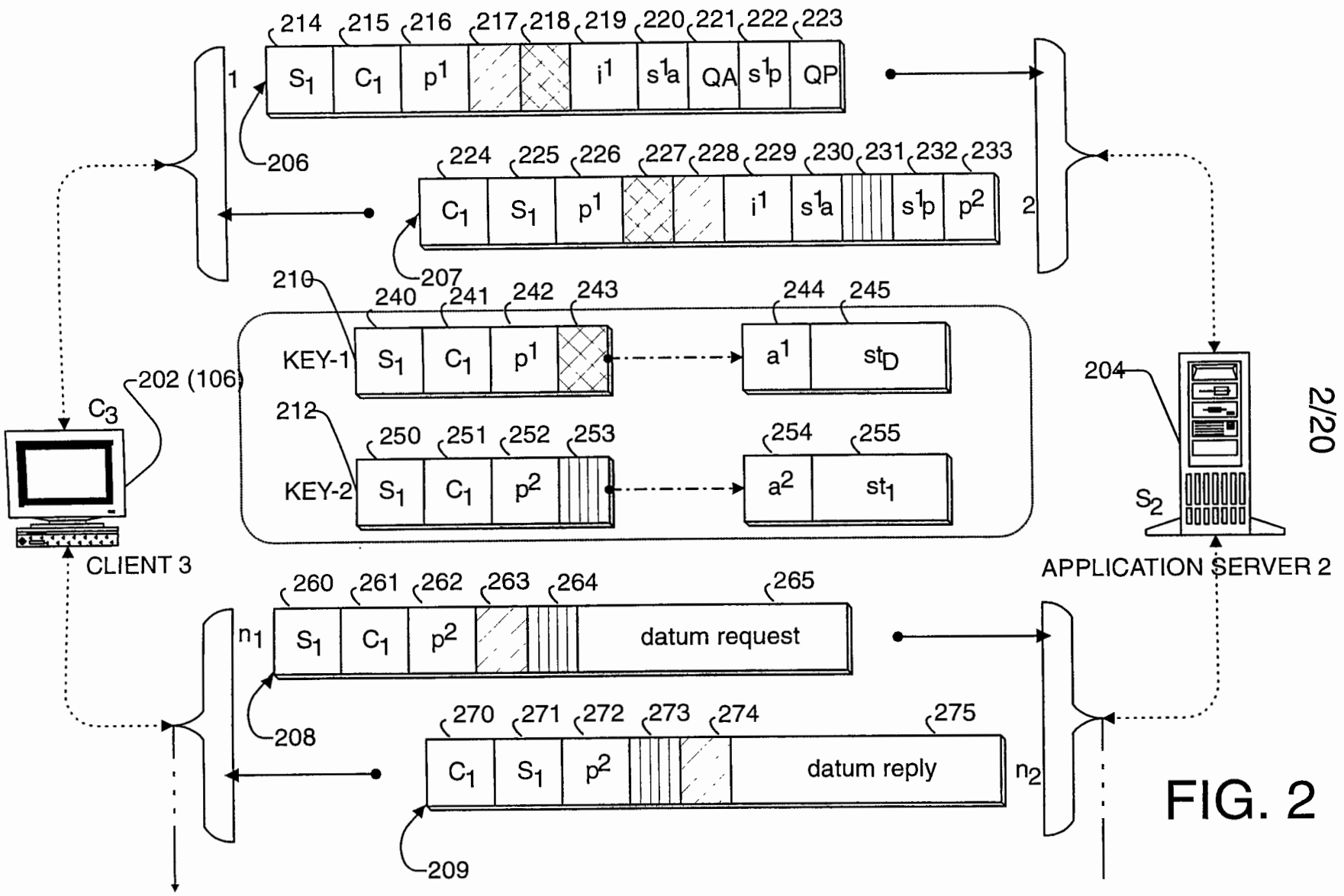

214 215 216 217 218 219 220 221 222 223
S1 C1 p1 i1 s1a QA s1p QP
1
206
224 225 226 227 228 229 230 231 232 233
C1 S1 p1 i1 s1a s1p p2
2
207
210
240 241 242 243
KEY-1 S1 C1 p1
244 245
a1 stD
212
250 251 252 253
KEY-2 S1 C1 p2
254 255
a2 st1
202 (106)
C3
CLIENT 3
204
S2
APPLICATION SERVER 2
260 261 262 263 264 265
n1
S1 C1 p2 datum request
208
270 271 272 273 274 275
C1 S1 p2 datum reply
n2
209

FIG. 2

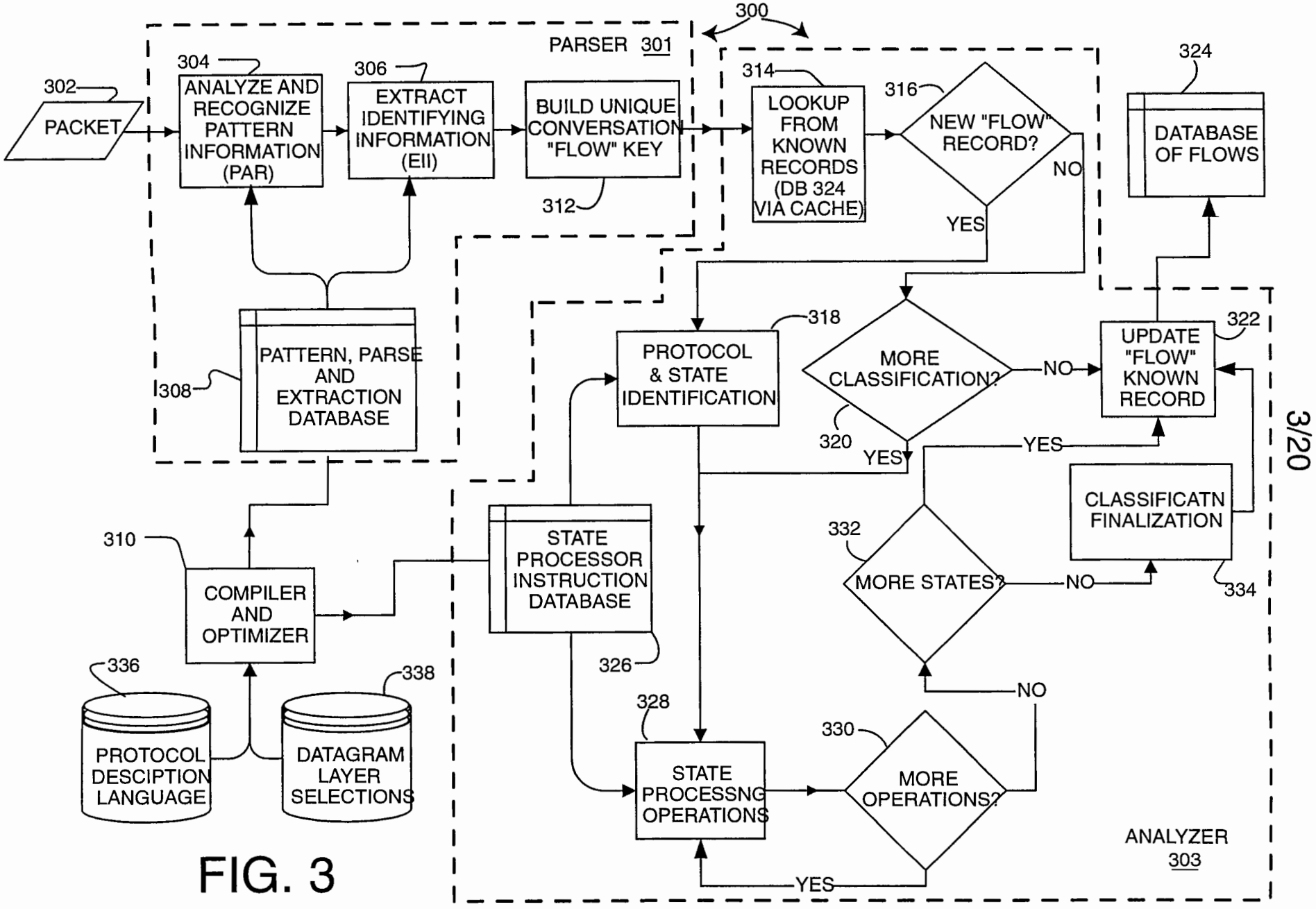

300
PARSER 301
302
PACKET
304
ANALYZE AND RECOGNIZE PATTERN INFORMATION (PAR)
306
EXTRACT IDENTIFYING INFORMATION (EII)
BUILD UNIQUE CONVERSATION "FLOW" KEY
312
314
LOOKUP FROM KNOWN RECORDS (DB 324 VIA CACHE)
316
NEW "FLOW" RECORD?
NO
YES
324
DATABASE OF FLOWS
308
PATTERN, PARSE AND EXTRACTION DATABASE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

FIG. 3

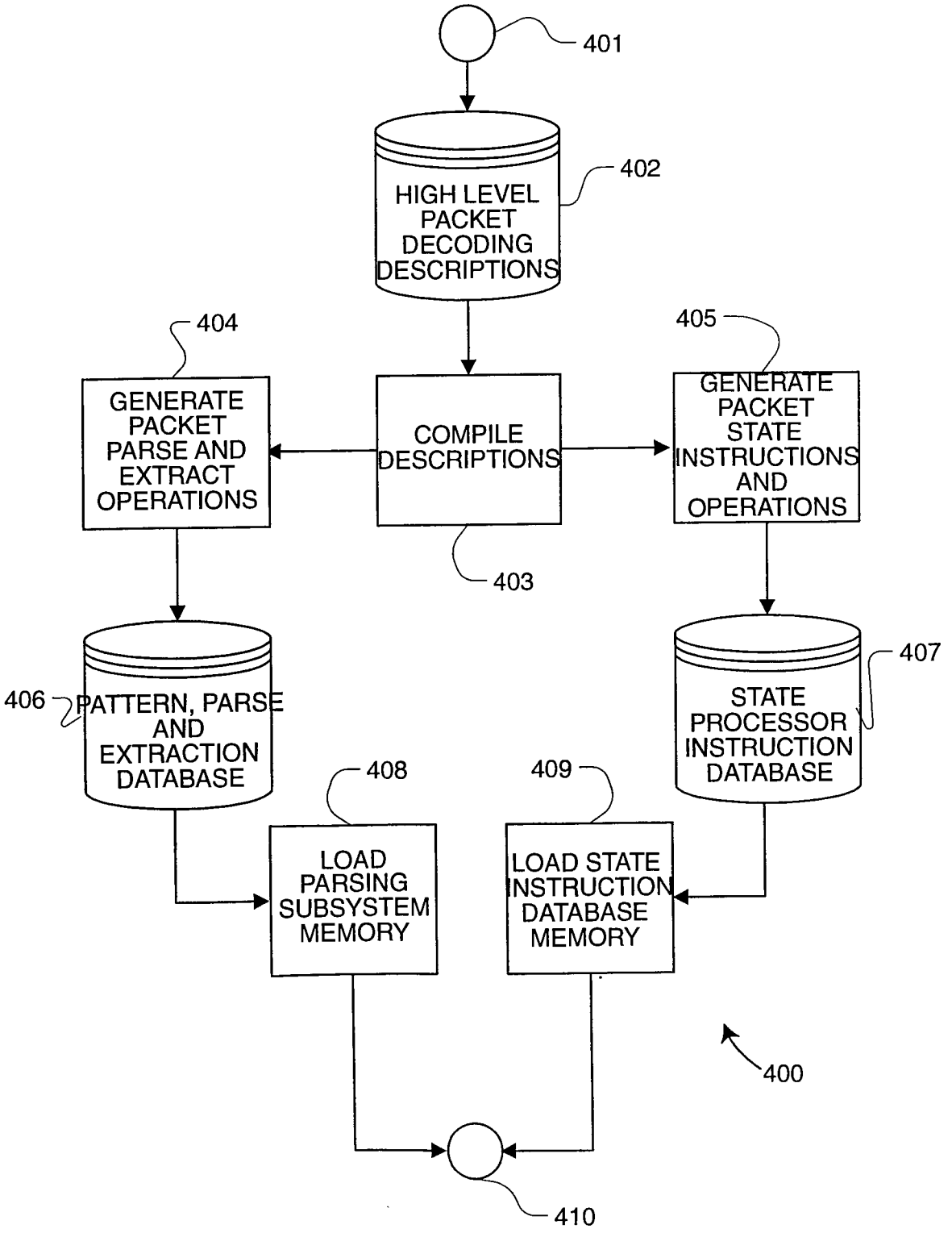

401
HIGH LEVEL PACKET DECODING DESCRIPTIONS
402
404
GENERATE PACKET PARSE AND EXTRACT OPERATIONS
COMPILE DESCRIPTIONS
403
405
GENERATE PACKET STATE INSTRUCTIONS AND OPERATIONS
406
PATTERN, PARSE AND EXTRACTION DATABASE
407
STATE PROCESSOR INSTRUCTION DATABASE
408
LOAD PARSING SUBSYSTEM MEMORY
409
LOAD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

FIG. 4

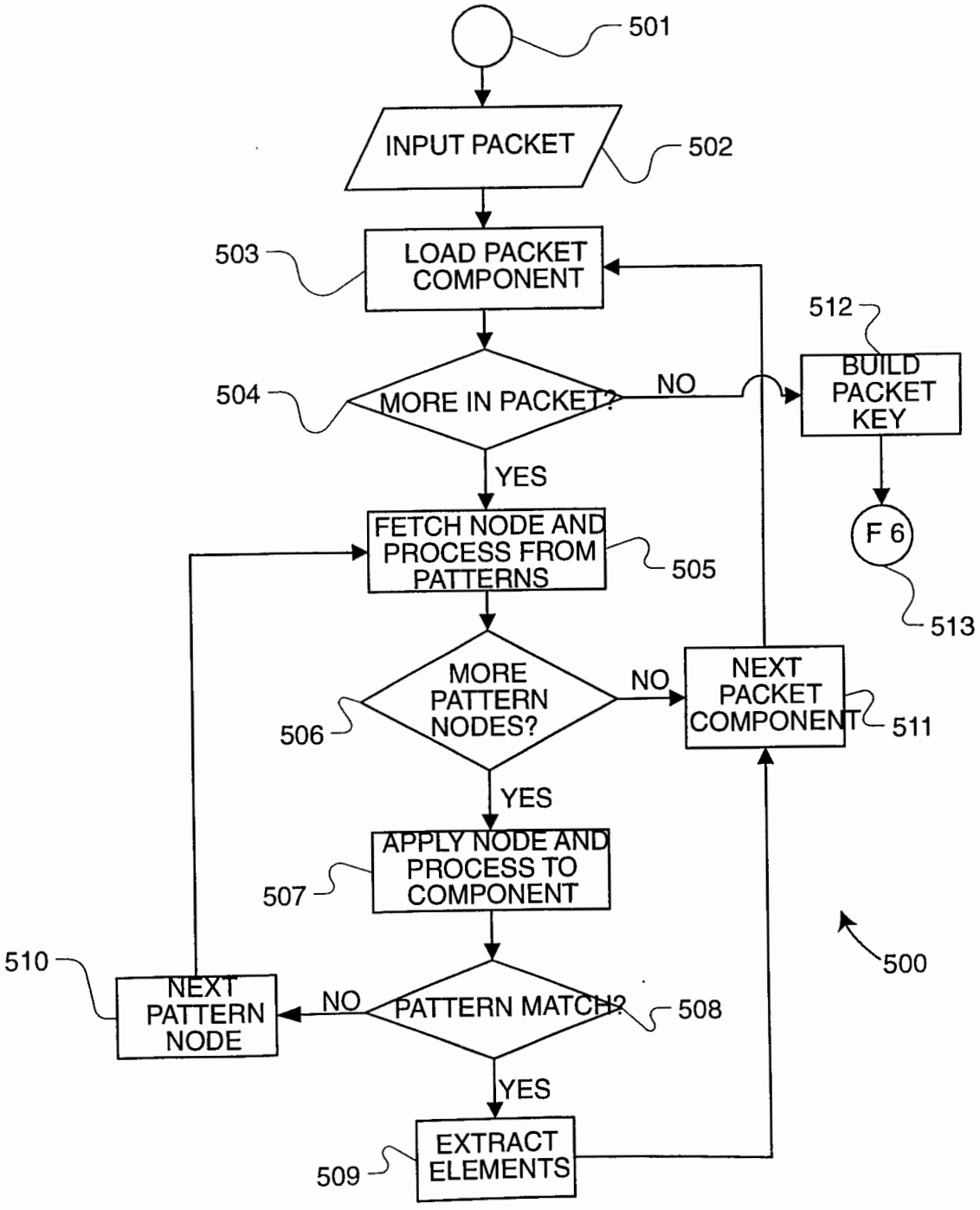

501
INPUT PACKET
502
LOAD PACKET COMPONENT
503
MORE IN PACKET?
504
NO
YES
BUILD PACKET KEY
512
F 6
513
FETCH NODE AND PROCESS FROM PATTERNS
505
MORE PATTERN NODES?
506
NO
NEXT PACKET COMPONENT
511
YES
APPLY NODE AND PROCESS TO COMPONENT
507
500
NEXT PATTERN NODE
510
NO
PATTERN MATCH?
508
YES
EXTRACT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

FIG. 5

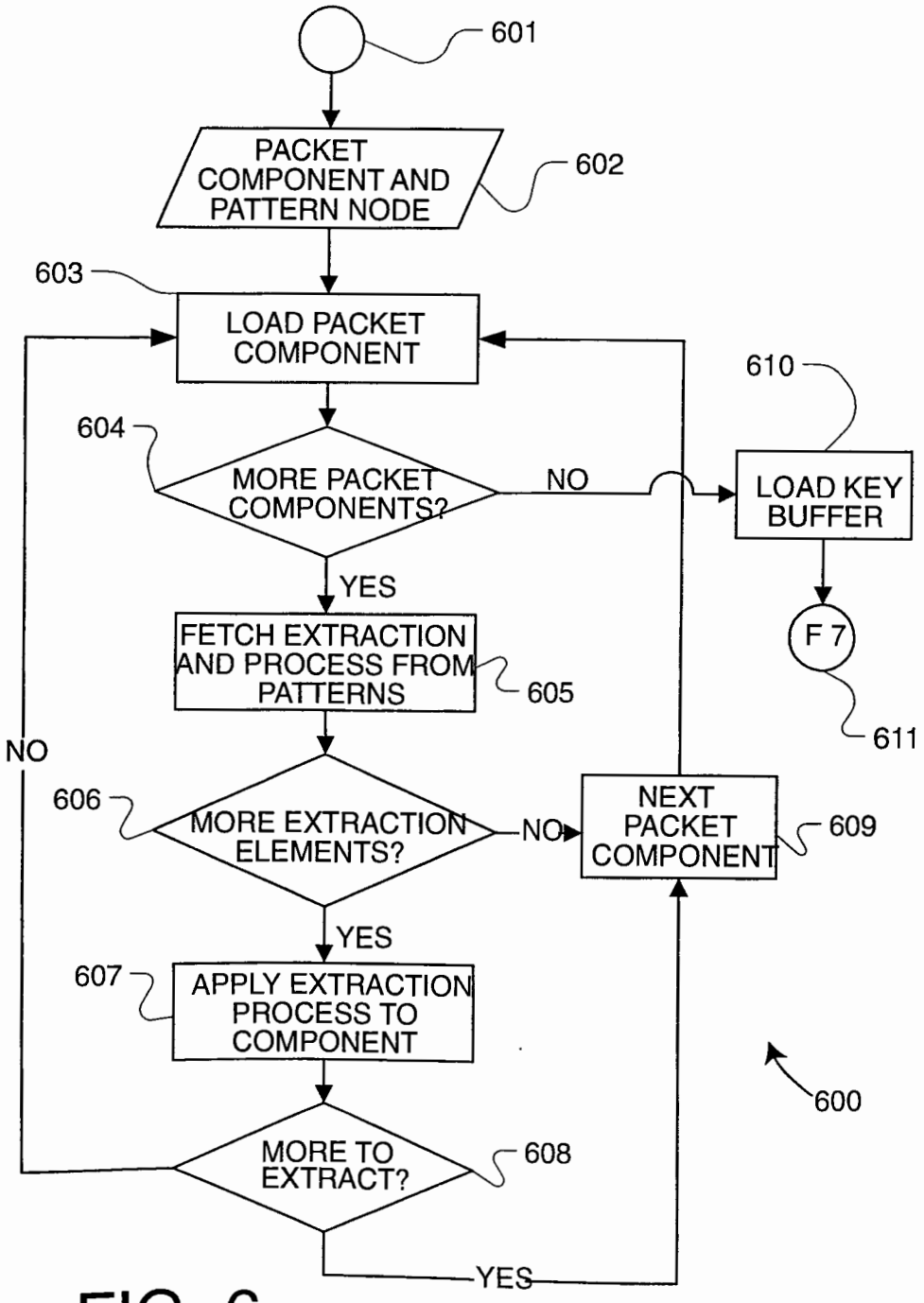

601
PACKET COMPONENT AND PATTERN NODE
602
603
LOAD PACKET COMPONENT
610
604
MORE PACKET COMPONENTS?
NO
LOAD KEY BUFFER
YES
F 7
FETCH EXTRACTION AND PROCESS FROM PATTERNS
605
611
NO
606
MORE EXTRACTION ELEMENTS?
NO
NEXT PACKET COMPONENT
609
YES
607
APPLY EXTRACTION PROCESS TO COMPONENT
600
MORE TO EXTRACT?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

FIG. 6

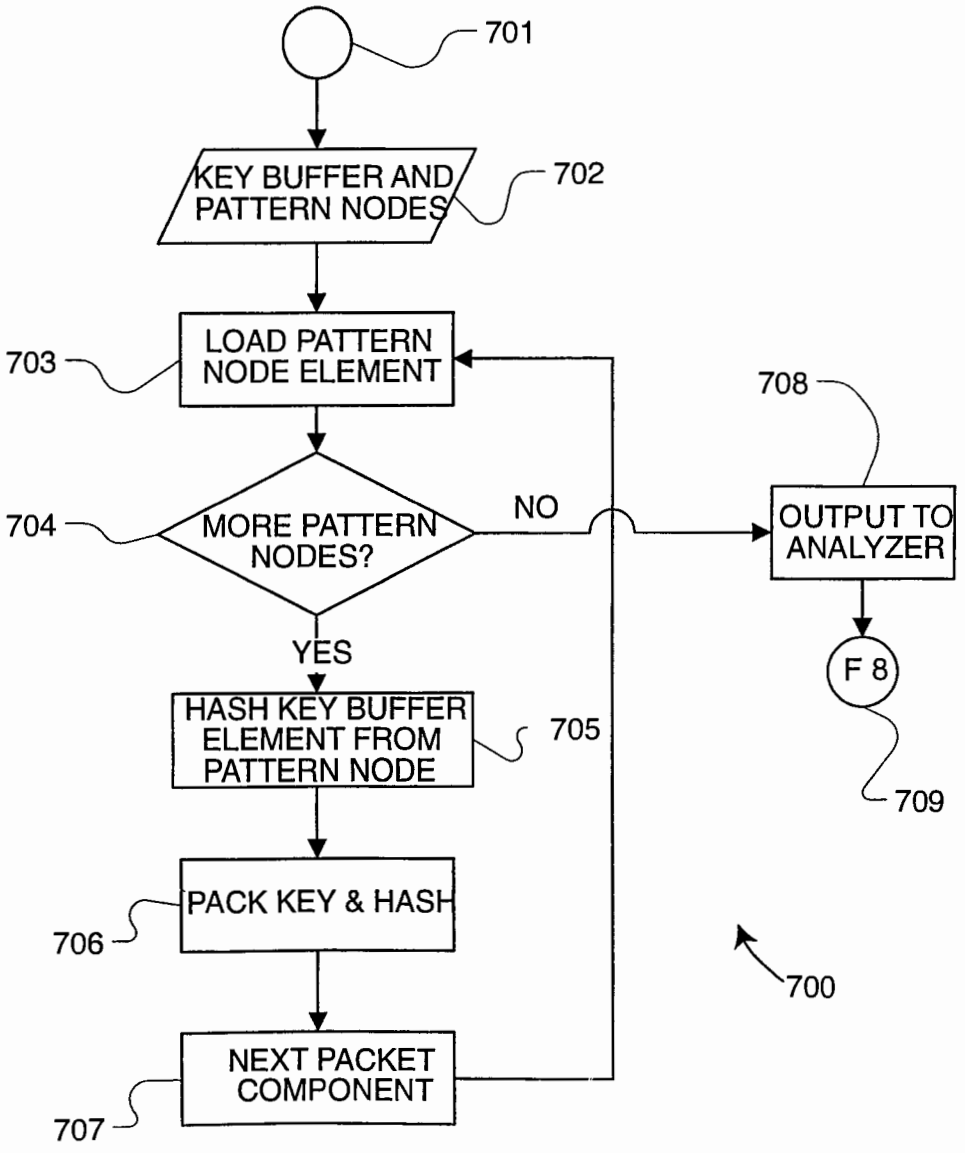

701
KEY BUFFER AND PATTERN NODES
702
LOAD PATTERN NODE ELEMENT
703
MORE PATTERN NODES?
704
NO
YES
OUTPUT TO ANALYZER
708
F 8
709
HASH KEY BUFFER ELEMENT FROM PATTERN NODE
705
PACK KEY & HASH
706
NEXT PACKET COMPONENT
707
700

FIG. 7

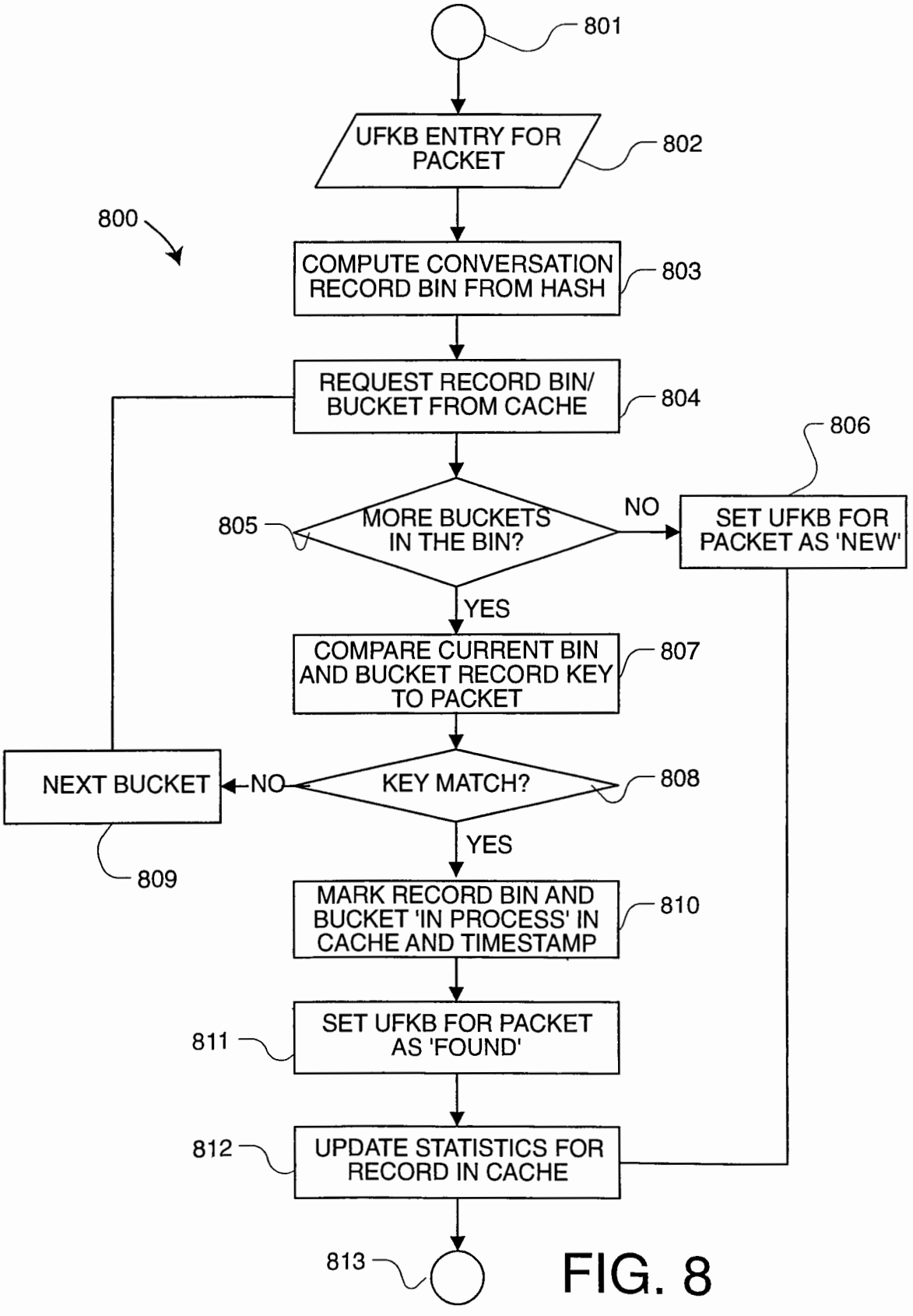

801
UFKB ENTRY FOR PACKET
802
800
COMPUTE CONVERSATION RECORD BIN FROM HASH
803
REQUEST RECORD BIN/ BUCKET FROM CACHE
804
806
805
MORE BUCKETS IN THE BIN?
NO
SET UFKB FOR PACKET AS 'NEW'
YES
COMPARE CURRENT BIN AND BUCKET RECORD KEY TO PACKET
807
NEXT BUCKET
NO
KEY MATCH?
808
YES
809
MARK RECORD BIN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

FIG. 8

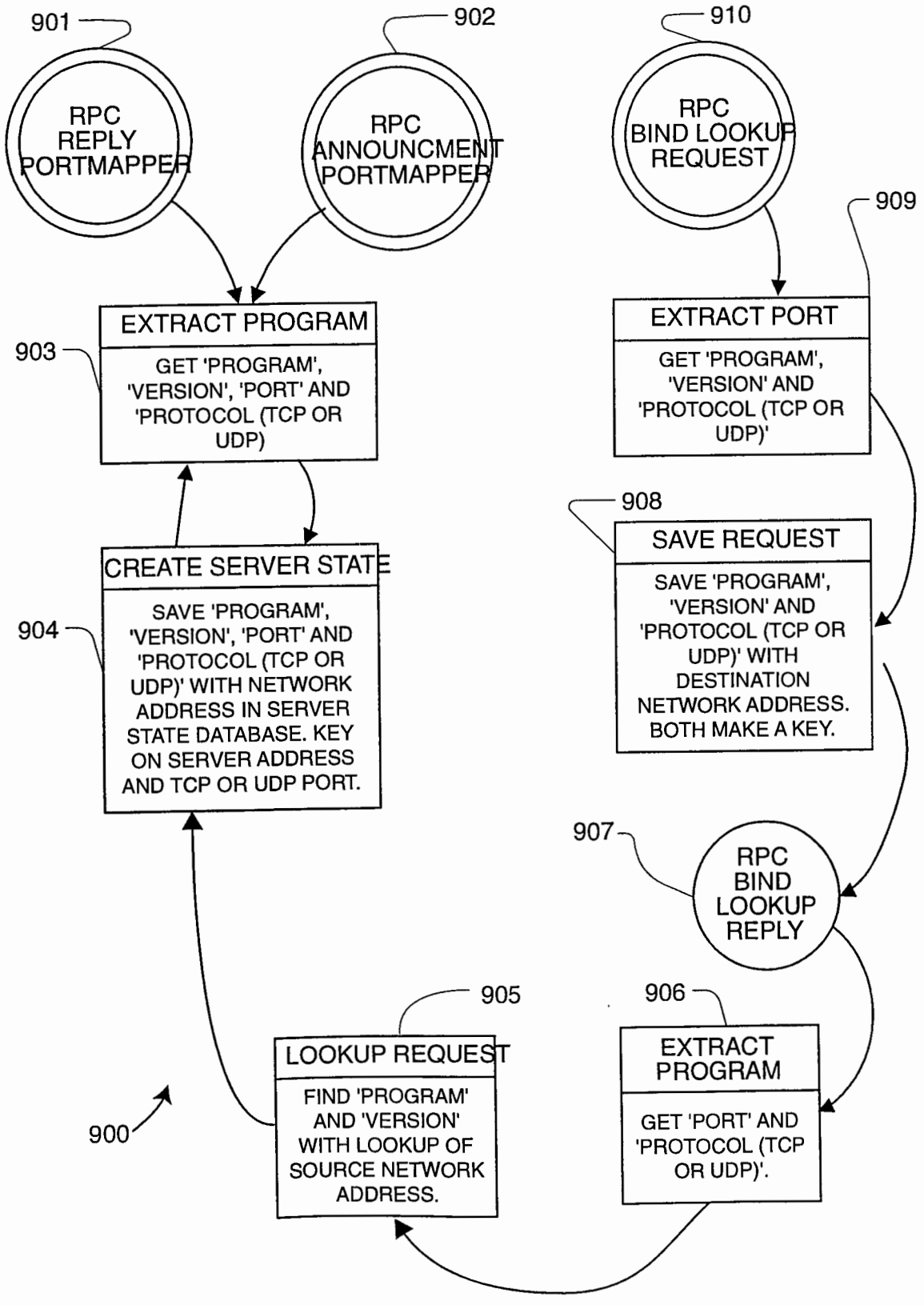

901
RPC REPLY PORTMAPPER
902
RPC ANNOUNCMENT PORTMAPPER
910
RPC BIND LOOKUP REQUEST
909
EXTRACT PROGRAM
GET 'PROGRAM', 'VERSION', 'PORT' AND 'PROTOCOL (TCP OR UDP)
903
EXTRACT PORT
GET 'PROGRAM', 'VERSION' AND 'PROTOCOL (TCP OR UDP)'
908
SAVE REQUEST
SAVE 'PROGRAM', 'VERSION' AND 'PROTOCOL (TCP OR UDP)'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

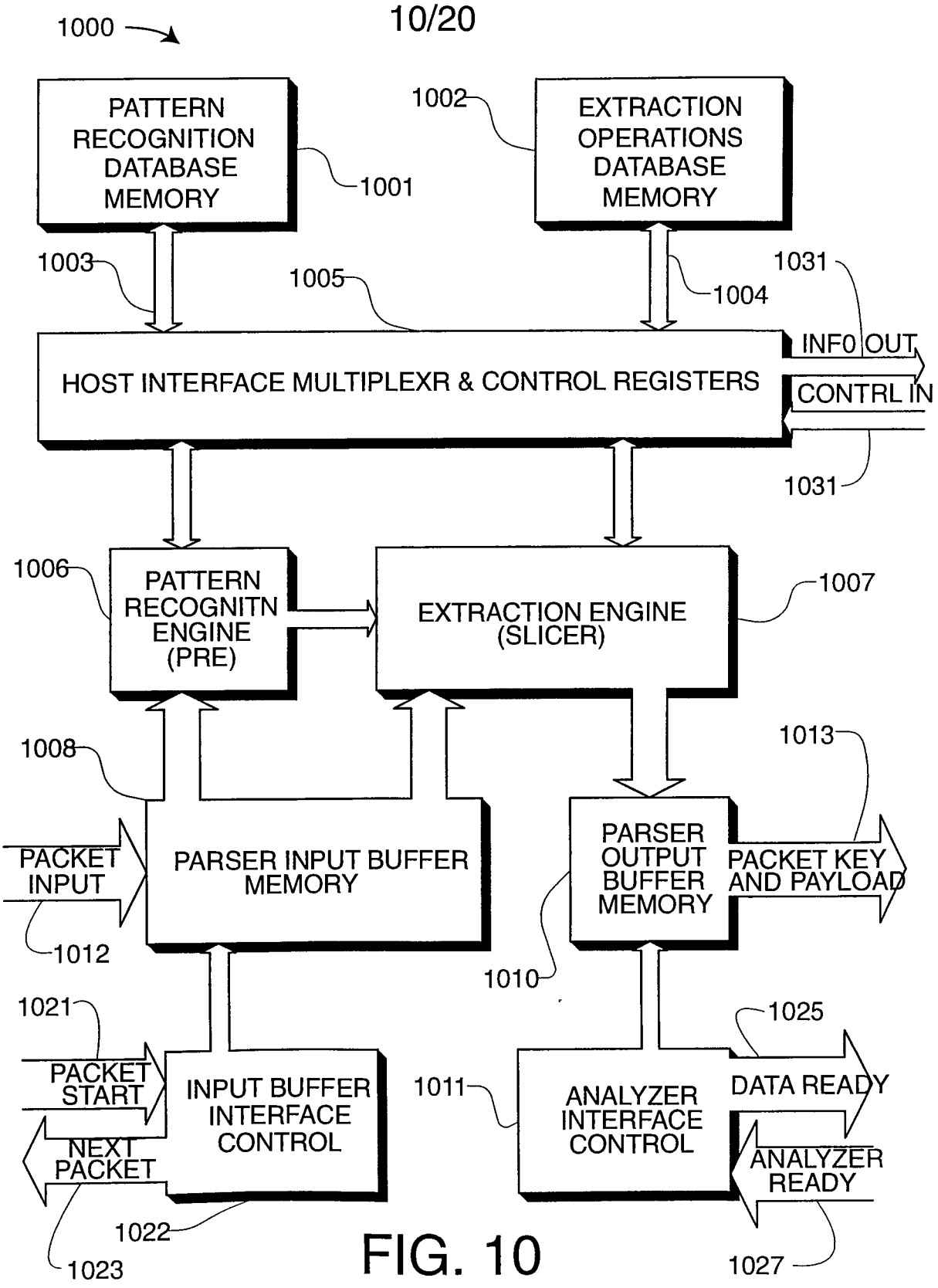

1000
PATTERN RECOGNITION DATABASE MEMORY
1001
1002
EXTRACTION OPERATIONS DATABASE MEMORY
1003
1004
1005
1031
INF0 OUT
CONTRL IN
HOST INTERFACE MULTIPLEXR & CONTROL REGISTERS
1031
1006
PATTERN RECOGNITN ENGINE (PRE)
EXTRACTION ENGINE (SLICER)
1007
1008
1013
PACKET INPUT
PARSER INPUT BUFFER MEMORY
PARSER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

FIG. 10

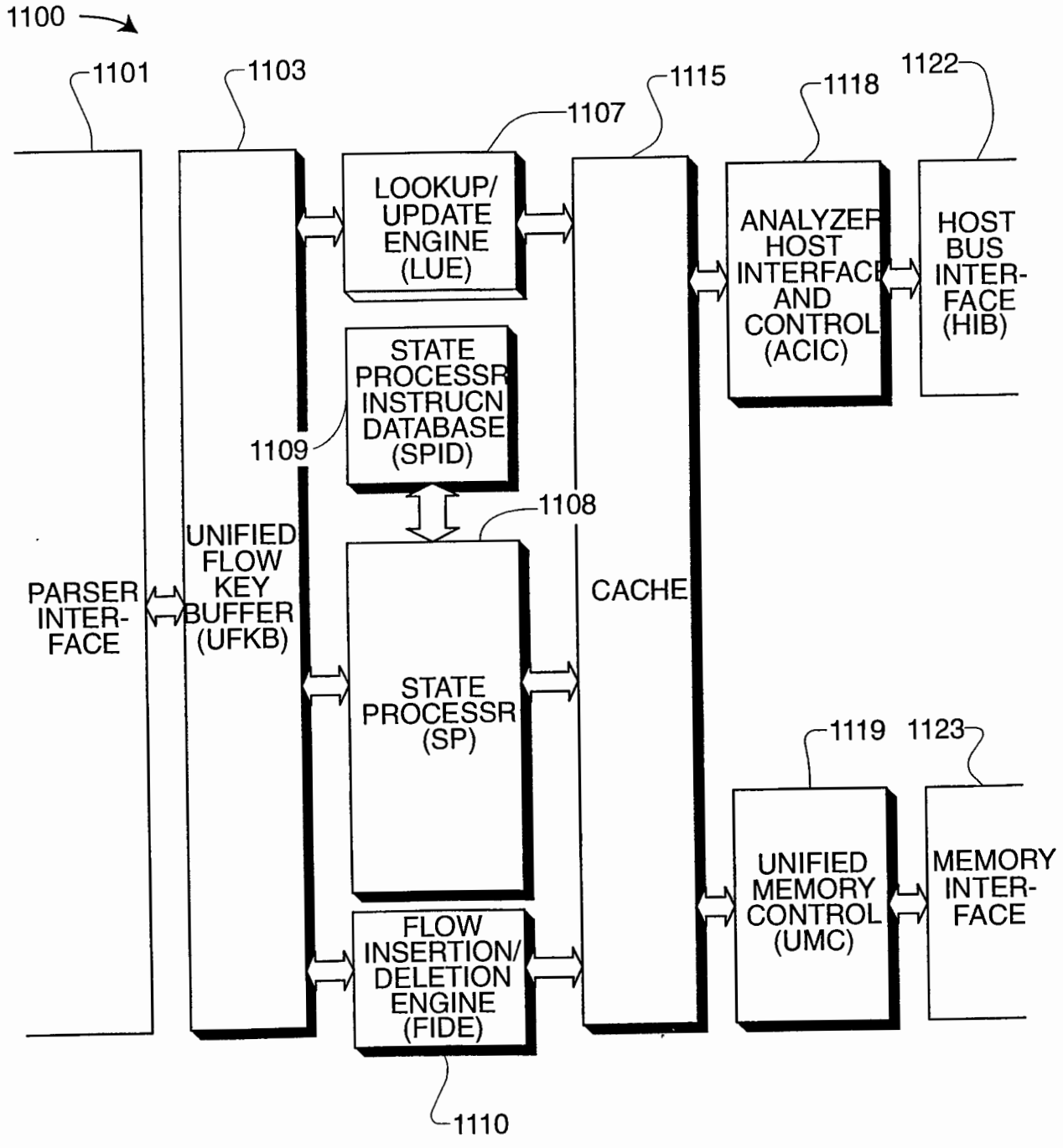

1100
1101
1103
1107
1115
1118
1122
LOOKUP/ UPDATE ENGINE (LUE)
ANALYZER HOST INTERFACE AND CONTROL (ACIC)
HOST BUS INTER-FACE (HIB)
STATE PROCESSR INSTRUCN DATABASE (SPID)
1109
1108
UNIFIED FLOW KEY BUFFER (UFKB)
PARSER INTER-FACE
CACHE
STATE PROCESSR (SP)
1119
1123
UNIFIED MEMORY CONTROL (UMC)
MEMORY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

FIG. 11

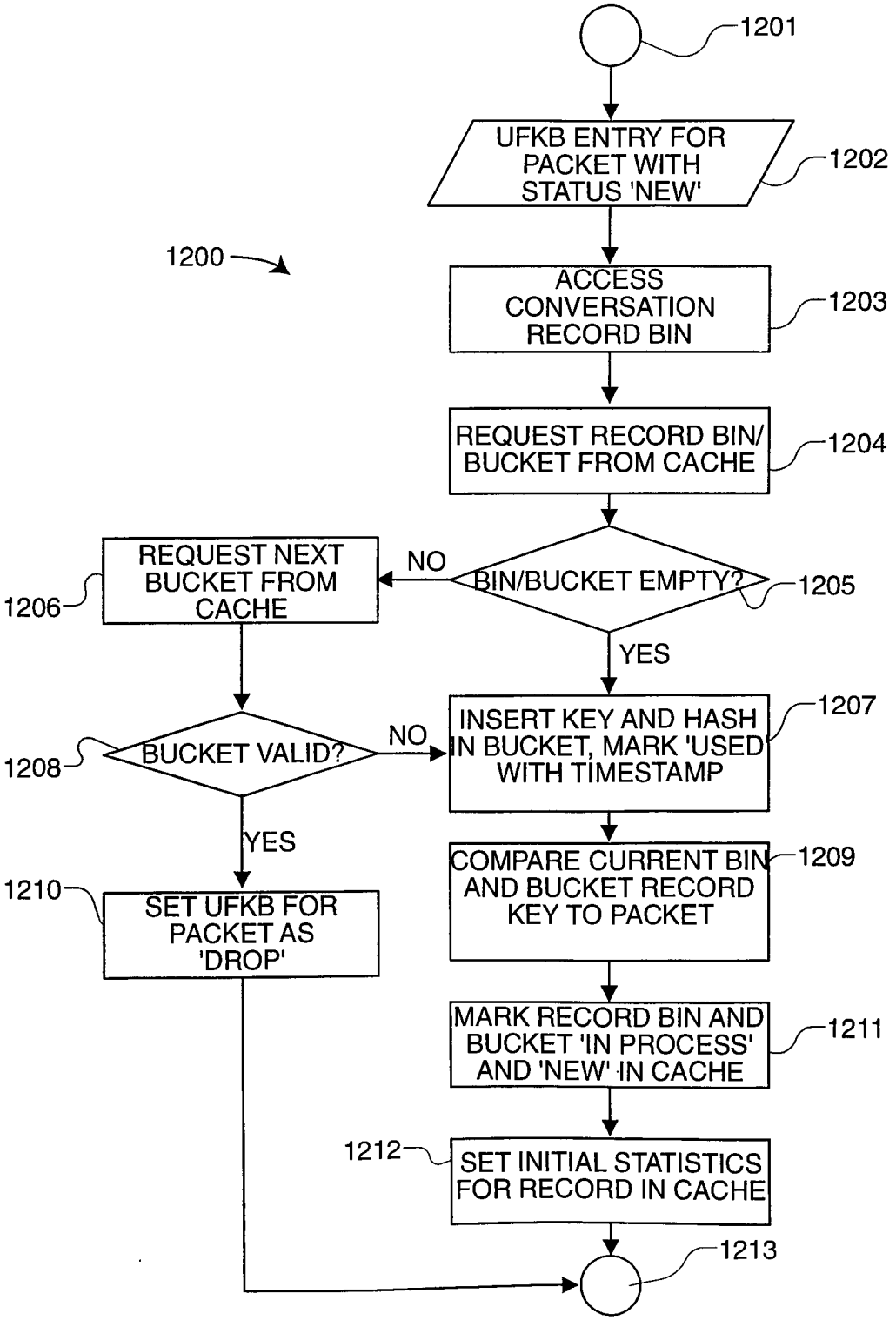

1201
UFKB ENTRY FOR PACKET WITH STATUS 'NEW'
1202
1200
ACCESS CONVERSATION RECORD BIN
1203
REQUEST RECORD BIN/ BUCKET FROM CACHE
1204
REQUEST NEXT BUCKET FROM CACHE
1206
NO
BIN/BUCKET EMPTY?
1205
YES
INSERT KEY AND HASH IN BUCKET, MARK 'USED' WITH TIMESTAMP
1207
BUCKET VALID?
1208
NO
YES
COMPARE CURRENT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

FIG. 12

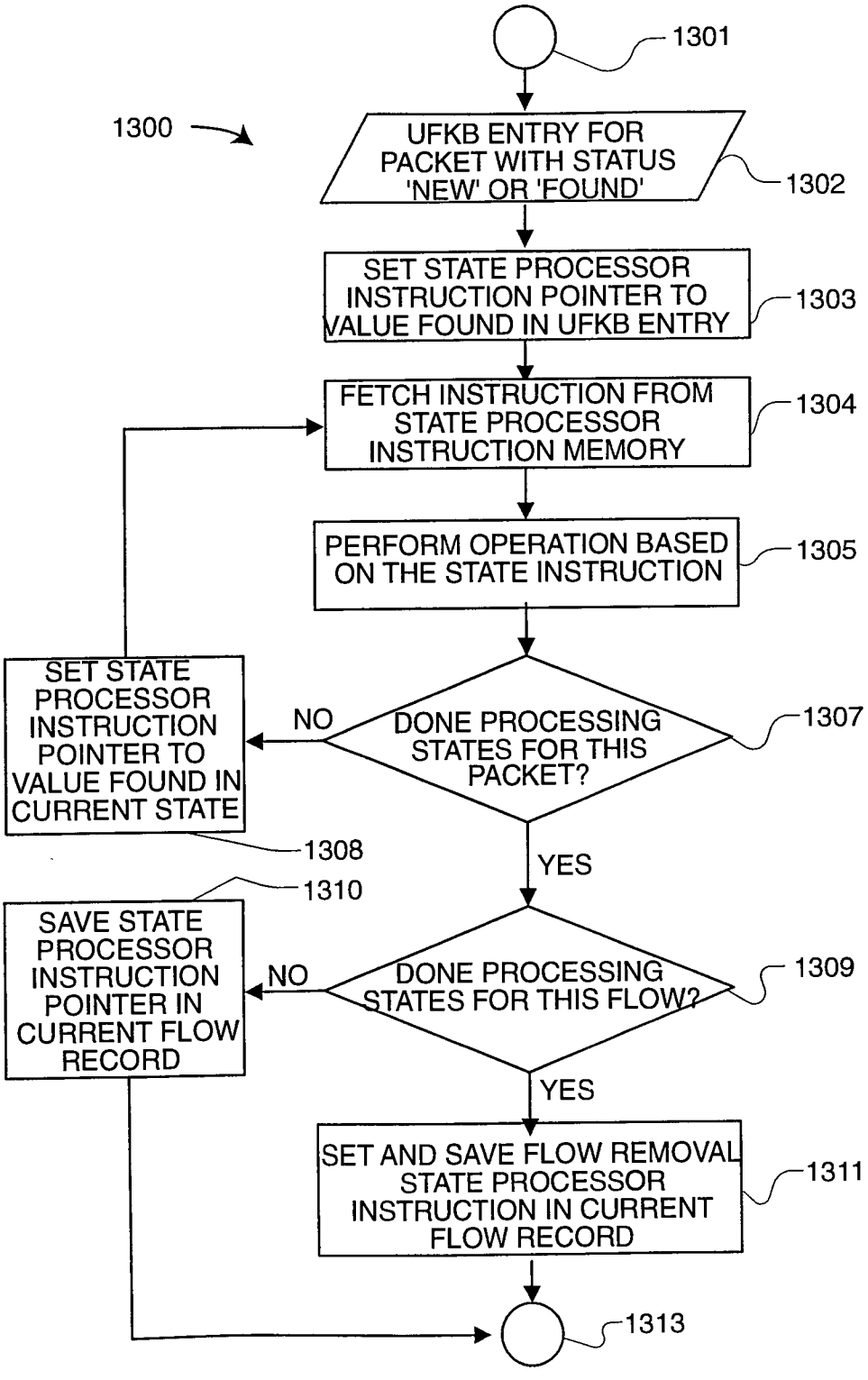

1301
1300
UFKB ENTRY FOR PACKET WITH STATUS 'NEW' OR 'FOUND'
1302
SET STATE PROCESSOR INSTRUCTION POINTER TO VALUE FOUND IN UFKB ENTRY
1303
FETCH INSTRUCTION FROM STATE PROCESSOR INSTRUCTION MEMORY
1304
PERFORM OPERATION BASED ON THE STATE INSTRUCTION
1305
DONE PROCESSING STATES FOR THIS PACKET?
1307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

FIG. 13

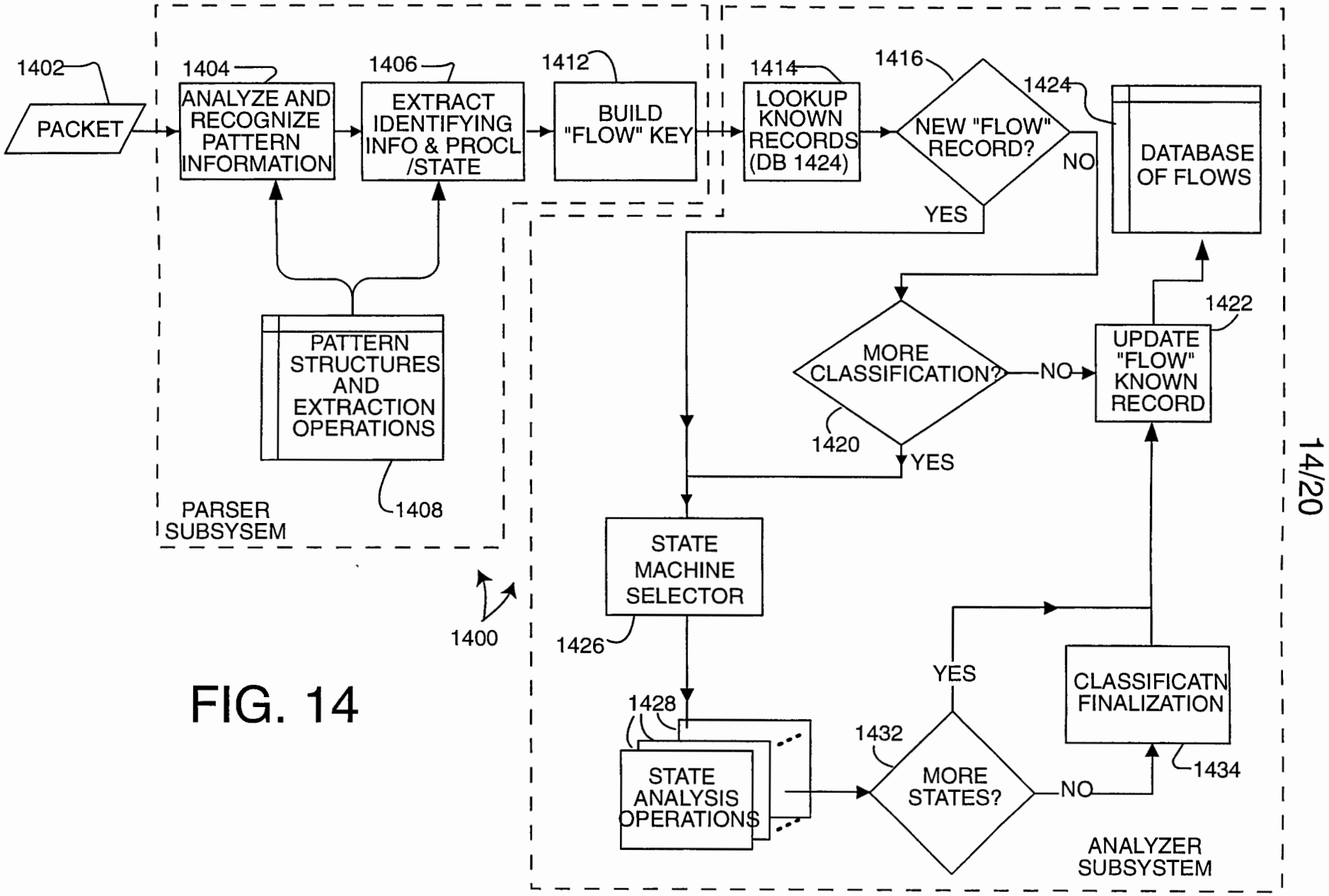

1402
PACKET
1404
ANALYZE AND RECOGNIZE PATTERN INFORMATION
1406
EXTRACT IDENTIFYING INFO & PROCL /STATE
1412
BUILD "FLOW" KEY
1414
LOOKUP KNOWN RECORDS (DB 1424)
1416
NEW "FLOW" RECORD?
NO
YES
1424
DATABASE OF FLOWS
PATTERN STRUCTURES AND EXTRACTION OPERATIONS
1408
PARSER SUBSYSEM
1400
MORE CLASSIFICATION?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

FIG. 14

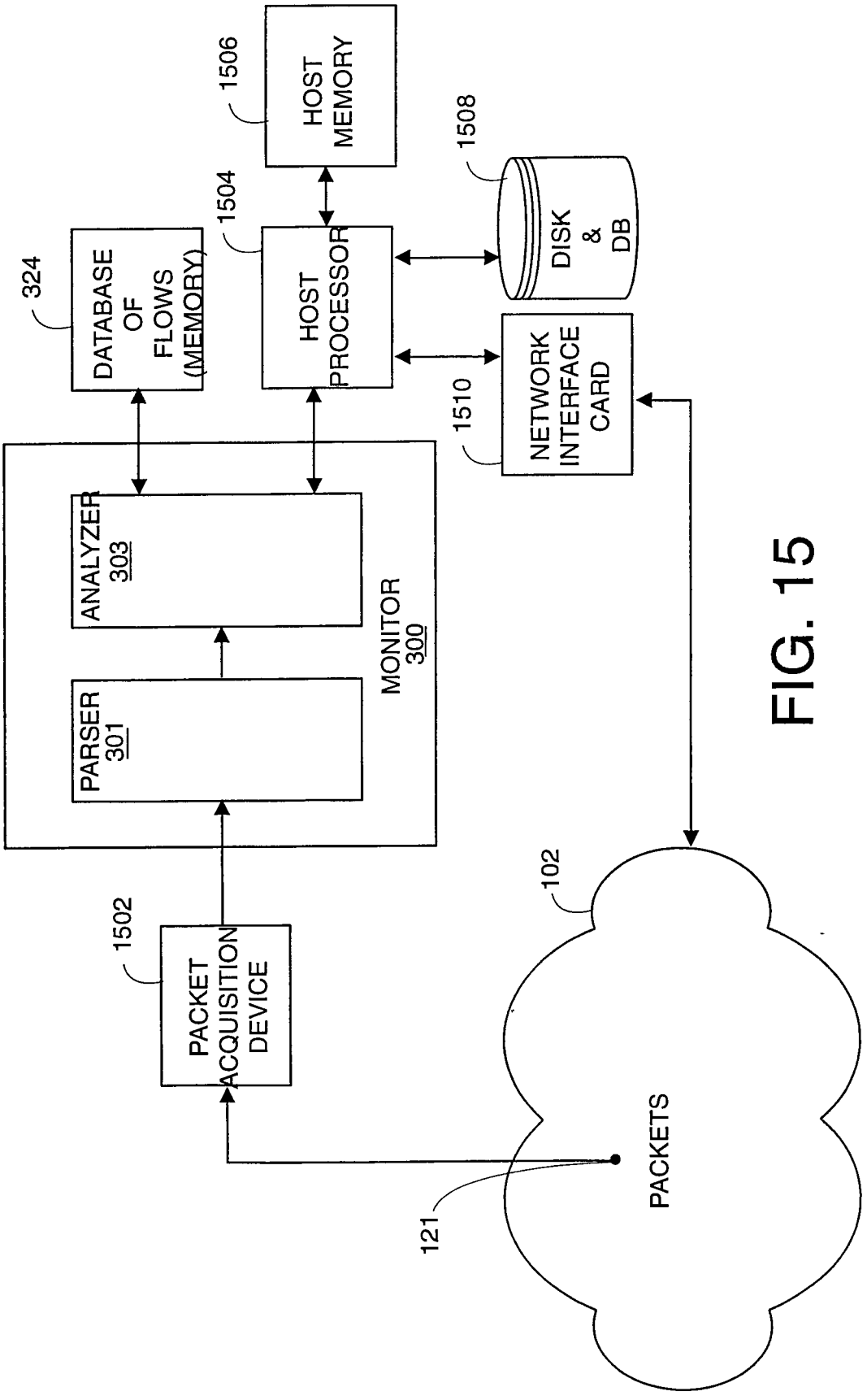

HOST MEMORY
1506
1508
DISK & DB
1504
HOST PROCESSOR
324
DATABASE OF FLOWS (MEMORY)
NETWORK INTERFACE CARD
1510
ANALYZER 303
MONITOR 300
PARSER 301
102
1502
PACKET ACQUISITION DEVICE
PACKETS
121

FIG. 15

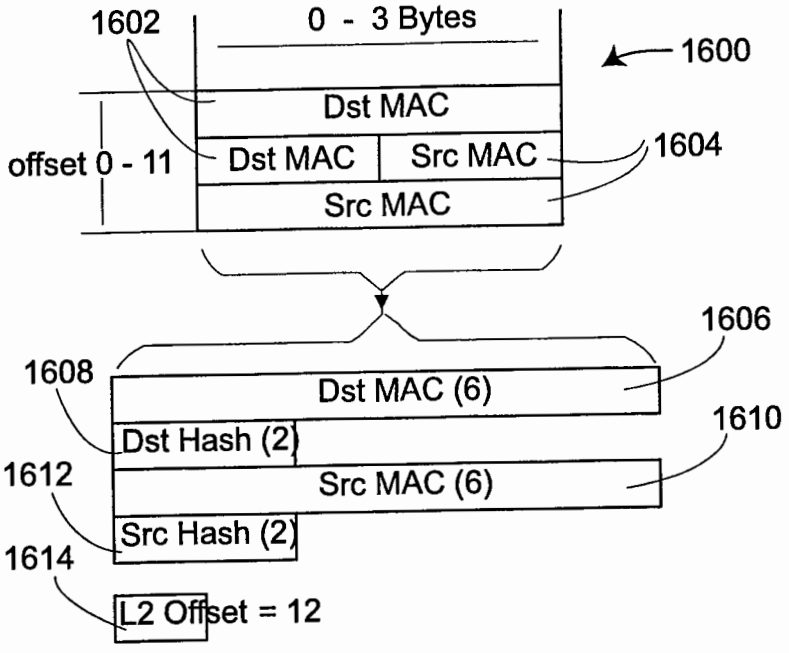

1602
0 - 3 Bytes
1600
Dst MAC
offset 0 - 11
Dst MAC
Src MAC
1604
Src MAC
1606
1608
Dst MAC (6)
Dst Hash (2)
1610
1612
Src MAC (6)
Src Hash (2)
1614
L2 Offset = 12

FIG. 16

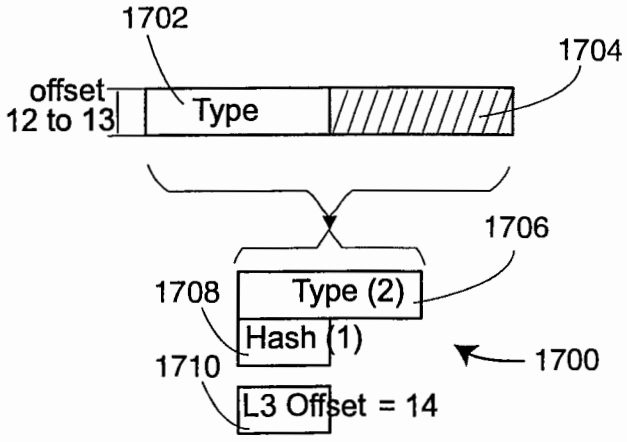


FIG. 17A

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

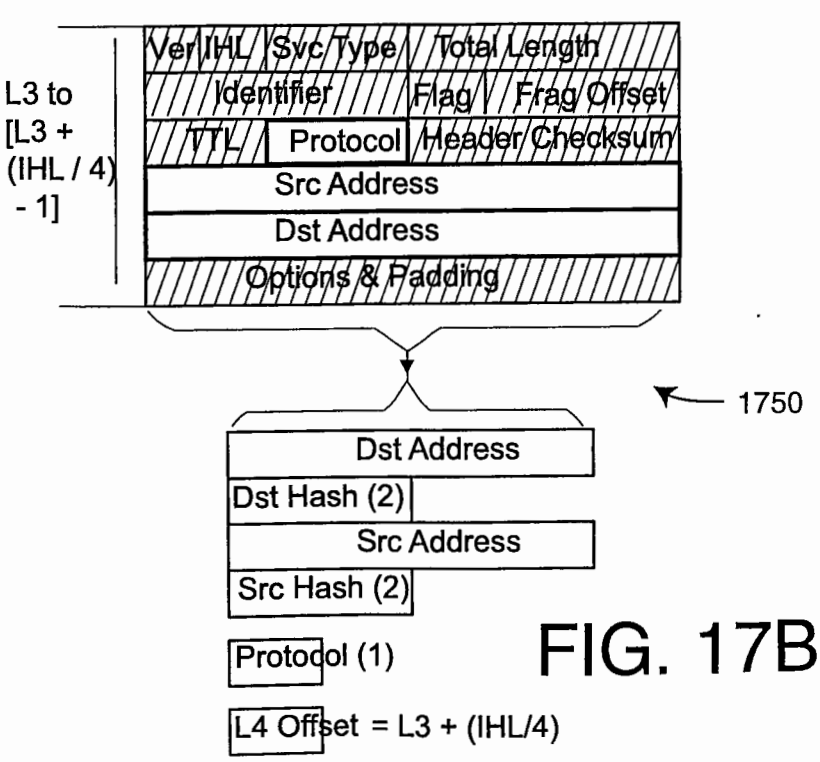


FIG. 17B

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

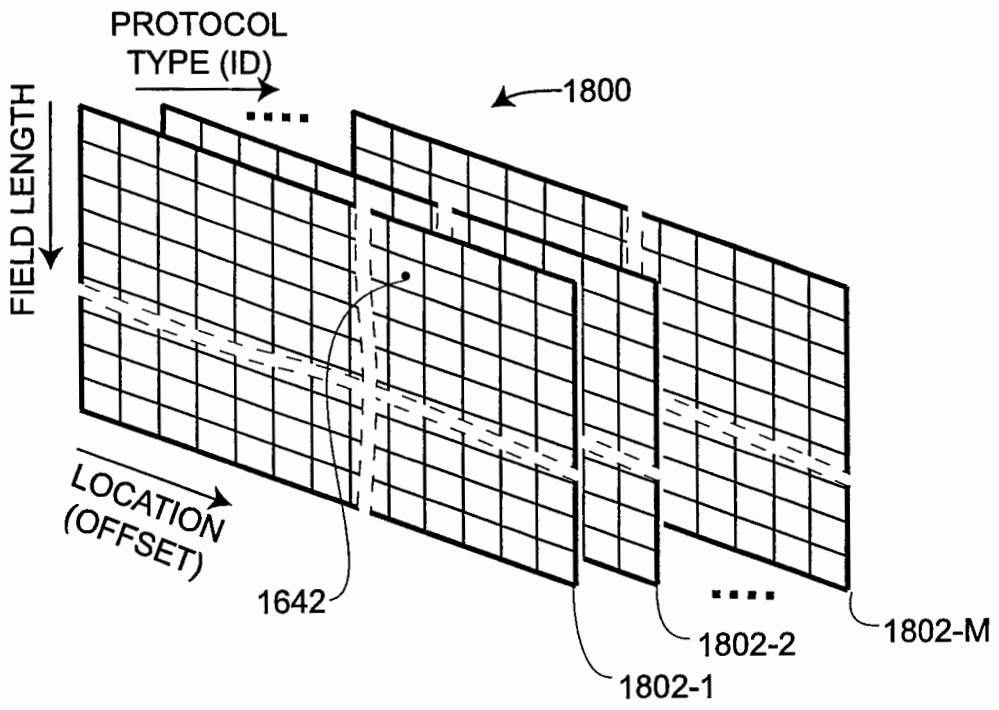


FIG. 18A

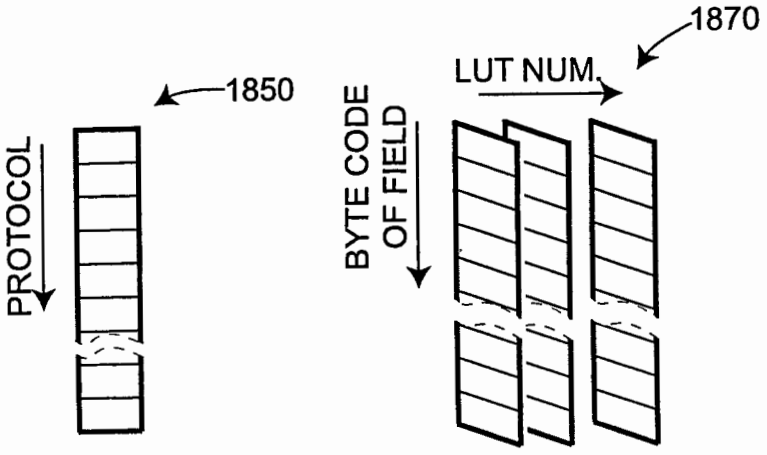


FIG. 18B

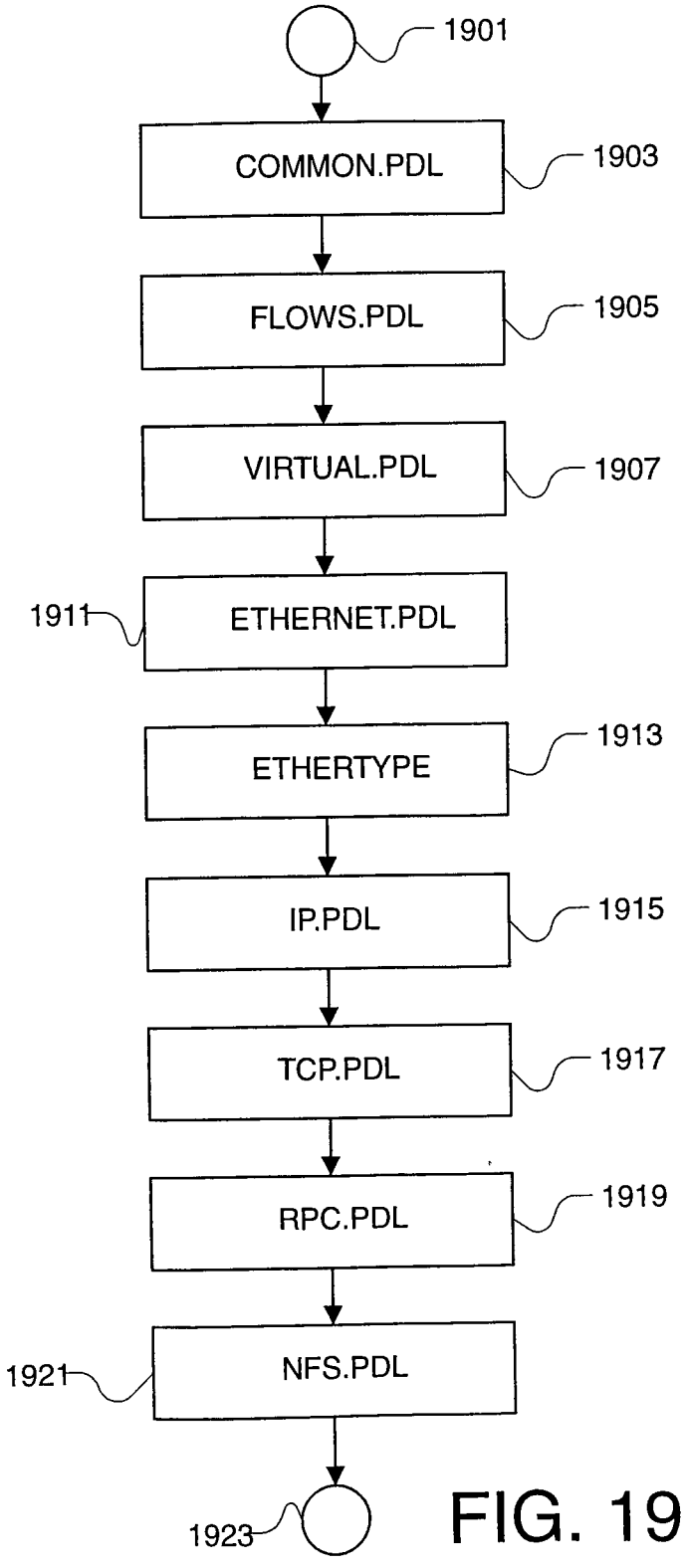

1901
COMMON.PDL
1903
FLOWS.PDL
1905
VIRTUAL.PDL
1907
1911
ETHERNET.PDL
ETHERTYPE
1913
IP.PDL
1915
TCP.PDL
1917
RPC.PDL
1919
1921
NFS.PDL
1923

FIG. 19

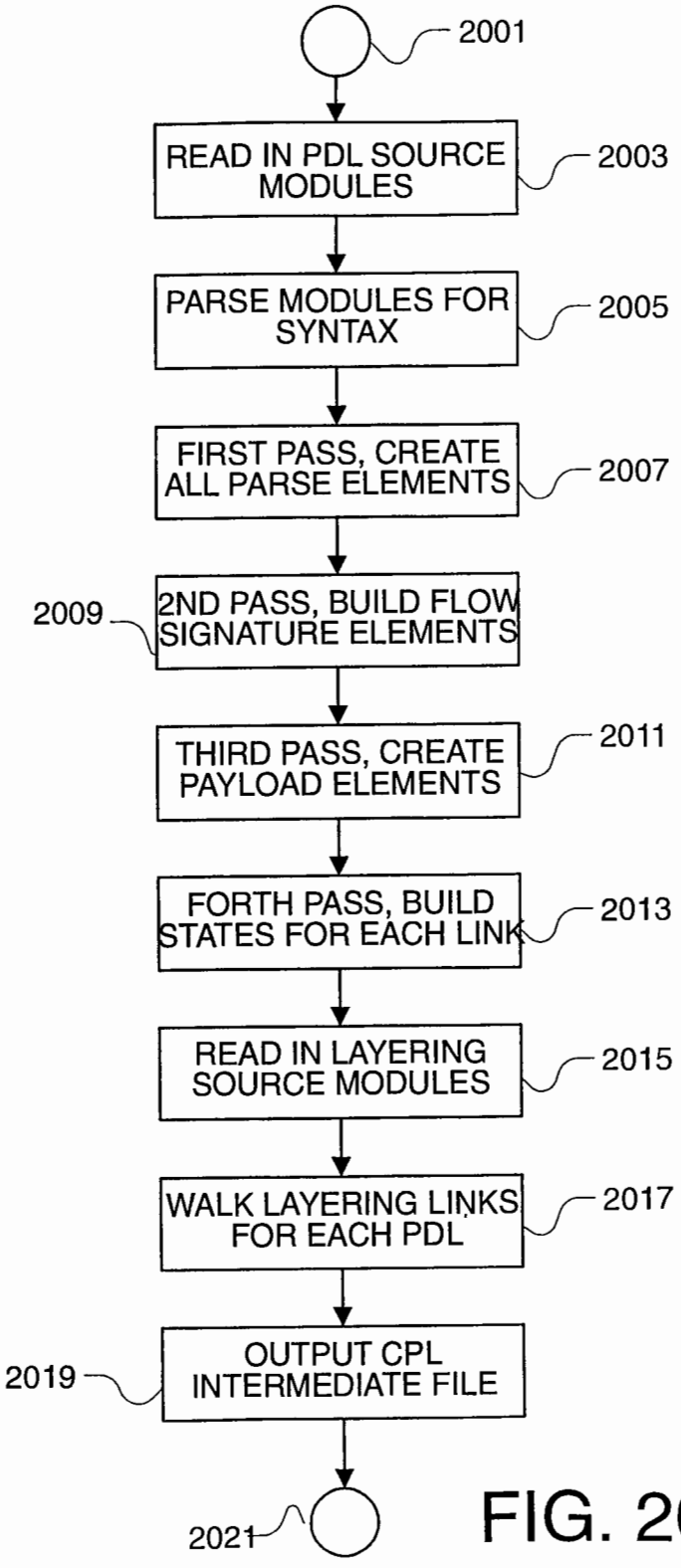

2001
READ IN PDL SOURCE MODULES
2003
PARSE MODULES FOR SYNTAX
2005
FIRST PASS, CREATE ALL PARSE ELEMENTS
2007
2009
2ND PASS, BUILD FLOW SIGNATURE ELEMENTS
THIRD PASS, CREATE PAYLOAD ELEMENTS
2011
FORTH PASS, BUILD STATES FOR EACH LINK
2013
READ IN LAYERING SOURCE MODULES
2015
WALK LAYERING LINKS FOR EACH PDL
2017
2019
OUTPUT CPL INTERMEDIATE FILE
2021

FIG. 20

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 18, 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: *09/609,179*
FILING DATE: *June 30, 2000*
PATENT NUMBER: *6,665,725*
ISSUE DATE: *December 16, 2003*

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

P. SWAIN
Certifying Officer

PART 2 OF 2 PART(S)

FORMALITIES LETTER

OC000000005346098

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
Address: COMMISSIONER OF PATENT AND TRADEMARKS
Washington, D C 20231

APPLICATION NUMBER	FILING/RECEIPT DATE	FIRST NAMED APPLICANT	ATTORNEY DOCKET NUMBER
09/609,179	06/30/2000	Russell S. Dietz	APPT-001-2

Dov Rosenfeld
5507 College Avenue
Suite 2
Oakland, CA 94618

Date Mailed: 08/23/2000

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).*
- 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 $ 820.**

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

T. Paine

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

PART 3 - OFFICE COPY

Sector 17

Our Ref./Docket No: APPS 01-2 Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz, *et al.* Application No.: 09/609179 Filed: June 30, 2000 Title: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE	Group Art Unit: 2756 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 $__860.00__ is attached, being for:

- __X__ Statutory basic filing fee: $ 690
- __X__ Additional claim fee of $ 0
- __X__ Assignment recordation fee of $ 40
- __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: Oct 20, 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,

Oct 20, 2006
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-001-2**

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:

PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE

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. 09/609179 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	Dov Rosenfeld, Reg. No. 38,687
5507 College Avenue, Suite 2	Tel: (510) 547-3378
Oakland, CA 94618	

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** 10/9/00

ADDITIONAL INVENTOR SIGNATURES:

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

Residence: 10400 Kenmore Drive, Fairfax, VA 22030

Post Office Address: Same

_____ _____

Inventor's Signature **Date**

Name of Third Inventor: James F. Torgerson **Citizenship:** USA

Residence: 227 157th Ave., NW, Andover, MN 55304

Post Office Address: Same

_____ _____

Inventor's Signature **Date**

PATENT APPLICATION

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION | **ATTORNEY DOCKET NO. APPT-001-2**

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:

PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE

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. 09/609179 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	Dov Rosenfeld, Reg. No. 38,687
5507 College Avenue, Suite 2	Tel: (510) 547-3378
Oakland, CA 94618	

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: Andrew A. Koppenhaver **Citizenship:** USA

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

Post Office Address: Same

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

Name of Third Inventor: James F. Torgerson **Citizenship:** USA

Residence: 227 157th Ave., NW, Andover, MN 55304

Post Office Address: Same

Inventor's Signature **Date**

PATENT APPLICATION

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION ATTORNEY DOCKET NO. APPT-001-2

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:

PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE

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. 09/609179 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**

ADDITIONAL INVENTOR SIGNATURES:

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

Residence: 10400 Kenmore Drive, Fairfax, VA 22030

Post Office Address: Same

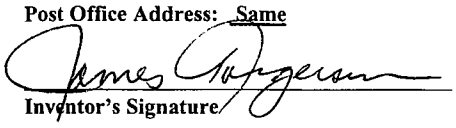
_____ _____

Inventor's Signature **Date**

Name of Third Inventor: James F. Torgerson **Citizenship:** USA

Residence: 227 157th Ave., NW, Andover, MN 55304

Post Office Address: Same



Inventor's Signature

9/27/00

Date

Our Ref./Docket No: API 001-2 Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz, *et al.* Application No.: 09/609179 Filed: June 30, 2000 Title: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE	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,

Oct 20, 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: Oct 20, 2000

Signed:

Name: Dov Rosenfeld, Reg. No. 38687

FORMALITIES LETTER

OC000000005346098

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
Address COMMISSIONER OF PATENT AND TRADEMARKS
Washington, D C 20231

APPLICATION NUMBER	FILING/RECEIPT DATE	FIRST NAMED APPLICANT	ATTORNEY DOCKET NUMBER
09/609,179	06/30/2000	Russell S. Dietz	APPT-001-2

Dov Rosenfeld
5507 College Avenue
Suite 2
Oakland, CA 94618

Date Mailed: 08/23/2000

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).*
- 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 $ 820.**

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

10/26/2000 SDENBOB1 00000068 500292 09609179
01 FC:201 40.00 DP
10/26/2000 SDENBOB1 00000068 500292 09609179
02 FC:105 130.00 DP
03 FC:101 690.00 DP
20.00 CH

Our Ref./Docket No: APP . 01-2 Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz, *et al.* Application No.: 09/609179 Filed: June 30, 2000 Title: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE	Group Art Unit: 2756 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 $__860.00__ is attached, being for:

- __X__ Statutory basic filing fee: $ 690
- __X__ Additional claim fee of $ 0
- __X__ Assignment recordation fee of $ 40
- __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: Oct 20 2000 Signed: _____

Name: Dov Rosenfeld, Reg. No. 38,687

__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,

Oct 20, 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

GP/2156
2152
+4
4-16-01

Our Docket/Ref. No.: APP[illegible]-2

Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz et al. Serial No.: 09/609179 Filed: June 30, 2000 Title: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE	Group Art Unit: 2756 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.

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: April 9, 2001

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: Apr 9 2001

Signature: _____
Dov Rosenfeld, Reg. No. 38,687

Our Docket/Ref. No.: APPT 1-2

Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz et al. Serial No.: 09/609179 Filed: June 30, 2000 Title: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE	Group Art Unit: 2756 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)

___ under 37 CFR 1.97(c) together with either a:
___ Certification under 37 CFR 1.97(e), or
___ a $180.00 fee under 37 CFR 1.17(p)
(After the CFR 1.97(b) time period, but before final action or notice of allowance, whichever occurs first)

___ under 37 CFR 1.97(d) together with a:
___ Certification under 37 CFR 1.97(e), and
___ a petition under 37 CFR 1.97(d)(2)(ii), and
___ a $130.00 petition fee set forth in 37 CFR 1.17(i)(1).
(Filed after final action or notice of allowance, whichever occurs first, but before payment of the issue fee)

_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.

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: Apr 9, 2001

Signature: _____
Dov Rosenfeld, Reg. No. 38,687

X Some of the references were cited in a search report from a foreign patent office in a counterpart foreign application. In particular, references AD, AF, AH, CI, EA, EB, EC, and ED were cited in a search report from a foreign patent office in a counterpart foreign application.

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

Respectfully submitted,

Date: April 9, 2001

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

Et al. FORM - 1449 SHEET 1 OF 5.

APR 12 2001 PATENT & TRADEMARK OFFICE USPTO

INFORMATION DISCLOSURE STATEMENT

(Use several sheets if necessary)

ATTY. DOCKET NO	SERIAL NO.
APPT-001-2	09/609179
APPLICANT: Dietz et al.	#4
FILING DATE: 6/30/2000	GROUP: ~~2756~~ 2158

RECEIVED APR 16 2001 Technology Center 2100

U.S. PATENT DOCUMENTS

*EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF APPROPRIATE
WD	AA	4736320	Apr. 5, 1988	Bristol	364	300	Oct. 8, 1985
WD	AB	4891639	Jan. 2, 1990	Nakamura	340	825.500	Jun. 23, 1988
WD	AC	5101402	Mar. 31, 1992	Chui et al.	370	17	May 24, 1988
WD	AD	5247517	Sep. 21, 1993	Ross et al.	370	85.5	Sep. 2, 1992
WD	AE	5247693	Sep. 21, 1993	Bristol	~~395~~ 709	~~800~~ 203	Nov. 17, 1992
WD	AF	5315580	May 24, 1994	Phaal	370	13	Aug. 26, 1991
WD	AG	5339268	Aug. 16, 1994	Machida	365	49	Nov. 24, 1992
WD	AH	5351243	Sep. 27, 1994	Kalkunte et. al.	370	92	Dec. 27, 1991
WD	AI	5365514	Nov. 15, 1994	Hershey et al.	370	17	Mar. 1, 1993
WD	AJ	5375070	Dec. 20, 1994	Hershey at al.	364	550	Mar. 1, 1993
WD	AK	5394394	Feb. 28, 1995	Crowther et al.	370	60	Jun. 24, 1993

FOREIGN PATENT DOCUMENTS

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

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

WD	AR	"Technical Note: the Narus System," Downloaded April 29, 1999 from www.narus.com, Narus Corporation, Redwood City California.
	AS	

EXAMINER: Khanh Dinh DATE CONSIDERED: 5/28/03

*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

Et al. FORM - 1449 SHEET 2 OF 5.

APR 12 2001 PATENT & TRADEMARK OFFICE 650

RECEIVED APR 16 2001 Technology Center 2100

INFORMATION DISCLOSURE STATEMENT

(Use several sheets if necessary)

ATTY. DOCKET NO.	SERIAL NO
APPT-001-2	09/609179
APPLICANT: Dietz et al. #4	
FILING DATE: 6/30/2000	GROUP: ~~2756~~ 2155

U.S. PATENT DOCUMENTS

*EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF APPROPRIATE
LED	BA	5414650	May 9, 1995	Hekhuis	364	715.02	Mar. 24, 1993
LD	BB	5430709	Jul. 4, 1995	Galloway	370	13	Jun. 17, 1992
LD	BC	5432776	Jul. 11, 1995	Harper	370	17	Sep. 30, 1993
LD	BD	5493689	Feb. 20, 1996	Waclawsky et al.	~~395~~ 709	~~821~~ 206	Mar. 1, 1993
LD	BE	5500855	Mar. 19, 1996	Hershey et al.	370	17	Jan. 26, 1994
LD	BF	5568471	Oct. 22, 1996	Hershey et al.	370	17	Sep. 6, 1995
LD	BG	5574875	Nov. 12, 1996	Stansfield et al.	395	403	Mar. 12, 1993
LD	BH	5586266	Dec. 17, 1996	Hershey et al.	~~395~~ 709	~~200.11~~ 216	Oct. 15, 1993
LD	BI	5606668	Feb. 25, 1997	Shwed	~~395~~ 709	~~200.11~~ 216	Dec. 15, 1993
LD	BJ	5608662	Mar. 4, 1997	Large et al.	364	724.01	Jan. 12, 1995
LD	BK	5634009	May 27, 1997	Iddon et al.	~~395~~ 709	~~200.11~~ 206	Oct. 27, 1995

FOREIGN PATENT DOCUMENTS

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

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

	BR	
	BS	

EXAMINER: Khanh Dinh DATE CONSIDERED: 5/29/03

*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.

Et al. FORM - 1449 SHEET 3 OF 5.

INFORMATION DISCLOSURE STATEMENT

(Use several sheets if necessary)

ATTY. DOCKET NO	SERIAL NO.
APPT-001-2	09/609179
APPLICANT	
Dietz et al.	#4
FILING DATE	GROUP
6/30/2000	~~2756~~ 2155

U.S. PATENT DOCUMENTS

*EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF BPPROPRIBTE
WD	CA	5651002	Jul. 22, 1997	Van Seters et all.	370	392	Jul. 12, 1995
WD	CB	5684954	Nov. 4, 1997	Kaiserswerth et al.	~~395~~ 709	~~200.2~~ 203	Mar. 20, 1993
WD	CC	5732213	Mar. 24, 1998	Gessel et al.	~~395~~ 709	~~200.11~~ 216	Mar. 22, 1996
WD	CD	5740355	Apr. 14, 1998	Watanabe et al.	395	183.21	Jun. 4, 1996
WD	CE	5761424	Jun. 2, 1998	Adams et al.	~~395~~ 709	~~200.47~~ 232	Dec. 29, 1995
WD	CF	5764638	Jun. 9, 1998	Ketchum	370	401	Sep. 14, 1995
WD	CG	5781735	Jul. 14, 1998	Southard	~~395~~ 709	~~200.54~~ 238	Sep. 4, 1997
WD	CH	5784298	Jul. 21, 1998	Hershey et al.	364	557	Jul. 11, 1996
WD	CI	5787253	Jul. 28, 1998	McCreery et al.	~~395~~ 709	~~200.61~~ 227	May 28, 1996
WD	CJ	5805808	Sep. 8, 1998	Hansani et al.	~~395~~ 709	~~200.2~~ 203	Apr. 9, 1997
WD	CK	5812529	Sep. 22, 1998	Czarnik et al.	370	245	Nov. 12, 1996

FOREIGN PATENT DOCUMENTS

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

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

	CR	
	CS	

EXAMINER	DATE CONSIDERED
Richard Dino	5/29/03

*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.

Et al. FORM - 1449

SHEET 4 OF 5.

ATTY. DOCKET NO.	SERIAL NO.
APPT-001-2	09/609179
APPLICANT: Dietz et al. #4	
FILING DATE: 6/30/2000	GROUP: 2756

INFORMATION DISCLOSURE STATEMENT

(Use several sheets if necessary)

U.S. PATENT DOCUMENTS

*EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF BPPROPRIBTE
LD	DA	5819028	Oct. 6, 1998	Manghirmalani et al.	~~395~~ 709	~~185.1~~ 203	Apr. 16, 1997
LD	DB	5825774	Oct. 20, 1998	Ready et al.	370	401	Jul. 12, 1995
LD	DC	5835726	Nov. 10, 1998	Shwed et al.	~~395~~ 709	~~200.59~~ 228	Jun. 17, 1996
LD	DD	5838919	Nov. 17, 1998	Schwaller et al.	~~395~~ 709	~~200.54~~ 208	Sep. 10, 1996
LD	DE	5841895	Nov. 24, 1998	Huffman	382	155	Oct. 25, 1996
LD	DF	5850386	Dec. 15, 1998	Anderson et al.	370	241	Nov. 1, 1996
LD	DG	5850388	Dec. 15, 1998	Anderson et al.	370	252	Oct. 31, 1996
LD	DH	5862335	Jan. 19, 1999	Welch, Jr. et al.	~~395~~ 709	~~200.54~~ 232	Apr. 1, 1993
LD	DI	5878420	Mar. 2, 1999	de la Salle	707	10	Oct. 29, 1997
LD	DJ	5893155	Apr. 6, 1999	Cheriton	711	144	Dec. 3, 1996
LD	DK	5903754	May 11, 1999	Pearson	~~395~~ 709	~~680~~ 238	Nov. 14, 1997

FOREIGN PATENT DOCUMENTS

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

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

	DR	
	DS	

EXAMINER: Khanh Dinh — DATE CONSIDERED: 5/29/03

*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.

Et al. FORM - 1449

APR 12 2001 PATENT & TRADEMARK OFFICE

SHEET 5 OF 5.

INFORMATION DISCLOSURE STATEMENT

(Use several sheets if necessary)

ATTY. DOCKET NO.	SERIAL NO.
APPT-001-2	09/609179
APPLICANT	
Dietz et al. #4	
FILING DATE	**GROUP**
6/30/2000	~~2756~~ 2150

RECEIVED APR 16 200 Technology Center 2

U.S. PATENT DOCUMENTS

*EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF BPPROPRIBTE
LD	EA	5917821	Jun. 29, 1999	Gobuyan et al.	370	392	Aug. 16, 1996
WS	EB	5414704	May 9, 1995	Spinney	370	60	Apr. 5, 1994
VB	EC	6014380	Jan 11, 2000	Hendel et al.	370	392	Jun. 30, 1997
LD	ED	5511215	Apr. 23, 1996	Terasaka et al.	~~395~~ 709	~~800~~ 246	Oct. 26, 1993
	EE						
	EF						
	EG						
	EH						
	EI						
	EJ						
	EK						

FOREIGN PATENT DOCUMENTS

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

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

	DR	
	DS	

EXAMINER	DATE CONSIDERED
Khanh Dinh	5/28/03

*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

Our Docket/Ref. No.: APPT

Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz et al. Serial No.: 09/609179 Filed: June 30, 2000 Title: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE	Group Art Unit: 2756 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.

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: April 9, 2001

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: Apr. 9 2001

Signature: _____
Dov Rosenfeld, Reg. No 38,687

#5

Our Docket/Ref. No.: APPT-001-2 Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz et al. Serial No.: 09/609179 Filed: June 30, 2000 Title: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE	Group Art Unit: 2756 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.

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 20002 Signature: _____
Dov Rosenfeld, Reg. No. 38,687

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 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

Et al.FORM - 1449 _____ _____ SHEET 1 OF 1.

ATTY. DOCKET NO.	SERIAL NO.
APPT-001-2	09/609179
APPLICANT: Dietz et al. #5	
FILING DATE: 6/30/2000	GROUP: ~~2756~~ 2155

INFORMATION DISCLOSURE STATEMENT

(Use several sheets if necessary)

U.S. PATENT DOCUMENTS

*EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF APPROPRIATE
CW	AA	5,703,877	Dec. 30, 1997	Nuber et al.	370	395	Nov. 22, 1995
CW	AB	5,826,017	Oct. 20, 1998	Holzmann	~~395~~ 709	~~200.6~~ 206	Feb. 10, 1992
	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
Khanh Dinh	5/28/03

*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

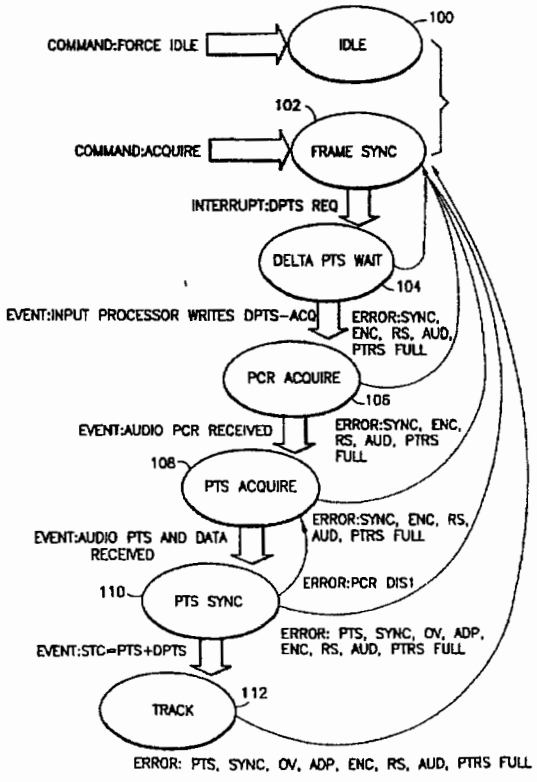
5,365,272	11/1994	Siracusa	348/461
5,376,969	12/1994	Zdepski	348/466
5,467,342	11/1995	Logston et al.	370/253
5,517,250	5/1996	Hoogenboom et al.	348/467
5,537,409	7/1996	Moriyama et al.	370/471

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



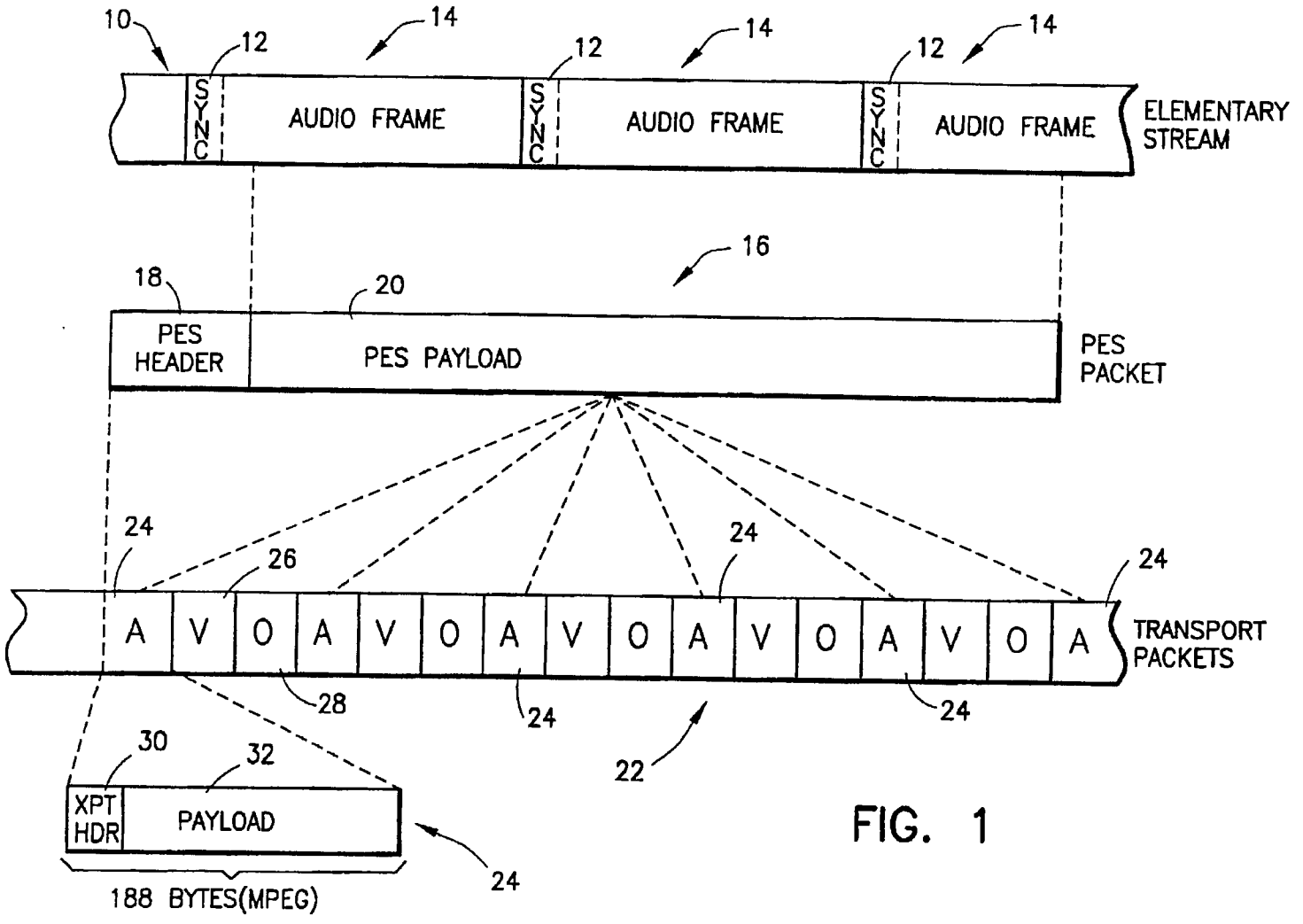

10
12
14
SYNC
AUDIO FRAME
ELEMENTARY STREAM
16
18
20
PES HEADER
PES PAYLOAD
PES PACKET
24
26
28
22
A
V
O
TRANSPORT PACKETS
30
32
XPT HDR
PAYLOAD
188 BYTES(MPEG)

FIG. 1

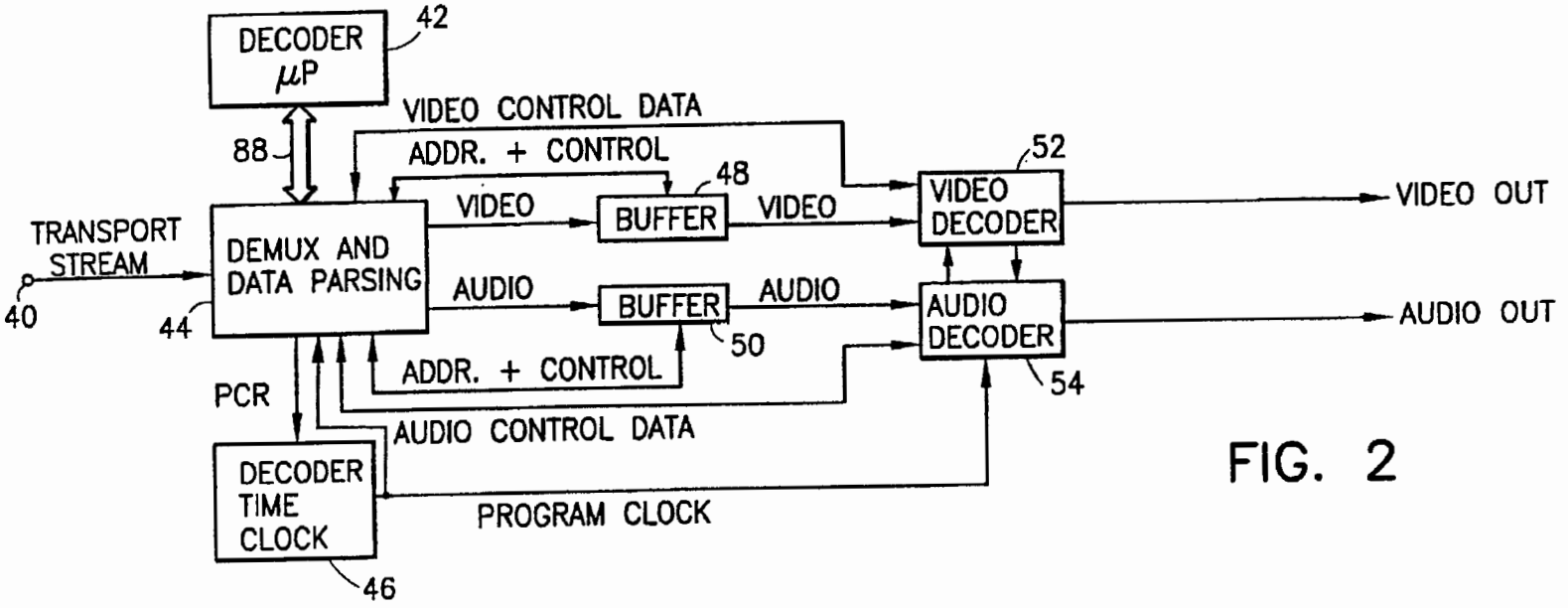

DECODER
μP
42
88
VIDEO CONTROL DATA
ADDR. + CONTROL
48
52
TRANSPORT
STREAM
40
DEMUX AND
DATA PARSING
44
VIDEO
BUFFER
VIDEO
VIDEO
DECODER
VIDEO OUT
AUDIO
BUFFER
AUDIO
AUDIO
DECODER
AUDIO OUT
50
54
ADDR. + CONTROL
PCR
AUDIO CONTROL DATA
DECODER
TIME
CLOCK
PROGRAM CLOCK
46

FIG. 2

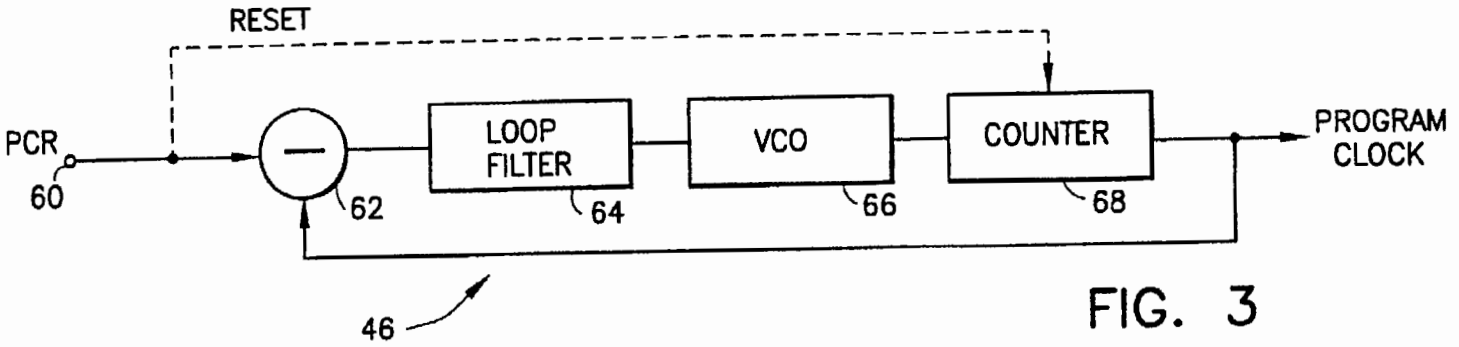
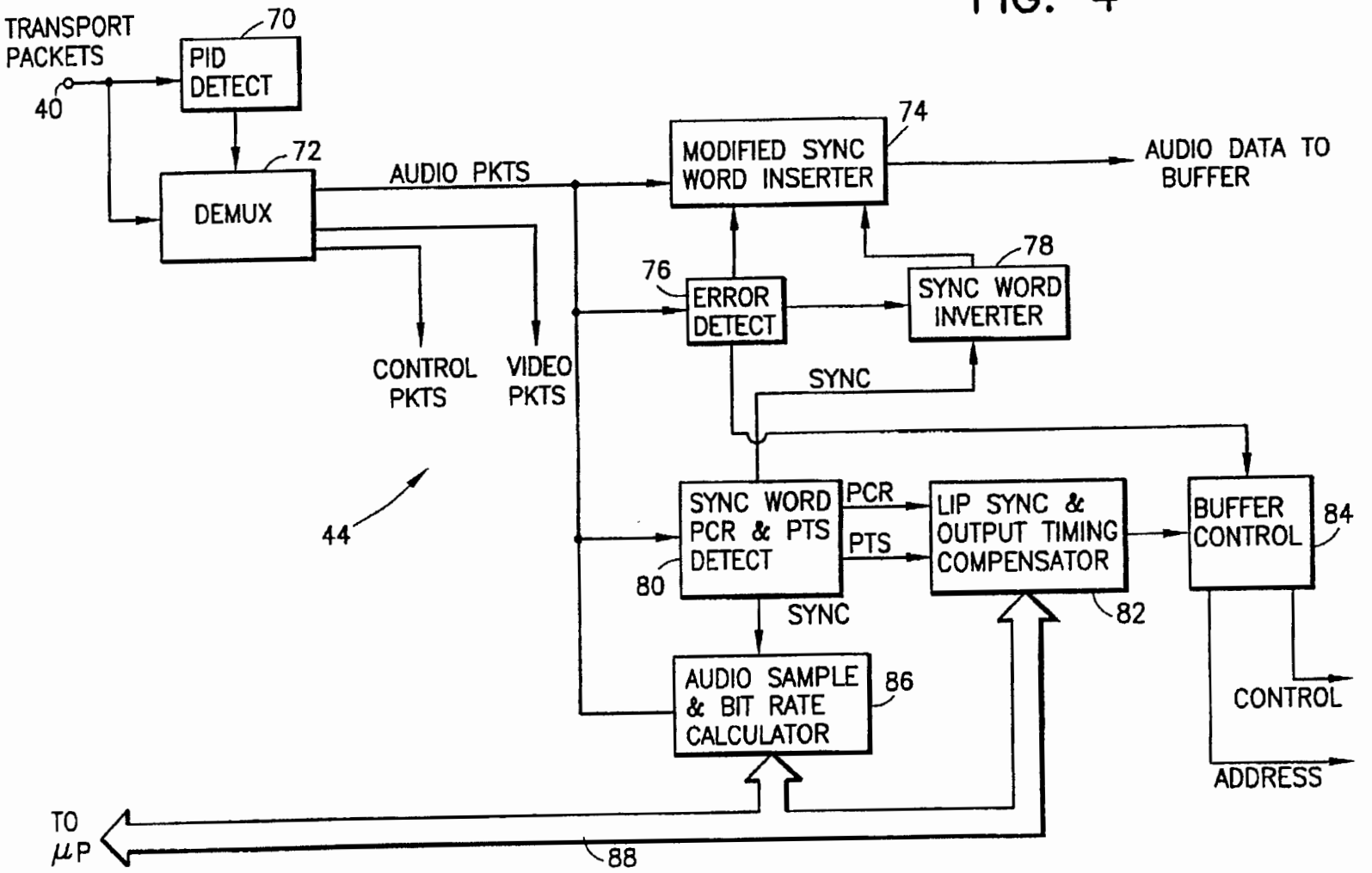

RESET
PCR
60
62
LOOP
FILTER
64
VCO
66
COUNTER
68
PROGRAM
CLOCK
46

FIG. 3

FIG. 4


TRANSPORT PACKETS
40
PID DETECT
70
DEMUX
72
AUDIO PKTS
MODIFIED SYNC WORD INSERTER
74
AUDIO DATA TO BUFFER
ERROR DETECT
76
SYNC WORD INVERTER
78
CONTROL PKTS
VIDEO PKTS
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

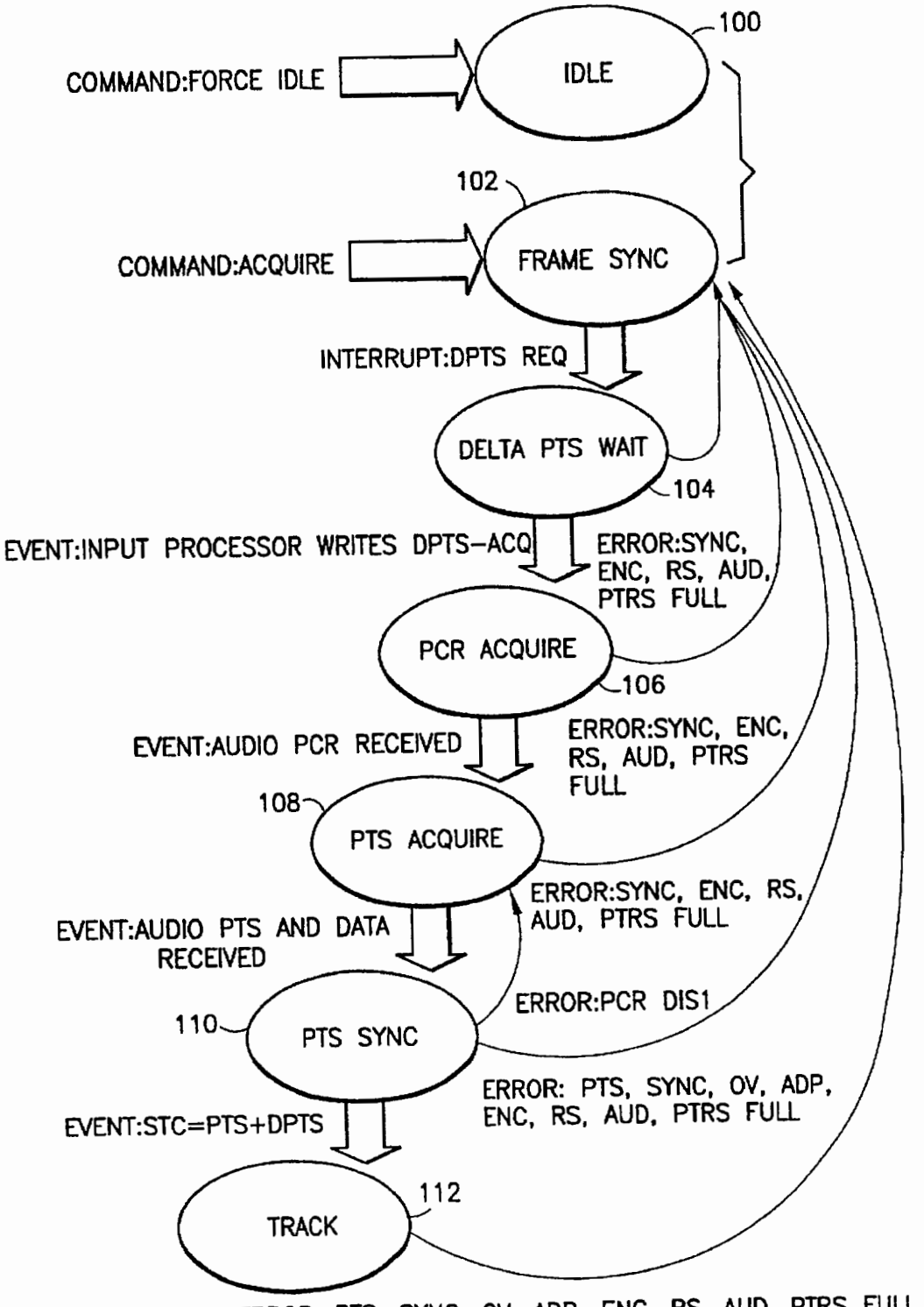

100
IDLE
COMMAND:FORCE IDLE
102
FRAME SYNC
COMMAND:ACQUIRE
INTERRUPT:DPTS REQ
DELTA PTS WAIT
104
EVENT:INPUT PROCESSOR WRITES DPTS-ACQ
ERROR:SYNC, ENC, RS, AUD, PTRS FULL
PCR ACQUIRE
106
EVENT:AUDIO PCR RECEIVED
ERROR:SYNC, ENC, RS, AUD, PTRS FULL
108
PTS ACQUIRE
ERROR:SYNC, ENC, RS, AUD, PTRS FULL
EVENT:AUDIO 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
pta_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.
fa_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 processing 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.

* * * * *

US005826017A

United States Patent [19]

Holzmann

[11] Patent Number: 5,826,017

[45] Date of Patent: Oct. 20, 1998

[54] **APPARATUS AND METHOD FOR COMMUNICATING DATA BETWEEN ELEMENTS OF A DISTRIBUTED SYSTEM USING A GENERAL PROTOCOL**

[75] Inventor: **Gerard Johan Holzmann**, Murray Hill, N.J.

[73] Assignee: **Lucent Technologies**, Murray Hill, N.J.

[21] Appl. No.: **830,291**

[22] Filed: **Feb. 10, 1992**

[51] **Int. Cl.**[6] **G06F 11/00**

[52] **U.S. Cl.** **395/200.6**; 395/285; 395/500; 370/401; 370/428; 370/465; 370/466; 370/467; 370/468; 370/469

[58] **Field of Search** 398/200, 500, 398/225; 370/401, 467, 428, 465, 466, 469; 395/200.6, 200.5, 500

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,545,052	10/1985	Steierman	370/68
4,688,170	8/1987	Waite et al.	395/500
4,733,357	3/1988	Peirent	364/200
4,754,400	6/1988	Wakahara et al.	395/180
4,855,905	8/1989	Estrada et al.	364/200
4,970,716	11/1990	Goto et al.	370/58.1
5,063,494	11/1991	Davidowski et al.	395/800
5,142,528	8/1992	Kobayashi et al.	370/79
5,163,055	11/1992	Lee et al.	371/32
5,175,817	12/1992	Adams et al.	395/200
5,182,748	1/1993	Sakata et al.	370/94.1
5,245,703	9/1993	Hubert	395/200
5,276,802	1/1994	Yamaguchi et al.	395/164
5,276,816	1/1994	Cavendish et al.	345/348
5,278,972	1/1994	Baker et al.	395/500
5,313,467	5/1994	Vargheses et al.	370/94.1
5,347,524	9/1994	I'Anson et al.	371/29.1
5,430,727	7/1995	Callon	370/85.13
5,452,433	9/1995	Nihart et al.	395/500
5,557,798	9/1996	Skeen et al.	395/650
5,574,919	11/1996	Netravali et al.	395/561
5,581,558	12/1996	Horney, II et al.	370/401
5,594,721	1/1997	Pan	370/392
5,623,666	4/1997	Pike et al.	707/200
5,659,555	8/1997	Lee et al.	371/27.1
5,680,552	10/1997	Netravali et al.	395/200.2

FOREIGN PATENT DOCUMENTS

EP-A-0289248	4/1988	European Pat. Off.	G06F 13/38

OTHER PUBLICATIONS

Gerard J. Holzmann, "Standardized Protocol Interfaces", AT&T Bell Labs, Murray Hill, N.J., Oct. 18, 1992.

J. E. Boillat, et al "Communication Protocols and Concurrency: An OCCAM Implementation of X.25", 1988 International Zurich Seminar on Digital Communications, Mar. 8, 1988, pp. 99–102.

M. H. Sherif, et al "Evaluation of Protocols from Formal Specifications: A Case Study with LAPD", IEEE Global Telecommunications Conf., vol. 3, Dec. 2, 1990, San Diego, pp. 879–886.

Chesson, G., "The Protocol Engine Project", Unix Review, Sep. 1987.

Toong, H.–M., "Microprocessors", Sci. Am., vol. 237, No. 3, p. 146, Sep. 1977.

Primary Examiner—Christopher B. Shin

[57] **ABSTRACT**

Apparatus and methods for communicating using protocols. The apparatus and methods employ protocol descriptions written in a device-independent protocol description language. A protocol is executed by employing a protocol description language interpreter to interpret the protocol description. Communication using any protocol for which there is a protocol description may be done by means of a general protocol. The general protocol includes a first general protocol message which includes a protocol description for a specific protocol. The protocol apparatus which receives the first protocol message employs a protocol description language interpreter to interpret the included protocol description and thereby to execute the specific protocol. The protocol apparatus may also be made to adapt to its environment by encaching protocol descriptions which were received in an earlier first general protocol message and interpreting an encached protocol description in response to a second general protocol message which includes a protocol identifier specifying the encached protocol description.

46 Claims, 8 Drawing Sheets

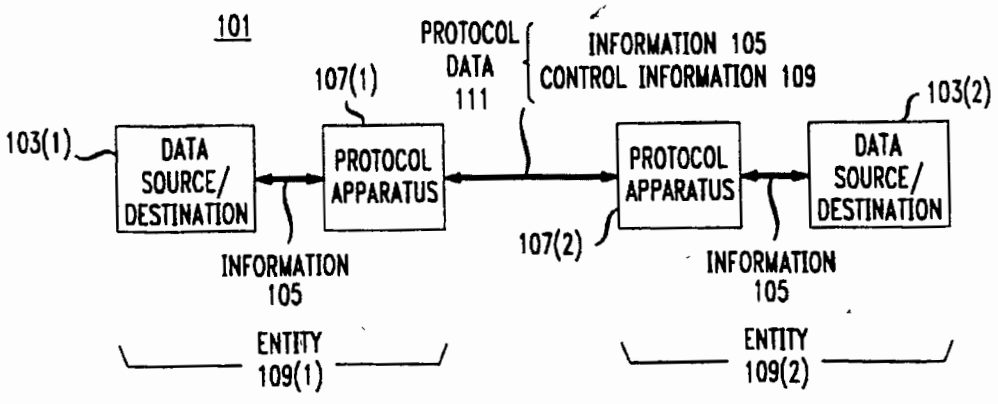


FIG. 1

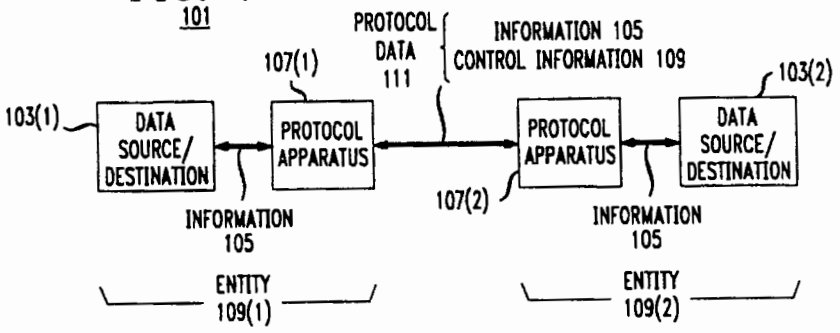


FIG. 2

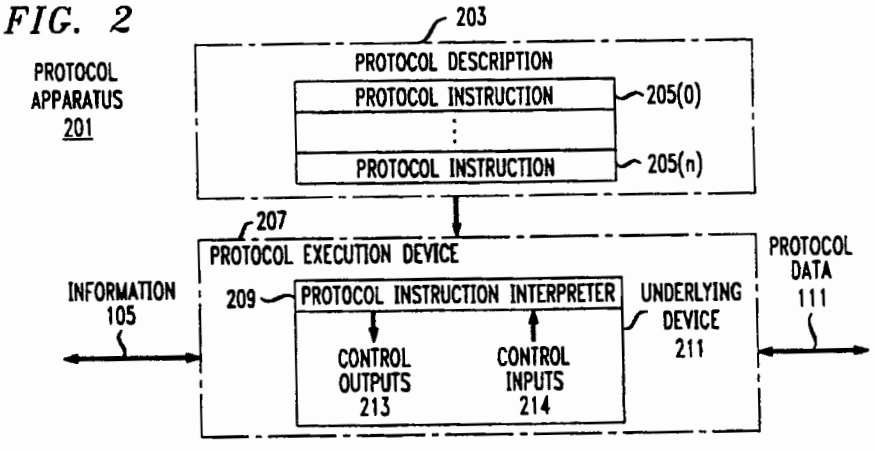


FIG. 3

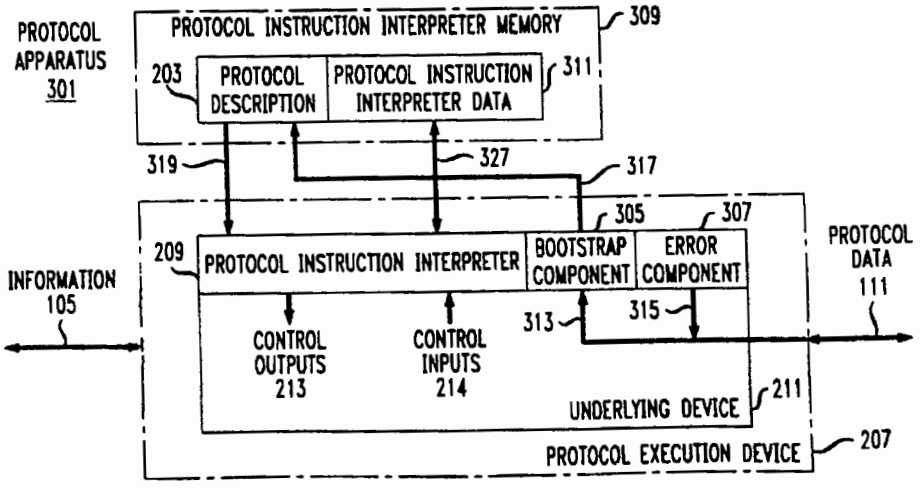


FIG. 4

401

INSTRUCTION SET - GENERAL 403

0. STACK MANAGEMENT AND EXPRESSION EVALUATION	
PLUS,MINUS,UMIN,TIMES,DIVIDE,MODULO	arithmetic operations
AND,OR,GT,LT,GE,LE,EQ,NE,NOT	boolean operations
SHIFTL,SHIFTR,BIT_AND,BIT_OR,XOR,BIT_COMPL	bitwise operators
I_PUSH_BYTE,I_PUSH_WORD	push a constant onto the stack
I_PUSH_BYTE_VAR,I_PUSH_WORD_VAR	push a variable onto the stack, constant operands
PUSH_BYTE_VAR,PUSH_WORD_VAR	push a variable onto the stack

INSTRUCTION SET - PROTOCOL RELATED 405

MNEMONIC	PARAMETERS		PURPOSE
		1. FSM CONTROL	
421 LOAD	2(buffer nr,state nr)	407	assign a new state definition
423 NXT	1(state nr)		perform a state transition
425 IF,ELSE	1(value)		conditional execution of commands
		2. UPPER INTERFACE	
ACCEPT	1(buffer nr)	409	pass a message to the upper layer
OBTAIN	1(buffer nr)		fetch a new message from the upper layer
		3. LOWER INTERFACE	
RECV	1(buffer nr)	411	receive a message from the lower layer into buffer
SEND	1(buffer nr)		send a message to the lower layer from buffer
413 CKSUM	1(buffer nr)		calculate a checksum on buffer contents
415 BYTEORDER	1(constant)	419	define byte-order of lower layer
WORD_SZ	1(constant)		number of bytes per word on lower layer
417		4. BUFFER MANAGEMENT	
ALLOC	2(buffer,maxsize)		allocate bufferspace for buffer
SETSIZE	2(buffer,size)		define the length field of a message in buffer
SETTIMO	2(buffer,time)		define a timeout period for a message in buffer
CPY_BYTE	3(buffer,index,value)		set a byte-value in a message
CPY_WORK	2(buffer,start_index,value)		set a work (N bytes)

FIG. 5

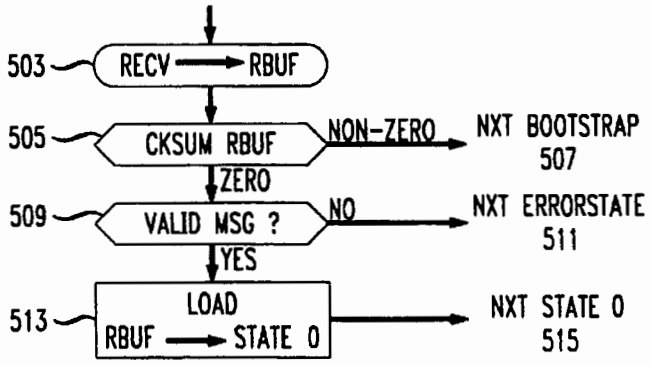

501
503
RECV → RBUF
505
CKSUM RBUF
NON-ZERO
NXT BOOTSTRAP
507
ZERO
509
VALID MSG ?
NO
NXT ERRORSTATE
511
YES
513
LOAD
RBUF → STATE 0
NXT STATE 0
515

FIG. 6

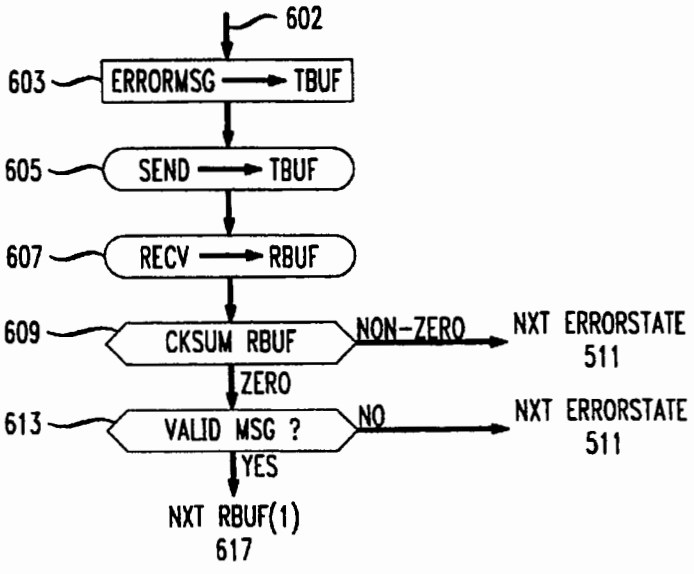

601
602
603
ERRORMSG → TBUF
605
SEND → TBUF
607
RECV → RBUF
609
CKSUM RBUF
NON-ZERO
NXT ERRORSTATE
511
ZERO
613
VALID MSG ?
NO
NXT ERRORSTATE
511
YES
NXT RBUF(1)
617

FIG. 7

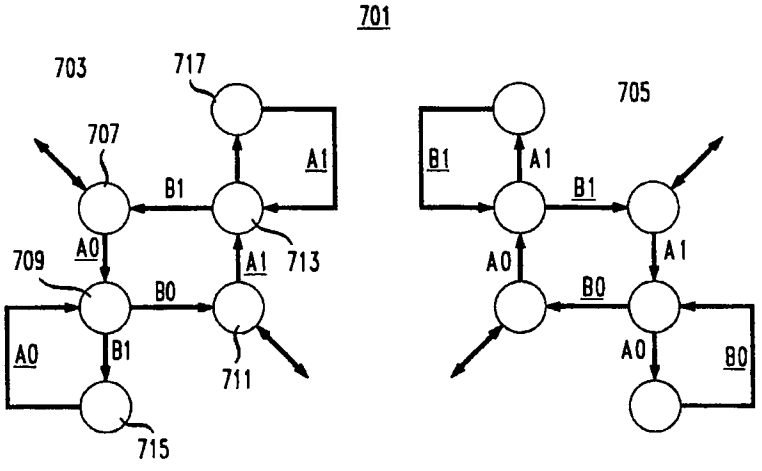

701
703
717
707
A1
B1
A0
A1
713
709
B0
A0
B1
711
715
705
B1
A1
B1
A0
A1
B0
A0
B0

FIG. 13

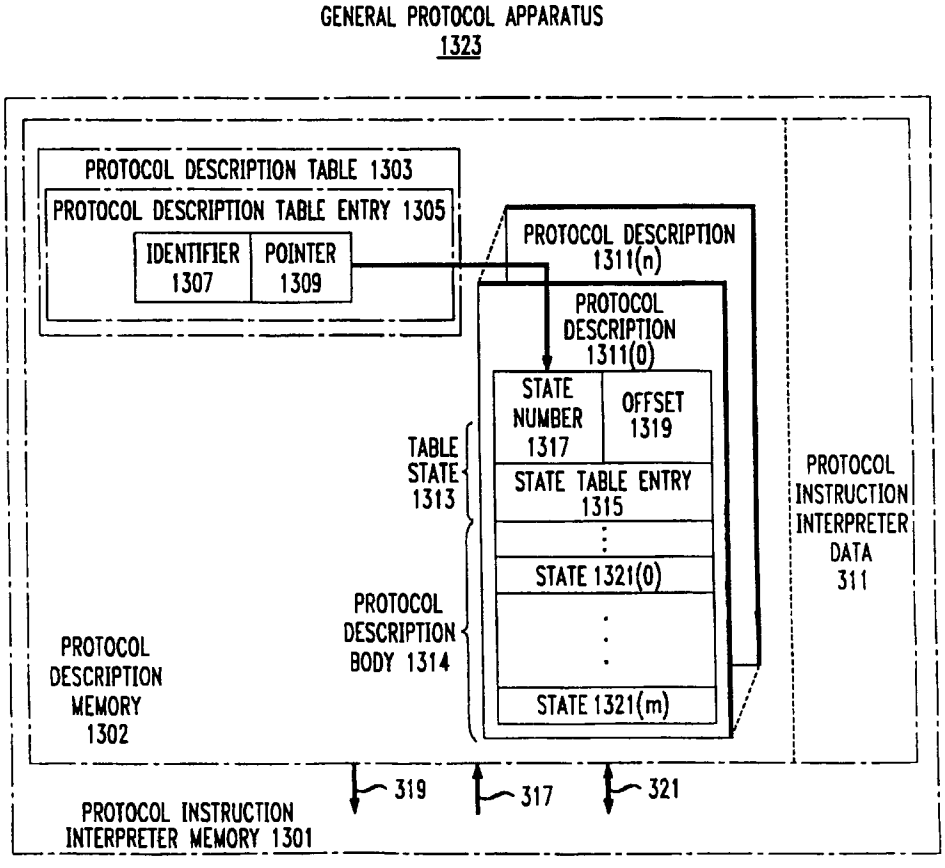

GENERAL PROTOCOL APPARATUS
1323
PROTOCOL DESCRIPTION TABLE 1303
PROTOCOL DESCRIPTION TABLE ENTRY 1305
IDENTIFIER 1307
POINTER 1309
PROTOCOL DESCRIPTION 1311(n)
PROTOCOL DESCRIPTION 1311(0)
STATE NUMBER 1317
OFFSET 1319
TABLE STATE 1313
STATE TABLE ENTRY 1315
STATE 1321(0)
PROTOCOL DESCRIPTION BODY 1314
STATE 1321(m)
PROTOCOL INSTRUCTION INTERPRETER DATA 311
PROTOCOL DESCRIPTION MEMORY 1302
319
317
321
PROTOCOL INSTRUCTION INTERPRETER MEMORY 1301

FIG. 8

801

```
/* Receiver Buffers */
#define RBUF      0     /* receive buffer */
#define TBUF      1     /* transmit buffer */
#define VAR_E     2     /* variable 'e' - receiver side    */
/* Transmitter Buffers */
#define M0        0     /* message m0 */
#define M1        1     /* message m1 */
803 #define R_run     2     /* Abp_rcv_run      */
#define R_ini     3     /* Abp_rcv_ini      */
#define R_ack     4     /* receive buffer - for acks      */
#define VAR_S     5     /* variable 's' - sender side     */
#define VAR_CNT   6     /* variable 'cnt' - sender side   */
#define B_O       1     /* byteorder */
#define NR_MSGS 32765 /* number of test messages sent */
BYTE Abp_rcv_ini[] = {        /* initialization Buf[r_ini]; recvd in State[0] */
/*0*/  BYTEORDER,      B_O,
/*2*/  I_ALLOC,        TBUF, 2,
/*5*/  I_SETSIZE,      TBUF, 2,
/*8*/  I_ALLOC,        VAR_E, 1
805 /*11*/ I_RECV,        RBUF,
/*13*/ I_LOAD,        1, RBUF,/* input becomes State[1] */
/*16*/ I_NXT,         1,      /* execute it */
/*18*/ 0,             0,      /* room for the checksum; required on 1st msg */
};
BYTE Abp_rcv_run[] = {  /* abp receiver Buf[R_run]; recvd in State[1] */
/*0*/  I_RECV,           RBUF,
/*2*/  I_CPY_BYTE,       TBUF,    0,       'A',
/*6*/  I_PUSH_BYTE_VAR,          RBUF,    1,
/*9*/  H_CPY_BYTE,       TBUF, 1,
/*12*/ I_SEND,           TBUF,
/*14*/ I_PUSH_BYTE_VAR,          RBUF,    1,
/*17*/ I_PUSH_BYTE_VAR,          VAR_E,   0,
/*20*/ EQ,
/*21*/ IF,        34,            /* e == rbuf[1] */
807 /*23*/          I_PUSH_BYTE,   1,
/*25*/          I_PUSH_BYTE_VAR,VAR_E,   0,
/*28*/          MINUS,
/*29*/          H_CPY_BYTE,     VAR_E,   0,
/*32*/          I_ACCEPT,       RBUF,
/*34*/ I_NXT,    1              /* stay in same state */
};
BYTE Msg0[] = { /* message Buf[M0], received in Buf[RBUF] */
        'M', 0
};
809 BYTE Msg1[] = { /* message Buf[M1], received in Buf[RBUF] */
        'M', 1
};
```

FIG. 9 901

```
/* sender behavior */
BYTE Abp_snd_ini[] = {                    /* becomes State[0] */
/*0*/   I_ALLOC,       VAR_S, 1,          /* the byte variable  's' */
/*3*/   I_CPY_BYTE,    VAR_S,0,0,         /* s = 0 */
/*7*/   I_ALLOC,       VAR_CNT,2,         /* the word variable 'cnt' */
/*10*/  I_CPY_WORD,    VAR_CNT, 0, 0,     /* cnt = 0 */
/*14*/  I_SEND,        R_ini,             /* send Buf[R_ini] == Abp_rcv_ini */
/*16*/  I_SEND,        R_run,             /* send Buf[R_run] == Abp_rcv_run */
/*18*/  I_NXT,         1                  /* begin actual behavior */
};

BYTE Abp_snd_run[] = {                    /* becomes State [1] */
/*0*/   I_PUSH_BYTE_VAR, VAR_S, 0,
/*3*/   SEND,                             /* send from buf[s] */
/*4*/   I_RECV,          R_ack,           /* recv into buf[r_ack] */
/*6*/   I_PUSH_BYTE_VAR,R_ack, 0,         /* look at Buf[R_ack].cont[0] */
/*9*/   I_PUSH_BYTE,     'A',
/*11*/  EQ,
/*12*/  IF,              32,              /* #12 - #13 */
/*14*/          I_PUSH_BYTE_VAR, R_ack, 1,      /* Buf[R_ack].cont[1] */
/*17*/          I_PUSH_BYTE_VAR, VAR_S, 0,      /* s */
/*20*/          EQ,
/*21*/          IF,       32,     /* #21 - #22 */
/*23*/                    I_PUSH_BYTE,   1,
/*25*/                    I_PUSH_BYTE_VAR, VAR_S, 0,
/*28*/                    MINUS,
/*29*/                    H_CPY_BYTE,    VAR_S, 0, /* s = 1 - s */
/*32*/ I_PUSH_BYTE,      1,      /* #32 - #33 */
/*34*/ I_PUSH_WORD_VAR,VAR_CNT, 0,
/*37*/ PLUS,
/*38*/ H_CPY_WORD,       VAR_CNT, 0,            /* cnt = 1 + cnt */
/*41*/ I_PUSH_WORD_VAR,VAR_CNT, 0,
/*44*/ I_PUSH_WORD,      NR_MSGS>>8,  NR_MSGS&255,
/*47*/ GE,                        /* cnt >= NR_MSGS */
/*48*/ IF,                54,     /* #48 - #49 */
/*50*/                    I_PUSH_WORD,  255, 255,        /* -1 = exit */
/*53*/                    NXT,
/*54*/ I_NXT,             1       /* #54 - #55 stay in this state */
};
```

903

905

907

FIG. 10

1001

```
                    1003
transition(n)
{       BYTE b1, *cur_state = State[n].cont;
        static   int Stack[SMAX+1];
        register BYTE *prot;  ——————— 1005
        register int w0, w1, w2, i;
        register int *sp = &Stack[SMAX];

        prot = cur_state;  ——— 1006
        while (prot) {          ——— 1009
1007 ———        switch (*prot++) {

                /***** FSM CONTROL *****/

                case NXT:
                        assert(sp < &Stack[SMAX]);
                        w0 = POP;
        1011            debug("next %d0, w0, 0, 0, 0);
                        if (w0 < 0 || w0 > SMAX || !State[w0].cont)
                                return ERRORSTATE;
                        return w0;
                case I_NXT:
                        w0 = *prot++;
                        debug("next %d0, w0, 0, 0, 0);
        1013            if (w0 < 0 || w0 > SMAX || !State[w0].cont)
                                return ERRORSTATE;
                        return w0;
                  . . .

                default:
                1015    debug("Error <%d>0, *prot, 0, 0, 0);
                        return ERRORSTATE;
        }       }
}
```

FIG. 11

1101

```
BYTE    Bootstrap[] = {
/*0*/   I_RECV, RBUF,              /* receive message into RBUF        */
/*2*/   I_CKSUM, RBUF,             /* always checksum initial msg      */
/*4*/   IF, 10,                    /* if nonzero goto instruction #10  */
/*6*/   I_PUSH_WORD, BOOTSTRAP>>8, BOOTSTRAP&255,
/*9*/   NXT,            /* stay in bootstrap state                     */
/*10*/  I_PUSH_BYTE_VAR, RBUF, 0,  /* get variable Buf[RBUF].cont[0]   */
/*13*/  I_PUSH_BYTE, BYTEORDER,
/*15*/  NE,                        /* Buf[RBUF].cont[0] != BYTEORDER   */
/*16*/  IF, 22,                    /* if false goto instruction #22    */
/*18*/  I_PUSH_WORD, ERRORSTATE>>8, ERRORSTATE&255,
/*21*/  NXT,
/*22*/  I_LOAD, 0,        RBUF,    /* it checks out, define State[0]   */
/*25*/  I_NXT,  0                  /* and execute it                   */
};
```

FIG. 12

1201

```
BYTE    Errorstate[] = {
/*0*/   I_ALLOC,         TBUF, 1,
/*3*/   I_SETSIZE,       TBUF, 1,
/*6*/   I_CPY_BYTE,      TBUF, 0, ERRORMSG,
/*10*/  I_SEND, TBUF,
/*12*/  I_RECV, RBUF,
/*14*/  I_CKSUM,         RBUF,
/*16*/  IF, 22,                    /* if nonzero move to instruction #22 */
/*18*/  I_PUSH_WORD, ERRORSTATE>>8, ERRORSTATE&255,
/*21*/  NXT,
/*22*/  I_PUSH_BYTE,VAR,RBUF, 0,
/*25*/  I_PUSH_BYTE,     NXT,
/*27*/  EQ,                        /* Buf[RBUF].cont[0] == NXT         */
/*28*/  IF, 18,                    /* if false move to instruction #18 */
/*30*/  I_PUSH_WORD_VAR, RBUF, 1,
/*33*/  NXT
};
```

APPARATUS AND METHOD FOR COMMUNICATING DATA BETWEEN ELEMENTS OF A DISTRIBUTED SYSTEM USING A GENERAL PROTOCOL

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention pertains to communications in distributed systems generally and specifically to protocols.

2. Description of the Prior Art

A computer protocol is a set of rules that governs the interaction of concurrent processes in a distributed computing system. For example, if computer A, connected to a disk drive containing files, wishes to print out one of the files on a printer connected to computer B, it can do so only if computer A has agreed with computer B on a protocol for doing so. The protocol must define matters such as the following:

- How does computer A ask computer B whether the printer is available?
- How does computer B tell computer A that the printer is or is not available?
- How does computer A tell computer B that it is starting to send data?
- How does computer B tell computer A to slow down the speed at which computer A is sending data or to stop sending?
- How does computer B tell computer A to resume sending?
- How does computer A tell computer B it is done sending?
- How does computer B tell computer A that it is done printing?

A general discussion of computer protocols may be found in Gerard J. Holzmann, *Design and Validation of Computer Protocols*, Prentice Hall, Englewood Cliffs, N.J. 1991.

Among the difficulties of implementing computer protocols are those which are consequences of the fact that the entities which execute a protocol are often different. For example, computer A and computer B of the foregoing example may be different kinds of machines. In other cases, the entities executing the protocol may be programs written in different programming languages. Because each of the entities which cooperate to execute the protocol must contain an implementation of at least its part of the protocol, there will be as many different implementations of at least parts of the protocol as there are different cooperating entities.

One of the difficulties which arises from this situation is the need to reimplement each protocol for each kind of entity which executes it. As the number of protocols and kinds of entities grows, more and more implementation effort is involved. An even more important difficulty is caused by the fact that the implementations of the same protocol for different entities are often done by different people; if the different people have different understandings of the protocol, the implementations may not be completely compatible and it will be hard to determine where they are incompatible and what the effects of any incompatibility will be.

The apparatus and methods disclosed in the following overcome these problems and others by permitting all entities which execute a protocol to execute the same description of the protocol.

SUMMARY OF THE INVENTION

In one aspect of the invention, the invention is a peripheral apparatus for communicating information using a protocol. The apparatus includes

means for transferring the information according to the protocol;

means for transferring the information between the peripheral apparatus and a host device;

means independent of the host device for storing a protocol description which describes the protocol and which employs a protocol description language which is independent of any particular implementation of the peripheral apparatus; and

protocol description interpretation means which is independent of the host device and which is capable of interpreting the protocol description language for interpreting the protocol description as required to transfer the information according to the protocol via the means for transferring the information according to the protocol and to transfer the information via the means for providing the information to the host device.

In another aspect, the invention is a method of communicating in a distributed system. The method includes the steps of:

in a first entity of the distributed system,

receiving a first general protocol message which includes a protocol description which describes a specific protocol, the protocol description employing a protocol description language which is independent of any particular implementation of the first entity; and

responding to the first general protocol message by employing first protocol description interpretation means capable of interpreting the protocol description language to interpret the protocol description as required to communicate using the specific protocol.

In still another aspect, the invention is protocol apparatus for communicating in a distributed system, the apparatus including:

means for receiving a first general protocol message, the first general protocol message including a protocol description which describes a specific protocol and which employs a protocol description language which is independent of any particular implementation of the protocol apparatus; and

means for responding to the first general protocol message which are capable of interpreting the protocol description language and which interpret the protocol description as required to communicate using the specific protocol.

In a further aspect, the invention is apparatus for communicating in a distributed system, the apparatus including:

first protocol apparatus for communicating using a general protocol and

second protocol apparatus for communicating using the general protocol,

the first protocol apparatus including

means for providing a first general protocol message which includes a protocol description which describes a specific protocol and which employs a protocol description language which is independent of any particular implementation of the second protocol apparatus; and

means for employing the specific protocol to communicate with the second protocol apparatus after providing the first general protocol message; and

the second protocol apparatus including

means for receiving the first general protocol message from the first protocol apparatus; and

means for responding to the first general protocol message which are capable of interpreting the protocol description language and which interpret the protocol description as required to communicate using the specific protocol.

The foregoing and other aspects, objects and advantages of the invention will be apparent to one of ordinary skill in the art who peruses the following Drawings and Detailed Description, wherein:

BRIEF DESCRIPTION OF THE DRAWING

FIG. **1** is a block diagram of a typical system in which protocols are used;

FIG. **2** is a block diagram of a first apparatus incorporating the invention;

FIG. **3** is a block diagram of a second apparatus incorporating the invention;

FIG. **4** is a table of instructions in a protocol description language;

FIG. **5** is a flowchart of the bootstrap state;

FIG. **6** is a flowchart of the error state;

FIG. **7** is a state diagram for the alternating bit protocol;

FIG. **8** is a protocol description for the receive side of the alternating bit protocol;

FIG. **9** is a protocol description for the send side of the alternating bit protocol;

FIG. **10** is a fragment of the transition procedure;

FIG. **11** is a protocol description for the bootstrap state;

FIG. **12** is a protocol description for the error state; and

FIG. **13** is a block diagram of an embodiment which encaches down-loaded protocol descriptions.

The reference numbers employed in the Drawing and the Detailed Description have three or more digits. The two least significant digits are a number within a figure; the remaining digits are the figure number. Thus, the element with the reference number "**305**" is first shown in FIG. **3**.

DETAILED DESCRIPTION

The following Detailed Description will first provide an overview of the techniques of the invention and will then disclose in detail how the Alternating Bit Protocol (described at Holzmann, supra, pp. 75–77) may be implemented using the techniques of the invention.

Overview: FIGS. 1–3

FIG. **1** is a block diagram of a system **101** in which protocols are used for communication between a first entity **109(1)** and a second entity **109(2)**. Each entity **109** includes a source or destination **103** for information (INFO) **105** and a protocol apparatus **107**.

As shown by the arrows, when entity **109(1)** is communicating with entity **109(2)**, information **105** goes from source **103(1)** to protocol apparatus (PA) **107(1)**, which employs a protocol as described above to transfer information **105** to protocol apparatus **107(2)**. As explained above, the connection between protocol apparatus **107(1)** and protocol apparatus **107(2)** carries not only information **105**, but also the control information **109** which protocol apparatus **107(1)** and **107(2)** require to carry out the protocol. The information **105** and control information **109** together make up protocol data (PDATA) **111**. Protocol apparatus **107(2)** then provides the information **105** which it receives to destination **103(2)**. When entity **109(2)** communicates with entity **109(1)**, information **105** from source **103(2)** goes to protocol apparatus **107(2)**, protocol apparatuses **107(1)** and **107(2)** carry out the protocol, which this time transfers the information from protocol apparatus **107(2)** to protocol apparatus **107(1)**, and protocol apparatus **107(1)** provides the information to destination **103(1)**.

A system of the type shown in FIG. **1** may be built in many different fashions and may be used in many environments. For example, protocol apparatus **107(1)** and **107(2)** may be connected by any kind of communications medium, including parallel and serial buses, telecommunications media and shared memory. Further, entities **109** may be processes running in a single system or processes running in different systems. Further, the communication may be between different levels of the same system or between different systems. Finally, the apparatus **107** may be implemented as a process executing in a multiprocess system or in special purpose hardware, or as some combination of these alternatives.

Because it is the purpose of a protocol to communicate between different entities **109** and protocol apparatus **107** is in each case part of the entity **109**, it is almost always the case that protocol apparatus **107(1)** and protocol apparatus **107(2)** are implemented by different individuals. That fact has important consequences. As explained in Holzmann, supra, it is extremely difficult to provide a description of a protocol which is both complete and unambiguous. When the description is incomplete or ambiguous, different individuals will implement protocol apparatus **107** which execute different versions of the protocol, and if two protocol apparatuses **107** which execute different versions of the protocol attempt to communicate using the protocol, the communication may fail. Worse, because the failures are the results of different interpretations of the protocol description, the manner of failure will be unpredictable and therefore cannot be taken into account in the design of the protocol. While a complete and unambiguous protocol description can reduce the problem, it does not eliminate it: the individuals implementing apparatus **107** can still have different understandings of the protocol description, and their implementations of apparatus **107** will reflect their understandings. Again, the result is the implementation of protocol apparatuses **107** which execute different versions of the protocol, and again, there is the risk that communications between entities employing such apparatuses **107** will fail.

FIG. **2** illustrates a protocol apparatus **201** which solves the foregoing problem. Protocol apparatus **201** has two main components: protocol description **203** and protocol execution device **207**. Protocol description **203** is a protocol description which is written using protocol instructions **205** belonging to a protocol description language. The protocol description language is independent of any particular hardware or software implementation of protocol apparatus **201**. There is a single protocol description **203** for the protocol, and every protocol apparatus **201** has a copy of part or all of the single protocol description **203**. Protocol execution device **207** executes the protocol by executing the protocol instructions in protocol description **203**. The protocol instructions **205** are executed by means of protocol instruction interpreter **209**. Protocol instruction interpreter **209** can interpret all of the instructions belonging to the protocol description language. As it interprets each instruction, it produces control outputs **213** to underlying device **211**, which actually receives information **105** and provides protocol data **111**. Underlying device **211** may in turn provide control inputs **214** to protocol language interpreter **209**. Underlying device **211** may be implemented in software, in hardware, or in a combination. Depending on how underlying device **211** is implemented, control outputs **213** may

include procedure calls or subroutine addresses, interprocess communications, instructions for a processor in underlying device **211**, or control outputs to hardware devices. In the latter case, protocol execution device **207** may be a specialized microprocessor which executes instructions in the protocol description language. Again depending on how underlying device **211** is implemented, control inputs **214** may include data returned by a procedure or a subroutine, data returned by an interprocess communication, the results of executions of instructions, or interrupts.

An implementation of protocol apparatus **201** which is particularly advantageous is an implementation as a peripheral device for a source or destination **103** such as a host computer. Such an implementation would be connected between the medium over which protocol data **111** is to be transferred and a bus of the host computer and would include its own memory for storing protocol description **203** and its own protocol execution device **207**. In such an implementation, protocol execution device **207** might be implemented as a processor which is capable of directly executing protocol instructions **205**. A particularly advantagous form of such a peripheral device would be one which was implemented in a single integrated circuit.

Protocol apparatus **201** has numerous advantages over protocol apparatus **107**. First, every protocol apparatus **201** uses a copy of a single protocol description **203**; thus, there is no possibility that different implementations of protocol apparatuses **201** will implement different versions of the protocol. Second, protocol description **203** is written only once, but will be used many times. It is therefore worthwhile to expend great efforts to ensure that protocol description **203** is in fact a correct, complete and unambiguous description of the protocol. Third, the part of protocol apparatus **201** which may differ in the different implementations is protocol execution device **207**. However, protocol execution device **207** must now only be able to correctly execute the protocol instructions **205** in protocol description **203**. That is, the problem is no longer the correct implementation of the protocol, but rather the correct implementation of an instruction set in a single device. This problem is, however, far better understood than the problem of implementing a protocol in two devices, and consequently, implementations of the protocol instruction set in different protocol apparatuses **201** are far more likely to be correct than implementations of the protocol itself.

Apparatus for Executing a General Protocol: FIGS. **3** and **13**

Perhaps the most significant advantage of protocol apparatus **201** is that it can execute any protocol for which there is a protocol description **203** written in the protocol description language. Consequently, protocol apparatus **201** can easily be modified to make a general protocol apparatus which executes a general protocol and which can therefore dynamically execute any protocol for which there is a protocol description **203**. The general protocol is simply the following:

- in a sending protocol apparatus, sending a general protocol message which includes a protocol description **203**;
- in a receiving general protocol apparatus, employing protocol instruction interpreter **209** to execute the protocol description **203** contained in the general protocol message.

FIG. **3** shows such a general protocol apparatus **301**. General protocol apparatus **301** includes protocol instruction interpreter memory (PIIM) **309**, which contains protocol description **203** for the protocol currently being executed by protocol apparatus **301** and protocol instruction interpreter data (PIIDATA) **311**, which is data employed by protocol instruction interpreter **209** in executing protocol description **203**. Protocol interpreter **209** has two additional components: bootstrap component (BOOT) **305** and error component (ERR) **307**. These components make it possible for general protocol apparatus **301** to execute the general protocol, and thereby make it possible for any protocol apparatus **107** which can provide a protocol description **203** to protocol apparatus **301** to use the protocol described in the protocol description **203** to communicate between the entities **109** to which protocol apparatus **107** and protocol apparatus **301** belong. Of course, both protocol apparatuses involved in the communication may be general protocol apparatuses **301**.

Protocol apparatus **301** executes the general protocol as follows: bootstrap **305** listens for a general protocol message (indicated by arrow **313**) from the other protocol apparatus. In a preferred embodiment, the general protocol message uses the same path between the protocol apparatuses as does protocol data **111**. In other embodiments, there may be a special path for the general protocol message. The general protocol message further contains at least the first part of protocol description **203** for the specific protocol to be executed. When bootstrap **305** receives the general protocol message, it loads the message into a buffer in protocol instruction interpreter data **311** and performs checks as described below. If the message passes the checks, bootstrap **305** loads the general protocol message into the portion of memory **309** reserved for protocol description **203**. Thereupon, interpreter **209** begins executing the protocol instructions **205** in the message, beginning with the initial instruction. If protocol description **203** is longer than the maximum size of an general protocol message, then the first part of protocol description **203** contains protocol instructions which, when executed, cause the rest of protocol description **203** to be loaded.

In a preferred embodiment, the general protocol requires that the general protocol message contain checking information which permits error checking and protocol data information which indicates how protocol instruction interpreter **209** is to interpret protocol data **111** and that the receiving general protocol apparatus **301** use the checking information and the protocol data information. In the preferred embodiment, there are two items of checking information: a checksum for the general protocol message and a required first instruction. On receiving the general protocol message, bootstrap **305** computes the general protocol message's checksum and compares it with the checksum in the message; if they are different, there has been a transmission error and bootstrap **305** waits for another general protocol message. If bootstrap **305**'s check of the required first instruction in the general protocol message indicates that the general protocol message is not a protocol description **203**, the error component **307** of protocol instruction interpreter **209** returns an error message (indicated by arrow **315**) to the protocol apparatus **101** which provided the general protocol message. Thereupon, error **307** waits for a valid general protocol message. Once the general protocol message has been successfully received, it is executed by protocol instruction interpreter **209**, and as part of the execution, the protocol data information in the general protocol message is used to set parameter values in protocol instruction interpreter data **309**.

If both protocol apparatuses **107** involved in a communication are protocol apparatuses **301**, an enormous amount of flexibility is possible. For example, if an entity **109** which includes a protocol apparatus **301** requires that information **105** sent to it be sent according to a given protocol, the

apparatus **301** can respond to a general protocol which specifies another protocol by returning an error message which indicates that it only responds to a given specific protocol and then sending protocol description **203** for the given specific protocol to the entity from which it received the general protocol message. Such a tactic might be used by an entity **109** which requires that all data which it receives be encrypted according to a particular scheme.

Similarly, if a communication between two entities **109** involves different types of data and different protocols are better for transferring data belonging to the different types, then two protocol apparatuses **301** could carry out the communication by dynamically changing the protocols as required by the type of data currently being communicated. An example here might be a data transfer which involved both digital data representing analog signals and digital data representing numbers or characters. The two types of data have different degrees of tolerance for transmission errors, and the protocol used for each type of data might therefore employ different error checking and correction techniques.

Adaptive General Protocol Apparatus: FIG. **13**

The flexibility of general protocol apparatus **301** comes at a cost: each communication using a specific protocol includes the overhead of sending protocol description **203** for the specific protocol to general protocol apparatus **301**, checking the general protocol message, and loading protocol description **203** into protocol instruction interpreter memory **309**. This overhead can be avoided by equipping general protocol apparatus **301** with a protocol instruction interpretation memory **1301** (FIG. **13**) which is large enough to hold a number of protocol descriptions **203** and modifying the general protocol to permit use of a protocol description identifier specifying one of the protocol descriptions in place of a protocol description **203**. For such an adaptive general protocol apparatus, the general protocol would be as follows:

In a sending protocol apparatus, sending a first message which includes a protocol description identifier for a protocol description **203**;

In a receiving general protocol apparatus, responding to the first message by:

a. determining whether the receiving general protocol apparatus has a copy of protocol description **203** specified by the identifier;

b. if it does, executing the protocol description **203** specified by the identifier;

c. if it does not, returning an error message indicating that it does not have a copy of the specified protocol description **203**;

in the sending protocol apparatus, responding to the error message by sending a second message which includes protocol description **203**; and

in the receiving protocol apparatus, responding to the second message by:

a. storing protocol description **203** in the receiving protocol apparatus; and

b. executing the protocol description.

As may be seen from the foregoing description of the general protocol for the adaptive general protocol apparatus, such a general protocol apparatus would quickly adapt itself to the environment in which it was employed. It would in short order contain copies of the protocol descriptions **203** for all of the protocols which were frequently employed by the entities **109** which used the adaptive general protocol apparatus, and would consequently only very rarely need to request a copy of the protocol description **203** for a protocol from the sender. Put another way, an adaptive general protocol apparatus will encache protocol descriptions **203** for frequently-used protocols in the same way that a memory system encaches frequently-used memory blocks.

An adaptive general protocol apparatus may be implemented by modifying bootstrap **303** and the contents of protocol instruction interpreter memory **301**. The modifications to the contents of protocol instruction interpreter memory for an adaptive general protocol apparatus **1323** are shown in FIG. **13**. As before, protocol instruction interpreter memory **1301** is divided into two parts, one containing data **311** used during execution of a protocol, and one for protocol descriptions. Here, protocol description memory (PDM) **1302** contains a protocol description table **1303** and one or more protocol descriptions **1311**. Protocol description table **1303** contains a protocol description table entry **1305** for each protocol description **1311** in memory **1309**. Each entry **1305** contains at least two pieces of information: an identifier **1307** for a protocol description **1311** and a pointer **1309** to the location in memory **1302** of protocol description **1311** specified by the identifier. There are many possible sources for the identifiers; for example, the identifier for a given protocol description **1311** may be the description **1311**'s checksum. In another embodiment, the source of the original protocol descriptions from which the protocol descriptions **1311** are copied may assign a unique identifier to each original protocol description.

As will be explained in more detail below, the protocol descriptions **203** employed in a preferred embodiment define a finite state machine. Consequently, a given protocol description **203** is divided into a set of numbered states (S) **1321**. To permit location of the states, protocol description **1311** is divided into two parts: protocol description body (PDB) **1314**, which contains the instructions for the states, and state table **1313**, which relates state numbers to the locations of the corresponding states **1321**. There is an entry **1315** in state table **1313** for each state **1321** in the protocol description body, and each entry contains the state number (SN) **1317** and the offset (OFF) **1319** of that state from the beginning of protocol description **1311**.

The modifications required in bootstrap **305** will be immediately apparent from FIG. **13** and the description of the general protocol for general protocol apparatus **1323**. When a general protocol message is received which contains a protocol description identifier for which the protocol description **1311** is in memory **1302**, bootstrap **305** simply causes interpreter **209** to begin executing the specified protocol description; otherwise, bootstrap **305** retains the identifier from the general protocol message and causes error **307** to return an error message and wait for a message which contains the protocol description **1311**. When the message arrives, error **307** causes bootstrap **305** to compare the retained identifier with the identifer in the general protocol message containing the protocol description **1311**, and if they agree, bootstrap **305** places the protocol description **1311** in memory **1302** and makes an entry **1305** for the new protocol description **1311** in protocol description table **1303**.

Of course, many variations on the above arrangements are possible. For example, memory **1302** is necessarily finite; consequently, bootstrap **305** may have to remove one protocol description **1311** to make room for another. One way of doing this would be to include size and most recent use information in protocol description table **1303**, and bootstrap **305** could use that information to determine which protocol descriptions **1311** should be removed. Further, the general protocol for general protocol apparatus **1323** might include a checksum in the general protocol message for the

protocol description **1311** identified by the identifier. Bootstrap **305** could use the checksum to make sure that the copy of the protocol description **1311** in memory **1302** was the same as the copy held by the sender. If it was not, bootstrap **305** could send an error message requesting the protocol description and then proceed as previously described for protocol descriptions for which there was no copy in memory **1302**.

Implementation of Protocol Apparatus **301**

A prototype implementation of protocol apparatus **301** has been constructed in which protocol execution device **207** is a computer capable of executing programs written in the well-known "C" programming language. In the prototype implementation, protocol instructions **205** belonging to a protocol description language are interpreted by a protocol instruction interpreter which is written in C and is executed by a process running on the computer. General protocol apparatus **301** has been tested by writing a protocol description **203** for the alternating bit protocol in the protocol description language and executing the protocol by executing the protocol description **203**. The following discussion will first disclose the protocol description language, then protocol interpreter **209**, bootstrap **305**, and error component **307**, and finally protocol description **203** for the alternating bit protocol.

The Protocol Description Language: FIG. **4**

FIG. **4** shows the instructions in a preferred embodiment of protocol description language **401**. The instructions fall into two classes: those which perform general stack management and expression evaluation, shown in table **403**, and those which perform operations which are particularly related to the execution of protocols, shown in table **405**.

As is apparent from table **403**, protocol instruction interpreter **209** is a stack machine. The stack, maintained in protocol instruction interpretation data **311**, is a standard integer size push-down stack. The PUSH_BYTE and PUSH_WORD instructions permit data to be pushed onto the push-down stack. The other instructions take their operands and parameters from the top of the stack and push their results back onto the top of the stack. When a stack overflow or underflow occurs, interpreter **209** ceases executing the protocol, error component **307** sends an error message to the other protocol apparatus **107**, and error component **307** then waits for an error handling message from the other protocol apparatus **107**. Of course, how the other protocol apparatus **107** responds to the error message is part of the protocol described in protocol description **203**. The same thing happens if an arithmetic error such as a zero divide or an integer overflow occurs.

The functions of the instructions in table **405** are generally clear from the table; however, certain instructions require a more detailed explanation. Beginning with the instructions in finite state machine control **407**, instructions **421** and **423** permit protocol detestation language **401** to describe a protocol as a finite state machine, that is, a finite set of states with definitions of transitions between the states. Thus, LOAD instruction **421** takes two parameters from the top of the stack, one specifying a buffer which contains a sequence of instructions of protocol description language **401**, and one specifying a state number. The LOAD instruction loads the contents of the buffer into a location in protocol description **203** and associates the state number with that location. NXT instruction **423** pops a state value from the top of the stack and begins execution of the sequence of instructions at the location in protocol description **203** associated with the state value. IF instruction **425** is a conditional branch instruction: the IF instruction pops the value at the top of the stack, and if the value at the top of the stack is equal to 0, the IF instruction branches to the instruction specified in the IF instruction's parameter; otherwise, it executes the instruction following the IF instruction.

Upper interface instructions **409** pass information **105** to data source/destination **103** and receive information **105** from data source/destination **103**. The information is passed from and received into buffers in protocol instruction interpretation data **311**. Lower interface instructions **41 1** deal with PDATA **111** sent and received between protocol apparatuses **107**. Three of these instructions are used in bootstrapping. CKSUM **413** computes the checksum of a buffer's contents and places the result in the top of the stack, where it can be used to determine whether a branch should be made to error component **307**. BYTEORDER defines the order of the bytes in the words of PDATA **111** which are sent according to the protocol. WORD_SZ defines the number of bytes in the words of PDATA **111** which are sent according to the protocol. Both instructions are used in the general protocol message to override default byte orders and word sizes, and they may be also used to change these aspects of PDATA **111** during transmission of a protocol. Buffer management instructions **419** allocate and size buffers in PII-DATA **311** and permit values from the top of the stack to be written to positions in the buffers. Most of the instructions also have a slightly faster variant (indicated by the prefix I_) which use constant operands specified in the instruction and therefore do not have to pop an operand from the stack.

The following is a short example program written in protocol description language **401**. The program first defines the byte order and word size for the protocol, loads the contents of a buffer RBUF into protocol description **203** and associates a state number represented as S with the location of the loaded contents, and then begins executing the contents as state S:

```
BYTEORDER,  1,          /* Most Significant Byte transmitted first */
WORD_SZ,    3,          /* 3 bytes per word */
I_LOAD,     S, RBUF,    /* assign RBUF to S */
I_NXT,      S,          /* goto state S*/
```

S is a constant value, as is RBUF, so I_LOAD and I_NXT are used in the program instead of LOAD and NXT.

While protocol description language **401** effectively describes protocols, a person implementing any substantial protocol would not want to write the protocol description directly in language **401**. To avoid this problem, the person implementing the protocol can describe the protocol in any formal protocol specification language that can be translated into language **401** and then translate the description in the formal specification language into language **401**. Even a regular programming language would do to describe the protocol. If the protocol is specified in a formal protocol specification language which permits validation of the protocol (see for example the PROMELA specification language, in Holzmann, supra, p. 91ff.), there is an added advantage that the complete protocol can be validated exhaustively before it is converted into protocol description language **401**. In this case, it will be certain that both sides of the protocol are implemented in precise accordance with the validated model.

Protocol Instruction Interpreter **209**

In a preferred embodiment, protocol instruction interpreter **209** is implemented by means of a run procedure which will be explained in more detail later. At the heart of that procedure is the procedure transition(n). A fragment of transistion(n) is shown in FIG. **10**. transition (n) **1001**

executes the protocol instructions **205** in one state until a NXT or I_NXT instruction transfers control to another state. The procedure is invoked with a single argument: the number of the state to which transition is to be made; when the procedure returns, it returns the number of the next state to which a transition is to be made. The variable cur_state is set on entry into the procedure to point to the beginning of the state specified by the argument. The register variable prot contains the current byte position in the state being executed. At **1006**, prot is set to the value of cur_state, so that execution starts at the first byte of the state. The while loop indicated at **1007** continues to execute as long as prot has a non-0 value, i.e., essentially until a return statement transfers control out of transition.

The body of the while loop is a switch statement which contains a case for each of the instructions in protocol description language **401**. On each iteration of the loop, the variable prot is incremented by 1, so that it points to the next byte in the state. The value of that byte determines which case is executed. If there is no case corresponding to that value, default case **1015** is executed, which puts interpreter **209** into the error state and thereby transfers control to error **307**. Where required, a case further contains debugging code and assertions to check whether requirements for the execution of the instruction are fulfilled. If interpreter **209** is only used with fully tested and verified protocol descriptions **203**, the assertions and debugging code may be disabled.

Fragment **1001** shows two cases in addition to default case **1015**: those for NXT and I_NXT. With NXT **1011**, the case simply pops the value at the top of the stack (i.e., the next state), checks whether the value is in the range of values for states, and returns the value. With I_NXT, the value of the next state is in the code, and not on the stack, so the case increments prot by one, checks whether the value at that point in the code is within the range, and returns the value.

Implementation of Bootstrap **305**: FIG. **5**

In a preferred embodiment, Bootstrap **305** is implemented as a state of interpreter **209**. Unlike the other states, which are defined by the protocol description loaded in by bootstrap **305**, bootstrap **305** and error **307** are required for the execution of the general protocol and therefore must be built into a protocol apparatus **301** before a protocol description **203** can be loaded.

Since these two states are required for the general protocol, they are the only ones that enforce a predefined format on incoming messages, and that must require, the presence of certain kinds of data to permit checking of the general protocol message. As soon as these two states have successfully received a general protocol message with protocol description **203**, they hand off control of the general protocol apparatus to the protocol description **203**.

In a preferred embodiment, bootstrap **305** is implemented with a single call run (BOOTSTRAP). Procedure run ( ) is the implementation of interpreter **209** in a preferred embodiment. The procedure is reproduced completely below

```
run (s)
{       int n = s;
        while (n >= 0 && n <= SMAX && State[n].cont)
            n = transition(n);
        return n;
}
```

run is a loop which invokes the procedure transition with a state number transition then puts interpreter **209** into the proper state of protocol description **203** or the states which implement bootstrap **305** or error **307**. The loop ceases execution when a value which is out of range of the legal state numbers is received. Thus, when invoked with BOOTSTRAP, a constant indicating the bootstrap state, the run simply puts interpreter **209** into the bootstrap state.

Most of the concepts involved in implementing an embodiment of protocol apparatus **301** can be illustrated using an implementation of bootstrap **305**. In a protocol apparatus **301** employing such an implementation, the code for bootstrap **305** would always be present in protocol instruction interpreter memory **309**.

For a general understanding of bootstrap **305**, the flow chart of FIG. **5** suffices. As shown in oval **503**, bootstrap implementation **501** waits for the general protocol message from the remote protocol apparatus **107**. When the message comes, it is loaded into a buffer in protocol instruction interpreter data **311**. Next, the message is checked. First, a checksum is performed, to make sure the message is uncorrupted. If the checksum is non-zero, a transmission error has occurred, and the machine returns to the start of the bootstrap state (diamond **505, 507**). If the checksum is zero, a check is made if the message has the correct type (diamond **509**). In a preferred embodiment, the first instruction is required to be the definition of the BYTEORDER for the lower interface. This byte-order definition specifies the order in which the bytes in a word sent according to the protocol are transmitted across the lower level interface: most or least significant byte first. It need not match the byte-order used in either the receiving or the sending entity. If the message is not a valid protocol description **203**, interpreter **209** enters error **307** (**511**).

If the message is a valid protocol description **203**, the contents of receive buffer is assigned to the initial state of the newly loaded protocol, and control is transferred to that state (box **515**). A full implementation **1101** of bootstrap **305** in protocol description language **401** is shown in FIG. **11**.

Implementation of Error **307**: FIG. **6**

In a preferred embodiment, error component **307** is also implemented as a state of interpreter **209**. Like the bootstrap state, this error state is part of the general protocol, not any specific protocol. It is only meant to provide a standard mechanism for responding to errors in the operation of the general protocol apparatus **301**, such as stack-overflow, memory allocation errors, arithmetic errors (e.g., divide by zero), etc. A flowchart for the error state is given in FIG. **6**.

As shown in FIG. **6**, error component implementation **601** first writes a predefined error message to TBUF (box **603**) and then notifies the remote protocol apparatus **107** of an error condition by sending the message in TBUF (oval **605**). It then awaits a response that it will receive into the default receive buffer RBUF (oval **607**). If the message was uncorrupted (diamond **609**) and was a protocol decription **203** (diamond **613**) control is transferred to the state that is specified in the message, using the NXT command (**617**). In all other cases (**611,615**), the error state is reentered from the top (**602**). A full implementation **1201** of error **307** in protocol description language **401** is shown in FIG. **12**.

An Implementation of the Alternating Bit Protocol: FIGS. **7–10**

FIG. **7** is a diagram of the finite state machines implemented by two protocol apparatuses **107** which are communicating by means of the alternating bit protocol. This protocol employs a single bit, which can have the value "1" or "0", as a message sequence number. When one of the apparatuses **107**, say the apparatus represented by finite state machine **703**, sends a message, it appends a "1" or an "0" bit as a sequence number to the message. The receiving finite state machine **705** sends an acknowledgment with the

sequence number which it received to the sending finite state machine **705**; if the acknowledgment's sequence number matches the sequence number of the sent message, the sending finite state machine can send another message with the other sequence number; if not, it repeats the last sent message.

In FIG. **7**, the circles specify states and the edges specify state transitions resulting from message exchanges. The edge labels specify the message exchanges. Each label consists of two characters: A indicates that the message comes from finite state machine **703**; B indicates that it comes from machine **705**. The second character specifies the sequence number, 1 or 0 as described above. When an edge label is underlined, it indicates that the message is being transmitted to the other finite state machine. The double headed arrows indicate states in which the receiver can accept a message from the sender or the sender can fetch a new message for output to the receiver.

In more detail, in state **707**, sender **703** receives a message A to be output to receiver **705**. It appends the "0" sequence number and outputs the message. In state **709**, it waits for the acknowledgment, indicated as the message B. If B has the sequence number "0", the next state is **711**, where a new message A is received for output and the "1" sequence number is appended. In state **713**, sender **703** waits for the acknowledgment, again indicated as B; if the message has the sequence number "1", the cycle begins again in state **707**. If the acknowledgment received in state **709** has the sequence number "1", the next state is **715**, which retransmits the message A transmitted in **707** with the "0" sequence number. If the right acknowledgment is received this time, the next state is **711**; otherwise, state **715** is reentered. Similarly, if an A message with a sequence number "1" receives an acknowledgment with a sequence number "0", sender **703** goes to state **717**, where it retransmits the A1 message.

FIG. **8** shows how the portion of the alternating bit protocol which responds to received messages is implemented using protocol description language **401**. The code **801** for the receive portion of the protocol has three parts: in part **803**, the buffers for both the transmit and receive portions are defined; for the receive portion, only three buffers are required: a transmit buffer TBUF, a receive buffer RBUF, and a variable VAR_E for the expected sequence number.

Portion **805** is state **0** of the portion of the finite state machine which receives messages. The protocol instructions **205** of state **0** are received in the general protocol message and are executed when bootstrap **305** transfers control to state **0**. As required in a preferred embodiment, the first instruction is a BYTEORDER instruction, which here specifies that the first byte of the words received according to the protocol is the most significant byte. The instructions at bytes **2** through **10** of the general protocol message allocate buffer space for TBUF and VAR_E. The instructions at bytes **11** through **17** specify that the receiver is to wait for the next message, which contains the protocol instructions **205** of state **1**, place that message in RBUF, load the contents of RBUF into the part of memory **309** reserved for protocol description **203**, and associate state **1** with the location of the loaded contents in memory **309**, and then execute state **1**. The part of portion **805** beginning at byte **18** is reserved for the checksum which bootstrap **305** uses to check for correctness of transmission.

Portion **807** is state **1** of the portion of the finite state machine which receives messages. In this state, the finite state machine waits for a message (byte **0**). When the message arrives, it is placed in RBUF. At bytes **2** through **12**, the finite state machine writes a value indicating an acknowledgment (in this case, the character 'A') into the transmittal buffer, obtains the sequence number in the last byte of RBUF, copies the sequence number into TBUF following the 'A', and sends the acknowledgment. At bytes **14** through **20**, the finite state machine compares the value of the sequence number retained in VAR_E with the sequence number in the last byte of RBUF. If they are the same, indicating that the received message had the right sequence number, bytes **23** through **33** are executed; otherwise, state **1** is again executed from the beginning, as shown at byte **34**. In bytes **23** through **31**, the sequence number saved in VAR_E is subtracted from 1 and the result saved in VAR_E again. Then, the message is sent to its destination and state **1** is again executed from the beginning.

FIG. **9** shows how the portion of the alternating bit protocol which sends messages is implemented in a preferred embodiment. In the preferred embodiment, protocol apparatus **107** in which send portion **901** is implemented is a protocol apparatus **201** or **301** which is sending to a protocol apparatus **301**. Consequently, the send portion is implemented in protocol description language **401** and includes instructions which download receive portion **801** to the receiving protocol apparatus **301**.

The buffers and variables used in portion **901** are defined in part **803** of FIG. **8**. In the prototype, buffers **0** and **1** are preset to each contain a message; as may be seen from part **809** of FIG. **8**, buffer **0** (named M0) is set to the ASCII character 'M' with the sequence number "0" appended to it, while buffer **1** is set to the ASCII character 'M' with the sequence number "1" appended to it. The buffer R_run contains the code of portion **807**, while the buffer R_ini contains the code of portion **805**. The buffer R_ack, finally, is used for the acknowledgement received from receiver **801**. There are two variables: VAR_S, which holds the sequence number which is to be attached to the message, and VAR_CNT, which is a count of the number of bytes sent by the sender.

Returning to FIG. **9**, the allocation and initialization of the sender buffers and variables defined in **803** takes place in section **903**, which is state **0**. In bytes **0–13**, VAR_S and VAR_CNT are both allocated and set to 0; in bytes **14** and **15**, receiver initialization code **805**, contained in the buffer R_ini, is sent to the receiver; in bytes **16** and **17**, the code **807** for state **1** of the receiver, contained in the buffer R_run, is sent to the receiver. These lines **905** thus perform the downloading of protocol description **203** to the receiving protocol apparatus **301**. At byte **18**, finally, the I_NXT instruction starts execution of state **1 907**.

At bytes **0–2** of state **1 907**, the current value of VAR_S is pushed onto the stack. At byte **3**, SEND takes its parameter from the top of the stack; thus, if VAR_S has the value 0, the message in buffer M0 is sent; if it has the value 1, the message in buffer M1 is sent. In an embodiment for use in an actual communications system, there would be code preceding the SEND instruction which employed the OBTAIN instruction to obtain a byte of data to be sent and place the data in buffer M0 or M1, depending on the value of VAR_S, and then employed CPY_BYTE to append "0" or "1" to the data, again depending on the value of VAR_S.

The code in bytes **4–8** receives the acknowledgment from the receiver and pushes it onto the stack. As pointed out in the discussion of the receiver, the acknowledgment has the form 'A'<sequence_number>. The top byte of the stack consequently should contain 'A' and the next byte should contain the sequence number. In bytes **9–11**, the top byte is

checked to see whether it contains 'A'. If it does, bytes **14–31** are executed; otherwise, execution continues at byte **32**. Bytes **14–31** check whether the acknowledgement has the right sequence number; if it does, they set VAR_S to the next sequence number. More specifically, bytes **14–20** check whether the sequence number in the acknowledgment is the same as the value of VAR_S. If it is not, execution continues at byte **32**; if it is, VAR_S is set to its new value by subtracting its current value from 1 (bytes **23–31**).

The code in bytes **32–54** updates VAR_CNT and terminates state **1** if the number of messages is greater than the constant NR_MSGS, defined in **803** to be 32765. In bytes **32–40**, **1** is added to the current value of VAR_CNT. In bytes **41–47**, VAR_CNT is pushed onto the stack, the most and least significant bytes of NR_MSGS is pushed onto the stack, and the two values are compared. If VAR_CNT>= NR_MSGS, bytes **50–52** put the value−1 on the stack. NXT returns that value to run, which thereupon terminates, as previously explained. Otherwise, byte **54** is executed, which causes state **1 907** to again begin execution.

Performance of Protocol Apparatus **201** or **301**

The performance of the implementation of the alternating-bit protocol just described was compared with the performance of an implementation in which the sender and receiver were simply coded in the "C" programming language. The extra overhead caused by the use of protocol description **203** and interpreter **209** instead of a "C" program ranged from 10–30%, depending on the length of the message being transmitted (longer messages have the lower overhead). In many applications, the extra overhead will be offset by the fact that the protocol apparatus of the invention can interpret any protocol for which there is a protocol description **203**. Further, there are many ways of reducing the overhead. Perhaps the most promising way is to implement interpreter **209** in hardware; such hardware would be capable of executing any protocol for which a protocol description **203** existed. Other optimizations include implementing interpreter **209** so that a minimum number of procedure calls are required, optimizing protocol descriptions **203** to avoid stack operations, and implementing interpreter **209** as an on-the-fly compiler, i.e., interpreter **209** operates by receiving protocol description **203** and compiling protocol description **203** into the machine instructions for the hardware which will actually implement the protocol. If the protocol apparatus is adaptive, the compilation would only have to be done before the first execution of the protocol description after it is loaded into the protocol apparatus.

Conclusion

The foregoing Detailed Description has disclosed to those of ordinary skill in the art how protocol apparatus may be constructed which is capable of executing any protocol for which there is a protocol description written in a given protocol language, how a sending protocol apparatus may provide a protocol description to a receiving protocol apparatus and thereby provide for execution of any protocol by the receiving protocol apparatus, and how a receiving protocol apparatus may be constructed which automatically adapts to the environment in which it is employed. Advantages of the techniques disclosed herein include more precise implementations of protocols, reduced implementation cost, and greatly increased flexibility.

While an example protocol description language and an example implementation of an interpreter for that protocol description language have been disclosed herein, it will be apparent to those of ordinary skill in the art that other protocol description languages and other implementations of the interpreter are possible. Moreover, other arrangements for downloading or otherwise specifying protocol descriptions may be used than those disclosed herein. That being the case, the Detailed Description is to be understood as being in all respects exemplary, but not restrictive, and the scope of the invention is to be determined not from the Detailed Description, but rather from the appended claims as interpreted in light of the Detailed Description and the doctrine of equivalents.

What is claimed is:

1. A method of communicating in a distributed system comprising the steps of:

in a first entity of the distributed system, receiving a first general protocol message which includes a protocol description which describes a specific protocol, the specific protocol described by the protocol description being initially unknown to the first entity, the protocol description being in a protocol description language which is independent of any particular hardware or software implementation of the first entity; and

responding to the first general protocol message by employing first protocol description interpretation means to execute the protocol description included in the first general message which enables the first entity to communicate with a second entity of the distributed system using the specific protocol.

2. Protocol apparatus for communicating in a distributed system, the apparatus comprising:

in a first entity of the distributed system,

means for storing a protocol description which describes a specific protocol, the protocol described by the protocol description being initially unknown to the first entity and in a protocol description language which is independent of any particular hardware or software implementation of the protocol apparatus; and

means for executing the protocol description to implement the specific protocol and enabling the first entity to communicate with a second entity of the distributed system using the specific protocol.

3. A method for communicating data between elements of a distributed system, comprising:

receiving in a first element of the distributed system a first data message defining an arbitrary data communication protocol which is initially unknown to the first element and is independent from any particular hardware or software implementation of the first element;

configuring the first element to receive data formatted in the arbitrary data communication protocol defined in the first data message; and

receiving, in the first element, at least one additional data message, the at least one additional data message transmitted using the arbitrary data communication protocol.

4. The method of communicating set forth in claim 3, wherein the step of configuring the first element includes:

determining whether the first data message has been correctly transmitted; and

if the first data message was not correctly transmitted, returning to the step of receiving the first data message.

5. The method of communicating set forth in claim 3, wherein the step of configuring the first element includes:

determining whether the first data message is a valid first data message; and

if the first data message is not valid, sending an error message.

6. The method of communicating set forth in claim 3, wherein the step of configuring the first element includes interpreting the arbitrary data communication protocol according to the at least one parameter contained in the first data message.

7. The method of communicating set forth in claim 6, wherein:

the at least one parameter specifies a byte order employed in the arbitrary data communication protocol; and

the step of interpreting the arbitrary data communication protocol includes interpreting the arbitrary data communication protocol according to the specified byte order.

8. The method of communicating set forth in claim 6, wherein:

the at least one parameter specifies a word size employed in the arbitrary data communication protocol; and

the step of interpreting the arbitrary data communication protocol includes interpreting the arbitrary data communication protocol according to the specified word size.

9. The method of communicating set forth in claim 3, further comprising:

sending the first data message from a second element to the first element; and

employing the arbitrary data communication protocol in communications between the first and second elements.

10. The method of communicating set forth in claim 9, further comprising responding to an error message which the first element provides to the second element in response to an error in the first data message.

11. The method of communicating set forth in claim 9, wherein the step of employing the arbitrary data communication protocol includes configuring the second element to receive data formatted in the arbitrary data communication protocol.

12. The method of communicating set forth in claim 3, wherein the step of configuring the first element includes directly executing instructions contained in the first data message and formatted in the arbitrary data communication protocol.

13. The method of communicating set forth in claim 3, wherein the step of configuring the first element includes compiling the arbitrary data communication protocol to produce instructions directly executable by means accessible to the first element.

14. The method set forth in claim 3, wherein the step of configuring the first element to receive data formatted in the arbitrary data communication protocol includes making the arbitrary data communication protocol included in the first data message accessible to the first element.

15. The method set forth in claim 3, further comprising:

receiving in the first element a second data message which defines a second arbitrary data communication protocol;

responding to the second data message by determining whether the second arbitrary data communication protocol is accessible to the first element;

if the second arbitrary data communication protocol is accessible, configuring the first element to interpret the second arbitrary data communication protocol; and

if the second arbitrary data communication protocol is not accessible, sending a first error message, and performing the step of receiving a first data message.

16. The method set forth in claim 15, wherein the step of receiving the first data message includes determining whether the arbitrary data communication protocol included in the first data message is equivalent to the second arbitrary data communication protocol.

17. The method set forth in claim 16, wherein the step of determining whether the arbitrary data communication protocol included in the first data message is equivalent to the second arbitrary data communication protocol includes sending a second error message if the arbitrary data communication protocol is not equivalent to the second arbitrary data communication protocol.

18. Protocol apparatus within a first element of a distributed system for communicating with other elements of the distributed system, the protocol apparatus comprising:

means for receiving a first data message which defines an arbitrary data communication protocol which is initially unknown to the first element and is independent from any particular hardware or software implementation of the first element; and

means for configuring the first element to receive data formatted in the arbitrary data communication protocol.

19. The protocol apparatus as set forth in claim 18, wherein the means for receiving a first data message checks the first data message to determine whether the first data message has been correctly transmitted; and

if the first data message has not been correctly transmitted, the means for configuring the first element does not configure the first element to receive data formatted in the arbitrary data communication protocol.

20. The protocol apparatus set forth in claim 18, wherein the means for receiving a first data message includes a checking and indicating means for checking the first data message and indicating whether the first data message is a valid first data message the protocol apparatus further comprising error handling means for sending an error message when the checking and indicating means indicates that the first data message is not valid.

21. The protocol apparatus set forth in claim 18, further comprising:

error handling means for sending an error message when an error occurs during operation of the means for configuring the first element to receive data formatted in the arbitrary data communication protocol.

22. The protocol apparatus set forth in claim 18, wherein:

the first data message includes at least one parameter for interpreting data transferred according to the arbitrary data communication protocol; and

the means for configuring the first element interprets the transmitted data according to the at least one parameter.

23. The protocol apparatus set forth in claim 22, wherein:

the at least one parameter specifies a byte order of the transmitted data; and

the means for configuring the first element interprets the transmitted data based on the specified byte order.

24. The protocol apparatus set forth in claim 22, wherein:

the at least one parameter specifies a word size of the transmitted data; and

the means for configuring the first element interprets the transmitted data based on the specified word size.

25. The protocol apparatus as set forth in claim 18, wherein the means for configuring the first element interprets the first data message by directly executing instructions contained in the first data message.

26. The protocol apparatus set forth in claim 18, wherein the means for configuring the first element interprets the first data message by compiling the first data message to produce instructions.

27. The protocol apparatus set forth in claim 18, wherein:
the protocol apparatus further includes error handling means;
the means for receiving the first data message further receives a second data message; and
the means for configuring the first element further includes:
means for receiving data formatted in a second arbitrary data communication protocol, and
means for determining whether the second arbitrary data communication protocol is accessible;
wherein,
if the second arbitrary data communication protocol is accessible, the configuring means interprets the second arbitrary data communication protocol, and
if the second arbitrary data communication protocol is not accessible, the configuring means sends a first error message.

28. The protocol apparatus set forth in claim 27, wherein the means for configuring the first element further makes accessible the arbitrary data communication protocol included in the first data message.

29. The protocol apparatus set forth in claim 27, wherein the means for configuring the first element further determines whether the arbitrary data communication protocol included in the first data message is the same as the second arbitrary data communication protocol.

30. The protocol apparatus set forth in claim 18, wherein the first data message is in a predetermined base format.

31. The protocol apparatus set forth in claim 30, wherein the predetermined base format includes a field specifying a word size.

32. The protocol apparatus set forth in claim 30, wherein the predetermined base format includes a field specifying a byte order.

33. An apparatus for communicating between computers in a distributed system, comprising:
a first data processing apparatus of the distributed system configured in a data communication protocol;
a second data processing apparatus of the distributed system configured in an arbitrary data communication protocol;
wherein the first data processing apparatus includes:
receiving means for receiving from the second data processing apparatus a general message that defines the arbitrary data communication protocol which is initially unknown to the first data processing apparatus and is independent from any particular hardware or software implementation of the first data processing apparatus, and
configuring means for placing the first data processing apparatus into the arbitrary data communication protocol defined by the general message, wherein the receiving means is able to receive an additional message in the arbitrary data communication protocol from the second data processing apparatus.

34. The apparatus for communicating set forth in claim 33, wherein the configuring means further makes the arbitrary data communication protocol accessible to the first data processing apparatus.

35. The apparatus for communicating set forth in claim 33, wherein:
the second data processing apparatus includes error handling means;
the receiving means further receives a second general message including a second arbitrary data communication protocol; and
the configuring means determines whether the second arbitrary data communication protocol is accessible, wherein, if the configuring means determines the second arbitrary data communication protocol is accessible, the configuring means interprets the second arbitrary data communication protocol, and if the configuring means determines the second arbitrary data communication protocol is not accessible, the receiving means sends a first error message to the second data processing apparatus.

36. The apparatus for communicating set forth in claim 35, wherein the configuring means further determines whether the arbitrary data communication protocol is the same as the second arbitrary data communication protocol.

37. A method for communicating data between a first data processing apparatus and a second data processing apparatus, the method comprising:
transmitting a first data message defining an arbitrary data communication protocol, which is initially unknown to the first data processing apparatus and is independent from any particular hardware or software implementation of the first data processing apparatus, to the first data processing apparatus from the second data processing apparatus, at least a first portion of the first data message being in the predetermined data communication protocol;
placing the first data processing apparatus into the arbitrary data communication protocol defined in the first data message; and
transmitting at least one additional data message between the first data processing apparatus and second data processing apparatus, the at least one additional message in the arbitrary data communication protocol defined in the first data message.

38. The method for communicating data set forth in claim 37, wherein the first data processing apparatus and the second data processing apparatus are computers.

39. The method for communicating data set forth in claim 37, wherein one of the first data processing apparatus and the second data processing apparatus is a peripheral apparatus.

40. A method for transmitting data to a first element of a distributed system, the method comprising:
receiving, in the first element, a first data message defining an arbitrary data communication protocol which is initially unknown to the first element and is independent from any particular hardware or software implementation of the first element, at least a first portion of the first data message being in the predetermined data communication protocol;
placing the first element into the arbitrary data communication protocol defined by the first data message; and
receiving, in the first element, at least one additional data message, the at least one additional data message in the arbitrary data communication protocol defined in the first data message.

41. A method for communicating between communications devices, comprising:
receiving, using a general protocol, a first message at a first communications device containing a protocol definition defining a specific protocol to be used for subsequent messages;
executing the received protocol definition to implement the specific protocol at the first communications device; and
receiving, at the first communications device, the subsequent messages using the specific protocol.

42. The method for communicating between communications devices as set forth in claim 41, wherein the first message comprises a set of messages.

43. A method for communicating between communication devices, comprising:

receiving, using a general protocol, a first message at a first communications device identifying a specific protocol to be used for subsequent messages;

determining if a protocol description corresponding to the specific protocol is available to the first communications device and if the protocol description corresponding to the specific protocol is stored in the memory, transmitting an acknowledge message, otherwise transmitting an error message;

in response to the error message, receiving a second message using the general protocol containing the protocol description for the specific protocol;

executing the received protocol definition to implement the specific protocol at the first communications device; and

receiving, at the first communications device, the subsequent messages using the specific protocol.

44. A communications device capable of communicating with other communications devices, comprising:

a communications circuit for routing communications messages to and from the other communications devices and transferring data to and from the communications device; and

a protocol apparatus that implements protocols used to communicate with the other communications devices, the protocol apparatus comprising:

a memory for storing at least one protocol definition, each protocol definition defining a corresponding protocol and being in a communications device-independent protocol description language,

a protocol interpreter that executes the stored protocol definition to implement the corresponding protocol, and

a bootstrap interpreter that inputs a message received from one of the other communications devices, the message being in a general protocol and containing a protocol definition defining a received protocol, stores the received protocol definition in the memory, and causes the protocol interpreter to execute the received protocol definition to implement the received protocol, and enables the communications device to receive subsequent messages from the one other communications device using the received protocol.

45. A communications device capable of communicating with other communications devices, comprising:

a communications circuit that processes communication messages from the other communications devices, at least one of the communication messages specifying a protocol definition;

a protocol apparatus that implements a protocol used to control communication sessions with the other communications devices, comprising:

a memory for storing at least one protocol definition, each protocol definition defining a corresponding protocol and being in an independent protocol description language;

a protocol instruction interpreter that executes a protocol definition stored in the memory; and

a bootstrap interpreter that inputs a message received from one of the other communications devices, the message containing a protocol definition, stores the received protocol definition in the memory, and causes the protocol instruction interpreter to execute the received protocol definition.

46. A protocol for communicating between elements, comprising:

a resident portion, the resident portion present in each element; and

a non-resident portion that is transmittable from a first one of the elements to the second one of the elements, subsequent communications between the first and second elements performed based on the non-resident portion of the protocol.

* * * * *

Our Docket/Ref. No.: AE. 01-2 Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz et al. Serial No.: 09/609179 Filed: June 30, 2000 Title: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE	Group Art Unit: 2756 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

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

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

Our Docket/Ref. No.: APT 001-2

Patent GP 2757 2157

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz et al. Serial No.: 09/609179 Filed: June 30, 2000 Title: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE	Group Art Unit: 2756 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

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

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

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address COMMISSIONER OF PATENTS AND TRADEMARKS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/609,179	06/30/2000	Russell S. Dietz	APPT-001-2	2668

7590 06/04/2003

Dov Rosenfeld
5507 College Avenue
Suite 2
Oakland, CA 94618

EXAMINER	
DINH, KHANH Q	

ART UNIT	PAPER NUMBER
2155	

DATE MAILED: 06/04/2003

6

Please find below and/or attached an Office communication concerning this application or proceeding.

PTO-90C (Rev. 07-01)

D

Office Action Summary	Application No.	Applicant(s)
	09/609,179	DIETZ ET AL.
	Examiner	Art Unit
	Khanh Dinh	2155

-- 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 *11 April 2002* .

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) *1-18* 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) *1-3,13,14,17 and 18* is/are rejected.

7)☒ Claim(s) *4-11,15 and 16* 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).

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, 5* .
4) ☐ Interview Summary (PTO-413) Paper No(s). _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: .

DETAILED ACTION

1. Claims 1-18 are presented for examination.

Claim Rejections - 35 USC § 112

2. The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

3. Claims 1 and 16 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.

In claim 1 (page 124, line 7 and line 13, word 2) and claim 16 (page 127, line 9 word 7 and line 16 word 12):

The term "**none or more**" should be changed to "*one or more*". The Examiner assumed the term *"one or more"* in this Office Action.

Correction is required.

Claim Rejections - 35 USC § 102

4. 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 –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

5. Claims 1-3, 13, 14, 17 and 18 are rejected under 35 U.S.C. 102(b) as being anticipated by Bruell US pat. No.5,680,585.

As to claim 1, Bruell discloses a method of performing protocol specific operations on a packet passing through a connection point on a computer network, the method comprising:

(a) receiving the packet (see fig.2a, abstract, col.4 lines 33-48).

(b) receiving a set of protocol descriptions for a plurality of protocols (i.e., using a wide range of data type protocols, see col.5 lines 10-22) that conform to a layered model, a protocol description for a particular protocol at a particular layer level including:

(i) the one or more child protocols of the particular protocol, the to packet including for any particular child protocol of the particular protocol information at on or more locations in the packet related to the particular child protocol (child fields, see col.4 line 49 to col.6 line 49).

(ii) the one or more locations in the packet where information is stored related to any child protocol of the particular protocol (see col.9 line 8 to col.10 line 63).

(iii) the one more protocol specific operations (i.e., tests using packets description files) to be performed on the packet for the particular protocol at the particular layer level (see col.9 line 8 to col.10 line 63 and col.14 line 37 to col.15 line 10).

(c) performing the protocol specific operations on the packet specified by the set of protocol descriptions based on the base protocol of the packet and the children of the protocols used in the packet (see fig.4, col.15 line 11 to col.16 line 42).

As to claim 2, Bruell discloses performing protocol specific operations is performed recursively for any children of the children (see col.9 line 8 to col.10 line 63 and col.14 line 37 to col.15 line 10).

As to claim 3, Bruell discloses which protocol specific operations (test platforms) are performed is step (c) depending on the contents of the packet such that the method adapts to different protocols according to the contents of the packet (see col.9 line 8 to col.10 line 63 and col.14 line 37 to col.15 line 10).

As to claim 13, Burell discloses the protocol specific operations including one or more parsing and extraction operations on the packet to extract selected portions of the packet to form a function of the selected portions for identifying the packet as belonging to a conversational flow (decoding packets received in accordance with a defined packet format, see col.4 line 49 to col.6 line 30 and col.14 line 37 to col.15 line 10).

As to claim 14, Bruell discloses the protocol descriptions are provided in a protocol description language (using Packet Description Language, see col.5 line 24 to col.6 line 49).

As to claim 17, Bruell discloses the protocol specific operations further including one or more state processing operations that are a function of the state of the flow of the packet (see fig.1, col.3 line 20 to col.4 line 33 and col.14 line 38 to col.15 line 10).

As to claim 18, Bruell discloses the protocol specific operations including one or more state processing operations that are a function of the state of the flow of the packet (see fig.1, col.3 line 20 to col.4 line 33 and col.14 line 38 to col.15 line 10).

Allowable Subject Matter

6. Claims 4-11 and 15 are objected to as being dependent upon a rejected base claim, but would be allowable if rewritten in independent form including all of the limitations of the base claim and any intervening claims.

7. Claim 16 would be allowable if rewritten to overcome the rejection(s) under 35 U.S.C. 112, second paragraph, set forth in this Office action and to include all of the limitations of the base claim and any intervening claims.

8. The following is a statement of reasons for the indication of allowable subject matter:

None of the cited prior art recites or discloses a network monitor to be analyze different packets or frame formats for performing specific operations comprising a combination of: storing a database in a memory, the database generated from the set of protocol descriptions and including a data structure containing information on the possible protocols and organized for locating the child protocol related information for any protocol, the data structure contents indexed by a set of one or more indices, the database entry indexed by a particular set of index values including an indication of validity, wherein the child protocol related information includes a child recognition pattern, wherein step (c) of performing the protocol specific operations includes, at any particular protocol layer level starting from the base level, searching the packet at to the particular protocol for the child field, the searching including indexing the data structure until a valid entry is found, and whereby the data structure is configured for rapid searches using the index set. The invention further includes the steps of: looking up a flow-entry database comprising one or more flow-entries, at least one flow-entry for each previously encountered conversational flow, the looking up using at least some of the selected packet portions

and determining if the packet matches an existing flow-entry; if the packet is of an existing flow, classifying the packet as belonging to the found existing flow; and if the packet is of a new flow, storing 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, wherein the parsing and extraction operations depend on the contents of one or more packet headers.

Other prior art cited

9. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

   a. Logan et al., US pat. No.5,721,827.

   b. Gupta et al., US pat. No.6,272,151.

   c. Rossmann, US pat. No.6,430,409.

   d. Trip et al., US pat. No.6,516,337.

   e. Harvey et al., US pat. No.6,519,568.

Conclusion

10. Claims 1-3, 13, 14, 17 and 18 are rejected.

11. Claims 4-11 and 15 are objected to as being dependent upon a rejected base claim, but would be allowable if rewritten in independent form including all of the limitations of the base claim and any intervening claims.

12. Claim 16 would be allowable if rewritten to overcome the rejection(s) under 35 U.S.C. 112, second paragraph, set forth in this Office action and to include all of the limitations of the base claim and any intervening claims.

13. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Khanh Dinh whose telephone number is (703) 308-8528. The examiner can normally be reached on Monday through Friday from 8:00 A.m. to 5:00 P.m.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Ayaz R. Sheikh, can be reached on (703) 305-9648. The fax phone numbers for this group are:

After Final: (703) 746-7238

Official: (703) 746-7239

Non-Official/ Draft: (703) 746-7240

A shortened statutory period for reply is set to expire THREE months from the mailing date of this communication. Failure to response within the period for response will cause the application to become abandoned (35 U.S. C. Sect. 133). Extensions of time may be obtained under the provisions of 37 CFR 1.136(A).

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the Group receptionist whose telephone number is (703) 305 -9600.

AYAZ SHEIKH
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100

Khanh Dinh
Art Unit 2155
Patent Examiner
5/29/2003

Notice of References Cited

09/609,179	Reexamination DIETZ ET AL.	
Examiner Khanh Dinh	Art Unit 2155	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-5,680,585	10-1997	Bruell, Gregory O.	703/26
	B	US-5,721,827	02-1998	Logan et al.	709/217
	C	US-6,272,151	08-2001	Gupta et al.	370/489
	D	US-6,430,409	08-2002	Rossmann, Alain	455/422.1
	E	US-6,516,337	02-2003	Tripp et al.	709/202
	F	US-6,519,568	02-2003	Harvey et al.	705/1
	G	US-			
	H	US-			
	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	
	V	
	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.

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.: 09/609179	***Ref./Docket No***: APPT-001-2
Applicant(s): Dietz, *et al.*	***Examiner***.: Dinh, Khanh Q.
Filing Date: June 30, 2000	***Art Unit***: 2155

FAX COVER PAGE

TO: Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

United States Patent and Trademark Office
(Examiner Dinh, Khanh Q., Art Unit 2155)

Fax No.: 703-746-7239

DATE: June 13, 2003

FROM: Dov Rosenfeld, Reg. No. 38687

RE: Response to Office Action

Number of pages including cover: *19*

OFFICIAL COMMUNICATION

PLEASE URGENTLY DELIVER A COPY OF THIS RESPONSE TO EXAMINER DINH, KHANH Q., ART UNIT 2155

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number 703-746-7239 addressed the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: 13 June 03 Signed: _____

Name: Dov Rosenfeld, Reg. No. 38687

TRANSMITTAL FORM

(to be used for all correspondence after initial filing)

Application Number	09/609179
Filing Date	30 Jun 2000
First Named Inventor	Dietz, Russell S.
Group Art Unit	2155
Examiner Name	Dinh, Khanh Q.
Attorney Docket Number	APPT-001-2

ENCLOSURES *(check all that apply)*

- ☐ Fee Transmittal Form
- ☐ Fee Attached
- ☒ 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
- ☐ 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
- ☐
- ☐

Remarks

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT/ CORRESPONDENCE ADDRESS

Firm *or* Individual name	Dov Rosenfeld, Reg. No. 38687
Signature	
Date	June 13, 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: June 13, 2003

Type or printed name	Dov Rosenfeld, Reg. No. 38687		
Signature		Date	June 13, 2003

6/13/03

Our Ref./Docket No: APPT-001-2 Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz, *et al.* Application No.: 09/609179 Filed: June 30, 2000 Title: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE	Group Art Unit: 2155 Examiner: Dinh, Khanh Q.

TRANSMITTAL: RESPONSE TO OFFICE ACTION

Mail Stop Non Fee Amendment
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:

____ formal drawings (with separate letter);

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.

__X__ No additional fee is required.

6-13-03

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number 703-746-7239 addressed the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: 13 June 03 Signed: _____

Name: Dov Rosenfeld, Reg. No. 38687

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.: 09/609179	***Ref./Docket No***: APPT-001-2
Applicant(s): Dietz, *et al.*	***Examiner***.: Dinh, Khanh Q.
Filing Date: June 30, 2000	***Art Unit***: 2155

FAX COVER PAGE

TO: Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

United States Patent and Trademark Office
(Examiner Dinh, Khanh Q., Art Unit 2155)

Fax No.: 703-746-7239

DATE: June 13, 2003

FROM: Dov Rosenfeld, Reg. No. 38687

RE: Response to Office Action

Number of pages including cover: 19

OFFICIAL COMMUNICATION

PLEASE URGENTLY DELIVER A COPY OF THIS RESPONSE TO EXAMINER DINH, KHANH Q., ART UNIT 2155

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number 703-746-7239 addressed the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: 13 June 03 Signed: _____

Name: Dov Rosenfeld, Reg. No. 38687

TRANSMITTAL FORM

(to be used for all correspondence after initial filing)

Application Number	09/609179
Filing Date	30 Jun 2000
First Named Inventor	Dietz, Russell S.
Group Art Unit	2155
Examiner Name	Dinh, Khanh Q.
Attorney Docket Number	APPT-001-2

ENCLOSURES ***(check all that apply)***

☐ Fee Transmittal Form	☐ Assignment Papers *(for an Application)*	☐ After Allowance Communication to Group
☐ Fee Attached	☐ Drawing(s)	☐ Appeal Communication to Board of Appeals and Interferences
☒ Amendment / Response	☐ Licensing-related Papers	☐ Appeal Communication to Group *(Appeal Notice, Brief, Reply Brief)*
☐ ☐ After Final	☐ Petition Routing Slip (PTO/SB/69) and Accompanying Petition	☐ Proprietary Information
☐ ☐ Affidavits/declaration(s)	☐ To Convert a Provisional Application	☐ Status Letter
☐ Extension of Time Request	☐ Power of Attorney, Revocation Change of Correspondence Address	☐ Additional Enclosure(s) *(please identify below):*
☐ Express Abandonment Request	☐ Terminal Disclaimer	☒ Return Postcard
☐ Information Disclosure Statement	☐ Small Entity Statement	☐
☐ Certified Copy of Priority Document(s)	☐ Request of Refund	☐
☐ Response to Missing Parts/ Incomplete Application	Remarks	
☐		
☐ ☐ Response to Missing Parts under 37 CFR 1.52 or 1.53		

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT/ CORRESPONDENCE ADDRESS

Firm *or* Individual name	Dov Rosenfeld, Reg. No. 38687
Signature	
Date	June 13, 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: June 13, 2003

Type or printed name	Dov Rosenfeld, Reg. No. 38687		
Signature		Date	June 13, 2003

Our Ref./Docket No: APPT-001-2 Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz, *et al.* Application No.: 09/609179 Filed: June 30, 2000 Title: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE	Group Art Unit: 2155 Examiner: Dinh, Khanh Q.

TRANSMITTAL: RESPONSE TO OFFICE ACTION

Mail Stop Non Fee Amendment
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:

____ formal drawings (with separate letter);

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.

__X__ No additional fee is required.

6-13-03

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number 703-746-7239 addressed the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: 13 June 03 Signed: _____

Name: Dov Rosenfeld, Reg. No. 38687

The fee has been calculated as shown below:

CLAIMS AS AMENDED						
	CLAIMS REMAINING AFTER AMENDMENT		HIGHEST NUMBER PREVIOUSLY PAID FOR	NO. OF EXTRA CLAIMS PRESENT	RATE	ADDITIONAL FEE
TOTAL CLAIMS	17	MINUS	20	0	$18	$ 0.00
INDEP. CLAIMS	3	MINUS	3	0	$84	$ 0.00
					TOTAL ADDITIONAL FEE DUE:	$ 0.00

__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 ($410)

_____ two months ($930) _____ four months ($1450)

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

_____ 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,

13 June 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

6-13-03

Our Ref./Docket No: APPT-001-2 Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz, *et al.* Application No.: 09/609179 Filed: June 30, 2000 Title: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE	Group Art Unit: 2155 Examiner: Dinh, Khanh Q.

TRANSMITTAL: RESPONSE TO OFFICE ACTION

Mail Stop Non Fee Amendment
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:

____ formal drawings (with separate letter);

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.

__X__ No additional fee is required.

6-13-03

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number 703-746-7239 addressed the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: 13 June 03 Signed: _____

Name: Dov Rosenfeld, Reg. No. 38687

The fee has been calculated as shown below:

CLAIMS AS AMENDED						
	CLAIMS REMAINING AFTER AMENDMENT		HIGHEST NUMBER PREVIOUSLY PAID FOR	NO. OF EXTRA CLAIMS PRESENT	RATE	ADDITIONAL FEE
TOTAL CLAIMS	17	MINUS	20	0	$18	$ 0.00
INDEP. CLAIMS	3	MINUS	3	0	$84	$ 0.00
					TOTAL ADDITIONAL FEE DUE:	$ 0.00

__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 ($410)

____ two months ($930) ____ four months ($1450)

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

____ 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,

13 June 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

6-13-03

#7/A
LOS
6-17-03
entered

Our Ref./Docket No: APPT-001-2 Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz, *et al.* Application No.: 09/609179 Filed: June 30, 2000 Title: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE	Group Art Unit: 2155 Examiner: Dinh, Khanh Q.

RESPONSE TO OFFICE ACTION UNDER 37 CFR 1.111

Mail Stop Non Fee Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

This is a response to the Office Action of June 4, 2003.

Official

6-13-03

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number 703-746-7239 addressed the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: 13 June 03 Signed: _____

Name: Dov Rosenfeld, Reg. No. 38687

INTRODUCTORY REMARKS:

In response to the Office Action of June 4, 2003, kindly amend this application as follows and kindly consider the following remarks.

6-13-03

AMENDMENT(S) TO THE CLAIMS:

The following listing of claims will replace all prior versions, and listings, of claims on the application. Claims being amended are set forth in a larger font than all other claims. All claims are set forth below with one of the following annotations.

- (Original): Claim filed with the application following the specification.
- (Currently amended): Claim being amended in the current amendment paper.
- (Previously amended): Claim not being currently amended, but which was amended in a previous amendment paper.
- (Cancelled): Claim cancelled or deleted from the application.
- (Withdrawn): Claim still in the application, but in a non-elected status.
- (Previously added): Claim added in an earlier amendment paper.
- (New): Claim being added in the current amendment paper.
- (Reinstated - formerly claim # _): Claim deleted in an earlier amendment paper, but re-presented with a new claim number in current amendment.
- (Previously reinstated): Claim deleted in an earlier amendment and reinstated in an earlier amendment paper.
- (Re-presented - formerly dependent claim # _): Dependent claim re-presented in independent form in current amendment paper.
- (Previously re-presented): Dependent claim re-presented in independent form in an earlier amendment, but not currently amended.

1. (Cancelled)

2. (Currently amended) A method according to claim ~~1~~ 10, wherein step (c) of performing protocol specific operations is performed recursively for any children of the children.

3. (Currently amended) A method according to claim ~~1~~ 10, wherein which protocol specific operations are performed is step (c) depends on the contents of the packet such that the method adapts to different protocols according to the contents of the packet.

4. (Currently amended) A method ~~according to claim 1, further comprising:~~ of performing protocol specific operations on a packet passing through a connection point on a computer network, the method comprising:

6-13-03

(a) receiving the packet;

(b) receiving a set of protocol descriptions for a plurality of protocols that conform to a layered model, a protocol description for a particular protocol at a particular layer level including:

(i) if there is at least one child protocol of the protocol at the particular layer level, the one or more child protocols of the particular protocol at the particular layer level, the packet including for any particular child protocol of the particular protocol at the particular layer level information at one or more locations in the packet related to the particular child protocol,

(ii) the one or more locations in the packet where information is stored related to any child protocol of the particular protocol, and

(iii) if there is at least one protocol specific operation to be performed on the packet for the particular protocol at the particular layer level, the one or more protocol specific operations to be performed on the packet for the particular protocol at the particular layer level; and

(c) performing the protocol specific operations on the packet specified by the set of protocol descriptions based on the base protocol of the packet and the children of the protocols used in the packet,

the method further comprising:

storing a database in a memory, the database generated from the set of protocol descriptions and including a data structure containing information on the possible protocols and organized for locating the child protocol related information for any protocol, the data structure contents indexed by a set of one or more indices, the database entry indexed by a particular set of index values including an indication of validity,

wherein the child protocol related information includes a child recognition pattern,

wherein step (c) of performing the protocol specific operations includes, at any particular protocol layer level starting from the base level, searching the packet at the particular protocol for the child field, the searching including indexing the data structure until a valid entry is found, and

whereby the data structure is configured for rapid searches using the index set.

2 5. (Original) A method according to claim 1 4, wherein the protocol descriptions are provided in a protocol description language, the method further comprising:

compiling the PDL descriptions to produce the database.

3 6. (Original) A method according to claim 1 4, wherein the data structure comprises a set of arrays, each array identified by a first index, at least one array for each protocol, each array further indexed by a second index being the location in the packet where the child protocol related information is stored, such that finding a valid entry in the data structure provides the location in the packet for finding the child recognition pattern for an identified protocol.

4 7. (Original) A method according to claim 3 6, wherein each array is further indexed by a third index being the size of the region in the packet where the child protocol related information is stored, such that finding a valid entry in the data structure provides the location and the size of the region in the packet for finding the child recognition pattern.

5 8. (Original) A method according to claim 4 7, wherein the data structure is compressed according to a compression scheme that takes advantage of the sparseness of valid entries in the data structure.

A1

6 9. (Original) A method according to claim 5 8, wherein the compression scheme combines two or more arrays that have no conflicting common entries.

7 10. (Original) A method according to claim 1 4, wherein the data structure includes a set of tables, each table identified by a first index, at least one table for each protocol, each table further indexed by a second index being the child recognition pattern, the data structure further including a table that for each protocol provides the location in the packet where the child protocol related information is stored, such that finding a valid entry in the data structure provides the location in the packet for finding the child recognition pattern for an identified protocol.

8 11. (Original) A method according to claim 7 10, wherein the data structure is compressed according to a compression scheme that takes advantage of the sparseness of valid entries in the set of tables.

9 12. (Original) A method according to claim 8 11, wherein the compression scheme combines two or more tables that have no conflicting common entries.

10
~~13.~~ (Currently amended) A method ~~according to claim 1, further comprising:~~ of performing protocol specific operations on a packet passing through a connection point on a computer network, the method comprising:

(a) receiving the packet;

(b) receiving a set of protocol descriptions for a plurality of protocols that conform to a layered model, a protocol description for a particular protocol at a particular layer level including:

(i) if there is at least one child protocol of the protocol at the particular layer level, the one or more child protocols of the particular protocol at the particular layer level, the packet including for any particular child protocol of the particular protocol at the particular layer level information at one or more locations in the packet related to the particular child protocol,

(ii) the one or more locations in the packet where information is stored related to any child protocol of the particular protocol, and

(iii) if there is at least one protocol specific operation to be performed on the packet for the particular protocol at the particular layer level, the one or more protocol specific operations to be performed on the packet for the particular protocol at the particular layer level; and

(c) performing the protocol specific operations on the packet specified by the set of protocol descriptions based on the base protocol of the packet and the children of the protocols used in the packet,

wherein the protocol specific operations include one or more parsing and extraction operations on the packet to extract selected portions of the packet to form a function of the selected portions for identifying the packet as belonging to a conversational flow.

13 10
~~14.~~ (Currently amended) A method according to claim ~~13~~, wherein the protocol descriptions are provided in a protocol description language.

14 13
~~15.~~ (Original) A method according to claim ~~14~~, further comprising:

compiling the PDL descriptions to produce a database and store the database in a memory, the database generated from the set of protocol descriptions and including a data structure containing information on the possible protocols and organized for locating the child protocol related information for any protocol, the data structure contents indexed by a set of one or more indices, the database entry indexed by a particular set of index values including an indication of validity,

wherein the child protocol related information includes a child recognition pattern, and

wherein the step of performing the protocol specific operations includes, at any particular protocol layer level starting from the base level, searching the packet at the particular protocol for the child field, the searching including indexing the data structure until a valid entry is found,

whereby the data structure is configured for rapid searches using the index set.

16. (Currently amended) A method according to claim 13, further comprising:

looking up a flow-entry database comprising ~~none or more flow-entries,~~ at least one flow-entry for each previously encountered conversational flow, the looking up using at least some of the selected packet portions and determining if the packet matches an ~~existing~~ flow-entry in the flow-entry database

if the packet is of an existing flow, classifying the packet as belonging to the found existing flow; and

if the packet is of a new flow, storing 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~~,~~

wherein for at least one protocol, the parsing and extraction operations depend on the contents of ~~none~~ one or more packet headers.

17. (Original) A method according to claim 13, wherein the protocol specific operations further include one or more state processing operations that are a function of the state of the flow of the packet.

18. (Original) A method ~~according to claim 1,~~ of performing protocol specific operations on a packet passing through a connection point on a computer network, the method comprising:

(a) receiving the packet;

(b) receiving a set of protocol descriptions for a plurality of protocols that conform to a layered model, a protocol description for a particular protocol at a particular layer level including:

(i) if there is at least one child protocol of the protocol at the particular layer level, the one or more child protocols of the particular protocol at the particular layer level, the packet including for any particular child protocol of the particular protocol at the particular layer level information at one or more locations in the packet related to the particular child protocol,

(ii) the one or more locations in the packet where information is stored related to any child protocol of the particular protocol, and

(iii) if there is at least one protocol specific operation to be performed on the packet for the particular protocol at the particular layer level, the one or more protocol specific operations to be performed on the packet for the particular protocol at the particular layer level; and

(c) performing the protocol specific operations on the packet specified by the set of protocol descriptions based on the base protocol of the packet and the children of the protocols used in the packet,

wherein the protocol specific operations include one or more state processing operations that are a function of the state of the flow of the packet.

REMARKS

Status of the Application:

Claims 1–18 are the claims of record of the application. Claims 1–3, 13, 14, 16, 17 and 18 have been rejected and claims 4–11, and 15 have been objected to as being dependent upon a rejected base claim, but would be allowable if rewritten in independent form.

Note that the examiner did not explicitly mention claim 12. As claim 12 depends on claim 11 and claim 11 was objected to as being dependent upon a rejected base claim, but would be allowable if rewritten in independent form, the same is assumed to apply to claim 12.

Amendment to the Claims:

Claim 1 has been cancelled. Some of the remaining claims were amended to not depend on any cancelled claim. Furthermore, the none or more phrases in the claims was amended.

Claim Rejections -35 USC § 112 Second Paragraph (Indefiniteness)

In paragraph 3 of the office action, claims 1 and 16 were rejected under 35 USC 112, second paragraph, as being indefinite. In particular the examiner asserted that the phrase "none or more" renders the claims indefinite.

Applicants respectfully disagree that the phrase "none or more" is indefinite in this context. The phrasing is common in computer language descriptions, and the meaning would be clear to those in the art. Nevertheless, the recitations in the claims that include "none or more" have been amended to indicate the definite meaning. In claim 1, for example, the recitation of "the none or more child protocols of the particular protocol" has been amended to "if there is at least one child protocol of the particular protocol at the particular layer level, the one or more child protocols of the particular protocol at the particular layer level, the packet including for any particular child protocol of the particular protocol information at one or more locations in the packet related to the particular child protocol.

Claim Rejections -35 USC § 102

In paragraph 5 of the office action, claims 1-3, 13, 14, 17 and 18 were rejected under 35 USC 102(b) as being anticipated by Bruell (U.S. Patent 5,680,585).

Claim 1 has been canceled and claims 2, 3 and 14 have been amended to depend on claim 13 that is believed to be allowable.

The rejection of claims 13, 17 and 18 are believed to be erroneous.

Description of Bruell.

Bruell describes a language for describing different packet formats. A packet description language file describes a format. A compiler translates the packet description language file

into a data structure. Application programs may both encode data into packets according to the defined format as well as decode packets that were assembled according to the protocol. To do this, application programs need only reference a data structure resulting from the compiled packet description language file. In this manner, numerous data packets formats may be defined in accordance with different data transfer media and packet protocols.

Claim 13

Regarding claim 13, the examiner asserts that Bruell discloses protocol specific operations that include one or more parsing and extraction operations on the packet to extract selected portions of the packet to form a function of the selected portions for identifying the packet as belonging to a conversational flow.

Applicants' respectfully disagree.

First, the examiner has failed to show that any protocol specific operations disclosed in Bruell are the protocol specific operations of step (c).

In the rejection of original claim 1 (now cancelled but incorporated into claim 13), the examiner asserts that Bruell in col. 9 line 8 to col. 10 line 43, and in col. 14, line 37 to col. 15, line 10 describes that the protocol description for a particular protocol at a particular layer level includes any protocol specific operations to be performed on the packet for the particular protocol at the particular layer level. The examiner also asserts that "test using packet description files" in Bruell are such protocol specific operations. Col. 9 line 8 to col. 10 line 43 of Bruell describes protocols and how Bruell's language can be used to describe how to interpret protocols, in particular, how to decode a packet based on the protocol. Col. 14, line 37 to col. 15, line 10 of Bruell describes how the language may be used to specify filtering. Thus, Bruell describes using the language to define a set of protocol specific operations. However, the examiner has not shown that in Bruell the feature the descriptions of the protocols themselves, i.e., the protocol description of the protocol describe the protocol specific operations to be performed.

In the rejection of original claim 1 (now cancelled but incorporated into claim 13), the examiner also asserts that Bruell in FIG. 4 and in col. 15 line 11–col. 16, line 42 discloses step (c) of performing the protocol specific operations on the packet specified by the set of protocol descriptions based on the base protocol of the packet and the children of the protocols used in the packet.

Applicants respectfully disagree. While the cited passage describes some protocol specific operations, protocol specific operations are part of what Bruell calls test application routines. For example, Bruell states that the test application routines

> "may include a send routine 410, a receive routine 420 and a tap routine 430. The test application routines each refer to the PDL data structures 405 for carrying out their respective functions with respect to the device under test 305."

6-13-03

Thus, one may argue that Bruell describes performing protocol specific operations. In Applicants' invention, any protocol specific operations to be performed on the packet for a particular protocol as part of step (c) are included in the protocol description for the particular protocol at a particular layer level (See restriction (iii) of step (b) that describes what is included in the protocol description for a particular protocol). However, in Bruell, the protocol description of the protocol does not describe the protocol description operations to be performed. Rather, the cited part of Bruell describes how test application routines may be written and how such written application routines, e.g., filtering, may use compiled protocol descriptions.

In the rejection of claim 13, the examiner asserts that "decoding packets in accordance with a defined packet format" describes "parsing and extraction operations on the packet to extract selected portions of the packet to form a function of the selected portions for identifying the packet as belonging to a conversational flow" and further that this is disclosed in col. 4 line 49 to col 6, line 30 and col. 14, line 37 to col. 15, line 10 of Bruell.

Applicants respectfully disagree.

Those in the art would understand that decoding a packet is the determining of the payload at each layer according to the protocol. This may include parsing, may include extraction operations, and may include extracting selected portions of the packet. However, the examiner has failed to show that Bruell discloses the feature of the extraction being to form a function of the selected portions for identifying the packet as belonging to a conversational flow. Applicants invention is in order to recognize packets, e.g., that pass through a node in a network, as belonging to a conversational flow. A conversational flow is the set of packets of a conversation. The function of the extracted portions is used to recognize a packet as belonging to a conversational flow. The examiner has failed to show that Bruell discloses such forming of a function.

Thus, claim 13 is believed allowable. Action to that end is respectfully requested.

Claim 17

Regarding claim 17, the examiner asserts that Bruell discloses that the protocol specific operations include one or more state processing operations that are a function of the state of the flow of the packet. In particular, that Bruell's FIG. 1 and col. 3, line 20 to col. 4, line 33, and col. 14, line 38 to col. 15, line 10 describe this feature.

Claim 17 depends on claim 13. The rejection of claim 13 is believed overcome. The above arguments with respect to claim 13 are incorporated herein by reference. Thus claim 17 is believed allowable and action to that end is respectfully requested.

However, even if the examiner remains unconvinced by applicant's arguments for overcoming the rejection of claim 13, Applicants still believe the examiner's arguments for rejecting claim 17 are erroneous.

State processing that depends on the state of a conversational flow is a concept described in the specification. A conversational flow is the set of packets of a conversation. The state of

a flow is described in the specification as an indication of all previous events in the flow. State processing thus is processing that depends on the state of a conversational flow, i.e., on the sequence of one or more previously encountered packets of the same conversational flow (or initial state in the case of the first packet in a conversational flow). The examiner has failed to show that Bruell includes this feature. For example, the word "state" does not even appear in Bruell.

The cited FIG. 1 of Bruell shows the process of constructing a packet. The cited part on cols. 3 and 4 of Bruell describes FIG. 1 and also the use of Bruell's language to define protocols and to process a single packet. No conversational flows or state processing are disclosed. The cited part on cols. 14 and 15 of Bruell describes how Bruell's packet description language (PDL) may be advantageously implemented in a system for testing internetwork routing devices (the device test environment 301). See FIG. 3. A user defines one or several PDL files 302. To use the device test environment 301, a user creates a test file 303 that specifies the number, type and optional content of packets for the device test environment 301 to send to and receive from a device under test 305. The remainder of the cited part of Bruell is repeated here:

The device test environment 301 follows the script created by the test files with reference to the data packet formats defined in the PDL files. The PDL files determine the structure and default content of each packet type. When a device test environment 301 reads an instruction in a test file for the device under test to send a type of packet, it assembles the test packet by referring to the packet assembly specification in the PDL files 302. For example, if the device test environment reads a test file instruction to send an Ethernet header (ENET.sub.-- HDR) packet, it looks for the ENET.sub.-- HDR specification in the PDL files and assembles the packet accordingly. Similarly, when the device test environment reads an instruction in a test file for the device under test to receive a type of packet, a test packet is provided for the device to receive. If the packet the test environment reads matches the packet type in the PDL files, then the device test environment 301 reports that the test succeeded; otherwise, it reports that it failed.

There is no concept of a conversational flow or of the state of the flow of the packet. Bruell discloses testing individual packets. Thus, applicants assert, the examiner has failed to show that Bruell describes state processing that is a function of the state of the flow of the packet.

The rejection of claim 17 is thus believed overcome and the claims are allowable. Action to that end is respectfully requested.

Claim 18

Regarding claim 18, the examiner asserts that Bruell discloses that the protocol specific operations include one or more state processing operations that are a function of the state of the flow of the packet. In particular, that Bruell's FIG. 1 and col. 3, line 20 to col. 4, line 33, and col. 14, line 38 to col. 15, line 10 describe this feature.

6-13-03

First, the examiner has failed to show that any protocol specific operations disclosed in Bruell are the protocol specific operations of step (c). The arguments presented above for this aspect of claim 13 also apply to the rejection of claim 18, as amended, and are incorporated herein by reference. Furthermore, the examiner has failed to show that Bruell discloses that the protocol specific operations include one or more state processing operations that are a function of the state of the flow of the packet. The arguments presented above for this aspect of claim 17 also apply to the rejection of claim 18 and are incorporated herein by reference.

The rejection of claim 18 is thus believed overcome and the claims are allowable. Action to that end is respectfully requested.

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,

13 June 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
Email: dov@inventek.com

6-13-03

INVENTEK

Fax

#8/B
LDJ
6-30-03
entered

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.: 09/609179	***Ref./Docket No***: APPT-001-2
Applicant(s): Dietz, *et al.*	***Examiner***.: Dinh, Khanh Q.
Filing Date: June 30, 2000	***Art Unit***: 2155

FAX COVER PAGE

TO: Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

United States Patent and Trademark Office
(Examiner Dinh, Khanh Q., Art Unit 2155)

Fax No.: 703-746-5510

DATE: June 27, 2003

FROM: Dov Rosenfeld, Reg. No. 38687

RE: Supplementary response: CLAIM 18 of Serial Number 09/609,179

Number of pages including cover: 6

Dear Sir,

Further to our telephone conversation yesterday, here is a revised amendment for claim 18, this time with a clean version included. I've included the same remarks on the rejection of claim 18 as was in the earlier supplemental response of 6/19/03 such that this forms a complete supplemental response.

Thank you very much,

Dov Rosenfeld

6/27/03

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 addressed the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: 6/27/03 Signed: _____

Name: Dov Rosenfeld, Reg. No. 38687

S/N 09/609179 (Our APPT-001-2)

Applicant(s): Dietz, *et al.* Application No.: 09/609179 Filed: June 30, 2000 Title: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE	Group Art Unit: 2155 Examiner: Dinh, Khanh Q.

SUPPLEMENTAL RESPONSE

Mail Stop Non Fee Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear sir,

Further to the telephone conversations between the examiner and the undersigned regarding claim 18, here is a revised amendment to claim 18.

AMENDMENT TO CLAIM 18:

Kindly amend claim 18 to read as follows. A recitation showing the deletions and additions follows on a separate sheet:

17 ~~18.~~ A method of performing protocol specific operations on a packet passing through a connection point on a computer network, the method comprising:

(a) receiving the packet;

B1

(b) receiving a set of protocol descriptions for a plurality of protocols that conform to a layered model, a protocol description for a particular protocol at a particular layer level including:

(i) if there is at least one child protocol of the protocol at the particular layer level, the one or more child protocols of the particular protocol at the particular layer level, the packet including for any particular child protocol of the particular protocol at the particular layer level information at one or more locations in the packet related to the particular child protocol,

(ii) the one or more locations in the packet where information is stored related to any child protocol of the particular protocol, and

(iii) if there is at least one protocol specific operation to be performed on the packet for the particular protocol at the particular layer level, the one or more protocol specific operations to be performed on the packet for the particular protocol at the particular layer level; and

(c) performing the protocol specific operations on the packet specified by the set of protocol descriptions based on the base protocol of the packet and the children of the protocols used in the packet,

B1

end

wherein the packet belongs to a conversational flow of packets having a set of one or more states, and wherein the protocol specific operations include one or more state processing operations that are a function of the state of the conversational flow of the packet, the state of the conversational flow of the packet being indicative of the sequence of any previously encountered packets of the same conversational flow as the packet.

Description of amendment to claim 18:

18. (Currently amended) A method ~~according to claim 1,~~ of performing protocol specific operations on a packet passing through a connection point on a computer network, the method comprising:

(a) receiving the packet;

(b) receiving a set of protocol descriptions for a plurality of protocols that conform to a layered model, a protocol description for a particular protocol at a particular layer level including:

(i) if there is at least one child protocol of the protocol at the particular layer level, the one or more child protocols of the particular protocol at the particular layer level, the packet including for any particular child protocol of the particular protocol at the particular layer level information at one or more locations in the packet related to the particular child protocol,

(ii) the one or more locations in the packet where information is stored related to any child protocol of the particular protocol, and

(iii) if there is at least one protocol specific operation to be performed on the packet for the particular protocol at the particular layer level, the one or more protocol specific operations to be performed on the packet for the particular protocol at the particular layer level; and

(c) performing the protocol specific operations on the packet specified by the set of protocol descriptions based on the base protocol of the packet and the children of the protocols used in the packet,

wherein the packet belongs to a conversational flow of packets having a set of one or more states, and wherein the protocol specific operations include one or more state processing operations that are a function of the state of the conversational flow of the packet, the state of the conversational flow of the packet being indicative of the sequence of any previously encountered packets of the same conversational flow as the packet.

REMARKS ON THE REJECTION OF CLAIM 18

Regarding claim 18, the examiner asserts that Bruell discloses that the protocol specific operations include one or more state processing operations that are a function of the state of the flow of the packet. In particular, that Bruell's FIG. 1 and col. 3, line 20 to col. 4, line 33, and col. 14, line 38 to col. 15, line 10 describe this feature.

Applicants respectfully disagree.

State processing that depends on the state of a conversational flow is a concept described in the specification. A conversational flow is the set of packets of a conversation. A conversational flow has a set of one or more states. The state of a flow is described in the specification as an indication of all previous events in the flow. State processing thus is processing that depends on the state of a conversational flow the packet belongs to, i.e., on the sequence of any previously encountered packets of the same conversational flow as the packet. The examiner has failed to show that Bruell includes this feature. For example, the word "state" does not even appear in Bruell.

The cited FIG. 1 of Bruell shows the process of constructing a packet. The cited part on cols. 3 and 4 of Bruell describes FIG. 1 and also the use of Bruell's language to define protocols and to process a single packet. No conversational flows or state processing are disclosed. The cited part on cols. 14 and 15 of Bruell describes how Bruell's packet description language (PDL) may be advantageously implemented in a system for testing internetwork routing devices (the device test environment 301). See FIG. 3. A user defines one or several PDL files 302. To use the device test environment 301, a user creates a test file 303 that specifies the number, type and optional content of packets for the device test environment 301 to send to and receive from a device under test 305. The remainder of the cited part of Bruell is repeated here:

> *The device test environment 301 follows the script created by the test files with reference to the data packet formats defined in the PDL files. The PDL files determine the structure and default content of each packet type. When a device test environment 301 reads an instruction in a test file for the device under test to send a type of packet, it assembles the test packet by referring to the packet assembly specification in the PDL files 302. For example, if the device test environment reads a test file instruction to send an Ethernet header (ENET.sub.-- HDR) packet, it looks for the ENET.sub.-- HDR specification in the PDL files and assembles the packet accordingly. Similarly, when the device test environment reads an instruction in a test file for the device under test to receive a type of packet, a test packet is provided for the device to receive. If the packet the test environment reads matches the packet type in the PDL files, then the device test environment 301 reports that the test succeeded; otherwise, it reports that it failed.*

There is no concept of a conversational flow or of the state of the flow of the packet. Bruell discloses testing individual packets. Thus, applicants assert, the examiner has failed to

show that Bruell describes state processing that is a function of the state of the flow of the packet.

The rejection of claim 18 is thus believed overcome and the 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,

June 27, 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

Notice of Allowability	Application No.	Applicant(s)	
	09/609,179	DIETZ ET AL.	
	Examiner	Art Unit	
	Khanh Dinh	2155	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address--

All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance (PTOL-85) or other appropriate communication will be mailed in due course. **THIS NOTICE OF ALLOWABILITY IS NOT A GRANT OF PATENT RIGHTS.** This application is subject to withdrawal from issue at the initiative of the Office or upon petition by the applicant. See 37 CFR 1.313 and MPEP 1308.

1. ☒ This communication is responsive to *6/13/2003*.
2. ☒ The allowed claim(s) is/are *2-18*.
3. ☒ The drawings filed on *6/30/2000* are accepted by the Examiner.
4. ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
   a) ☐ All b) ☐ Some* c) ☐ None of the:
      1. ☐ Certified copies of the priority documents have been received.
      2. ☐ Certified copies of the priority documents have been received in Application No. _____ .
      3. ☐ Copies of the certified copies of the priority documents have been received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

   * Certified copies not received: _____.
5. ☐ 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.
6. ☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121.

Applicant has THREE MONTHS FROM THE "MAILING DATE" of this communication to file a reply complying with the requirements noted below. Failure to timely comply will result in ABANDONMENT of this application. **THIS THREE-MONTH PERIOD IS NOT EXTENDABLE**

7. ☐ A SUBSTITUTE OATH OR DECLARATION must be submitted. Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL PATENT APPLICATION (PTO-152) which gives reason(s) why the oath or declaration is deficient.

8. ☐ CORRECTED DRAWINGS must be submitted.
   (a) ☐ including changes required by the Notice of Draftsperson's Patent Drawing Review (PTO-948) attached
      1) ☐ hereto or 2) ☐ to Paper No. ____.
   (b) ☐ including changes required by the proposed drawing correction filed _____, which has been approved by the Examiner.
   (c) ☐ including changes required by the attached Examiner's Amendment / Comment or in the Office action of Paper No. _____.

   Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the drawings in the front (not the back) of each sheet.

9. ☐ DEPOSIT OF and/or INFORMATION about the deposit of BIOLOGICAL MATERIAL must be submitted. Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Attachment(s)

1☐ Notice of References Cited (PTO-892)
2☐ Notice of Informal Patent Application (PTO-152)
3☐ Notice of Draftperson's Patent Drawing Review (PTO-948)
4☐ Interview Summary (PTO-413), Paper No.____ .
5☐ Information Disclosure Statements (PTO-1449), Paper No. ____.
6☐ Examiner's Amendment/Comment
7☐ Examiner's Comment Regarding Requirement for Deposit of Biological Material
8☐ Examiner's Statement of Reasons for Allowance
9☐ Other .

HOSAIN T. ALAM
PRIMARY EXAMINER

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

NOTICE OF ALLOWANCE AND FEE(S) DUE

7590 07/01/2003

Dov Rosenfeld
5507 College Avenue
Suite 2
Oakland, CA 94618

EXAMINER	
DINH, KHANH Q	
ART UNIT	**CLASS-SUBCLASS**
2155	709-230000

DATE MAILED: 07/01/2003

APPLICATION NO	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO	CONFIRMATION NO.
09/609,179	06/30/2000	Russell S. Dietz	APPT-001-2	2668

TITLE OF INVENTION: METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK

APPLN TYPE	SMALL ENTITY	ISSUE FEE	PUBLICATION FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	NO	$1300	$0	$1300	10/01/2003

THE APPLICATION IDENTIFIED ABOVE HAS BEEN EXAMINED AND IS ALLOWED FOR ISSUANCE AS A PATENT. PROSECUTION ON THE MERITS IS CLOSED. THIS NOTICE OF ALLOWANCE IS NOT A GRANT OF PATENT RIGHTS. THIS APPLICATION IS SUBJECT TO WITHDRAWAL FROM ISSUE AT THE INITIATIVE OF THE OFFICE OR UPON PETITION BY THE APPLICANT. SEE 37 CFR 1.313 AND MPEP 1308.

THE ISSUE FEE AND PUBLICATION FEE (IF REQUIRED) MUST BE PAID WITHIN THREE MONTHS FROM THE MAILING DATE OF THIS NOTICE OR THIS APPLICATION SHALL BE REGARDED AS ABANDONED. THIS STATUTORY PERIOD CANNOT BE EXTENDED. SEE 35 U.S.C. 151. THE ISSUE FEE DUE INDICATED ABOVE REFLECTS A CREDIT FOR ANY PREVIOUSLY PAID ISSUE FEE APPLIED IN THIS APPLICATION. THE PTOL-85B (OR AN EQUIVALENT) MUST BE RETURNED WITHIN THIS PERIOD EVEN IF NO FEE IS DUE OR THE APPLICATION WILL BE REGARDED AS ABANDONED.

HOW TO REPLY TO THIS NOTICE:

I. Review the SMALL ENTITY status shown above.

If the SMALL ENTITY is shown as YES, verify your current SMALL ENTITY status:

A. If the status is the same, pay the TOTAL FEE(S) DUE shown above.

B. If the status is changed, pay the PUBLICATION FEE (if required) and twice the amount of the ISSUE FEE shown above and notify the United States Patent and Trademark Office of the change in status, or

If the SMALL ENTITY is shown as NO:

A. Pay TOTAL FEE(S) DUE shown above, or

B. If applicant claimed SMALL ENTITY status before, or is now claiming SMALL ENTITY status, check the box below and enclose the PUBLICATION FEE and 1/2 the ISSUE FEE shown above.

☐ Applicant claims SMALL ENTITY status. See 37 CFR 1.27.

II. PART B - FEE(S) TRANSMITTAL should be completed and returned to the United States Patent and Trademark Office (USPTO) with your ISSUE FEE and PUBLICATION FEE (if required). Even if the fee(s) have already been paid, Part B - Fee(s) Transmittal should be completed and returned. If you are charging the fee(s) to your deposit account, section "4b" of Part B - Fee(s) Transmittal should be completed and an extra copy of the form should be submitted.

III. All communications regarding this application must give the application number. Please direct all communications prior to issuance to Box ISSUE FEE unless advised to the contrary.

IMPORTANT REMINDER: Utility patents issuing on applications filed on or after Dec. 12, 1980 may require payment of maintenance fees. It is patentee's responsibility to ensure timely payment of maintenance fees when due.

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: Mail **Mail Stop ISSUE FEE**
Commissioner for Patents
Alexandria, Virginia 22313-1450
Fax (703)746-4000

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 4 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note. Legibly mark-up with any corrections or use Block 1)

7590 07/01/2003

Dov Rosenfeld
5507 College Avenue
Suite 2
Oakland, CA 94618

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission

I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Box Issue Fee address above, or being facsimile transmitted to the USPTO, on the date indicated below.

(Depositor's name)
(Signature)
(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/609,179	06/30/2000	Russell S. Dietz	APPT-001-2	2668

TITLE OF INVENTION: METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK

APPLN. TYPE	SMALL ENTITY	ISSUE FEE	PUBLICATION FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	NO	$1300	$0	$1300	10/01/2003

EXAMINER	ART UNIT	CLASS-SUBCLASS
DINH, KHANH Q	2155	709-230000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).

❑ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.

❑ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. **Use of a Customer Number is required.**

2. For printing on the patent front page, list (1) the names of up to 3 registered patent attorneys or agents OR, alternatively, (2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.

1 _____

2 _____

3 _____

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. Inclusion of assignee data is only appropriate when an assignment has been previously submitted to the USPTO or is being submitted under separate cover. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE (B) RESIDENCE: (CITY and STATE OR COUNTRY)

Please check the appropriate assignee category or categories (will not be printed on the patent) ❑ individual ❑ corporation or other private group entity ❑ government

4a. The following fee(s) are enclosed:

❑ Issue Fee

❑ Publication Fee

❑ Advance Order - # of Copies _____

4b. Payment of Fee(s):

❑ A check in the amount of the fee(s) is enclosed.

❑ Payment by credit card. Form PTO-2038 is attached.

❑ The Commissioner is hereby authorized by charge the required fee(s), or credit any overpayment, to Deposit Account Number _____(enclose an extra copy of this form).

Commissioner for Patents is requested to apply the Issue Fee and Publication Fee (if any) or to re-apply any previously paid issue fee to the application identified above.

(Authorized Signature) (Date)

NOTE; The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office.

This collection of information is required by 37 CFR 1.311. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, Alexandria, Virginia 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, Alexandria, Virginia 22313-1450.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

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.
09/609,179	06/30/2000	Russell S. Dietz	APPT-001-2	2668

7590 07/01/2003

Dov Rosenfeld
5507 College Avenue
Suite 2
Oakland, CA 94618

EXAMINER	
DINH, KHANH Q	
ART UNIT	**PAPER NUMBER**
2155	9

DATE MAILED: 07/01/2003

Determination of Patent Term Adjustment under 35 U.S.C. 154 (b)
(application filed on or after May 29, 2000)

The patent term adjustment to date is 643 days. If the issue fee is paid on the date that is three months after the mailing date of this notice and the patent issues on the Tuesday before the date that is 28 weeks (six and a half months) after the mailing date of this notice, the term adjustment will be 643 days.

If a continued prosecution application (CPA) was filed in the above-identified application, the filing date that determines patent term adjustment is the filing date of the most recent CPA.

Applicant will be able to obtain more detailed information by accessing the Patent Application Information Retrieval (PAIR) system. (http://pair.uspto.gov)

Any questions regarding the patent term extension or adjustment determination should be directed to the Office of Patent Legal Administration at (703)305-1383.

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
09/609,179	06/30/2000	Russell S. Dietz	APPT-001-2	2668

7590 07/01/2003

EXAMINER
DINH, KHANH Q

Dov Rosenfeld
5507 College Avenue
Suite 2
Oakland, CA 94618
UNITED STATES

ART UNIT	PAPER NUMBER
2155	

DATE MAILED: 07/01/2003

Notice of Fee Increase on January 1, 2003

If a reply to a "Notice of Allowance and Fee(s) Due" is filed in the Office on or after January 1, 2003, then the amount due will be higher than that set forth in the "Notice of Allowance and Fee(s) Due" since there will be an increase in fees effective on January 1, 2003. See Revision of Patent and Trademark Fees for Fiscal Year 2003; Final Rule, 67 Fed. Reg. 70847, 70849 (November 27, 2002).

The current fee schedule is accessible from: http://www.uspto.gov/main/howtofees.htm.

If the issue fee paid is the amount shown on the "Notice of Allowance and Fee(s) Due," but not the correct amount in view of the fee increase, a "Notice to Pay Balance of Issue Fee" will be mailed to applicant. In order to avoid processing delays associated with mailing of a "Notice to Pay Balance of Issue Fee," if the response to the Notice of Allowance and Fee(s) due form is to be filed on or after January 1, 2003 (or mailed with a certificate of mailing on or after January 1, 2003), the issue fee paid should be the fee that is required at the time the fee is paid. If the issue fee was previously paid, and the response to the "Notice of Allowance and Fee(s) Due" includes a request to apply a previously-paid issue fee to the issue fee now due, then the difference between the issue fee amount at the time the response is filed and the previously paid issue fee should be paid. See Manual of Patent Examining Procedure, Section 1308.01 (Eighth Edition, August 2001).

Questions relating to issue and publication fee payments should be directed to the Customer Service Center of the Office of Patent Publication at (703) 305-8283.

PTOL-85 (REV. 05-03) Approved for use through 04/30/2004.

INVENTEK
Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618, USA
Phone: (510)547-3378; Fax: (510)653-7992
dov@inventek.com

Official Fax

RECEIVED
JUL 2 8 2003
Technology Center 2100

Patent Application Ser. No.: 09/609179	*Ref./Docket No*: APPT-001-2
Applicant(s): Dietz, *et al.*	*Examiner.*: Dinh, Khanh Q.
Filing Date: June 30, 2000	*Art Unit*: 2155

FAX COVER PAGE

TO: Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

United States Patent and Trademark Office
(Examiner Dinh, Khanh Q., Art Unit 2155)

Fax No.: 703-746-7239

DATE: July 08, 2003

FROM: Dov Rosenfeld, Reg. No. 38687

RE: Amendment after Allowance

Number of pages including cover: 6

OFFICIAL COMMUNICATION

PLEASE URGENTLY DELIVER A COPY OF THIS RESPONSE TO EXAMINER DINH, KHANH Q., ART UNIT 2155

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number 703-746-7239 addressed the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: July 8, 2003

Signed: _____
Name: Dov Rosenfeld, Reg. No. 38687

Official 7/8/03

TRANSMITTAL FORM

(to be used for all correspondence after initial filing)

Application Number	09/609179
Filing Date	30 Jun 2000
First Named Inventor	Dietz, Russell S.
Group Art Unit	2155
Examiner Name	Dinh, Khanh Q.
Attorney Docket Number	APPT-001-2

RECEIVED
JUL 2 8 2003
Technology Center 2100

ENCLOSURES *(check all that apply)*

- ☐ Fee Transmittal Form
- ☐ Fee Attached
- ☒ 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
- ☐ 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
- ☐
- ☐

Remarks

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT/ CORRESPONDENCE ADDRESS

Firm or Individual name	Dov Rosenfeld, Reg. No. 38687
Signature	
Date	July 8, 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: July 8, 2003

Type or printed name	Dov Rosenfeld, Reg. No. 38687		
Signature		Dat	July 8, 2003

Our Ref./Docket No: APPT-001-2 Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz, *et al.*	Group Art Unit: 2155
Application No.: 09/609179	Examiner: Dinh, Khanh Q.
Filed: June 30, 2000	Notice of Allowance mailed: July 1, 2003
Title: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE	Confirmation No.: 2668

AMENDMENT AFTER ALLOWANCE UNDER 37 CFR 1.312

RECEIVED
JUL 2 8 2003
Technology Center 2100

Mail Stop Non Fee Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

This is an amendment after allowance under 37 CFR 1.312.

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number 703-746-7239 addressed the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: July 8, 2003 Signed:

Name: Dov Rosenfeld, Reg. No. 38687

INTRODUCTORY REMARKS:

Kindly amend this application as follows and kindly consider the following remarks.

AMENDMENT TO THE TITLE

Kindly delete the title of record and substitute the following title therefor:

--PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE--

REMARKS

Status of the Application:

A Notice of Allowance was mailed on July 1, 2003.

Amendment to the Title:

Upon receipt of the Notice of Allowance, it was noticed that the title cited on the Notice was wrongly written as

METHOD AND APPRATUS FOR MONITIRING TRAFFIC IN A NETWORK

This is not the title of the invention as filed. A filing receipt was issued on November 7, 2000 with this wrong title. The application was filed on June 20, 2000 with the correct title, which is:

PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE

Correction of the title is respectfully requested.

Applicants understand that an amendment after allowance under 37 CFR 1.312 is not a right, but is discretionary. The original error was the error of the Patent Office. Applicants respectfully request that this amendment be entered.

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,

July 8, 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

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.: 09/609179	*Ref./Docket No*: APPT-001-2
Applicant(s): Dietz, *et al.*	*Examiner.*: Dinh, Khanh Q.
Filing Date: June 30, 2000	*Art Unit*: 2155

FAX COVER PAGE

TO: Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

United States Patent and Trademark Office
(Examiner Dinh, Khanh Q., Art Unit 2155)

Fax No.: 703-746-7238

DATE: July 14, 2003

FROM: Dov Rosenfeld, Reg. No. 38687

RE: Amendment after Allowance

Number of pages including cover: 6

OFFICIAL COMMUNICATION

PLEASE URGENTLY DELIVER A COPY OF THIS RESPONSE TO EXAMINER DINH, KHANH Q., ART UNIT 2155

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number 703-746-7238 addressed the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: 7/14/03 Signed: _____

Name: Dov Rosenfeld, Reg. No. 38687

Our Ref./Docket No: APPT-001-2 Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz, *et al.*	Group Art Unit: 2155
Application No.: 09/609179	Examiner: Dinh, Khanh Q.
Filed: June 30, 2000	Notice of Allowance mailed: July 1, 2003
Title: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE	Confirmation No.: 2668

AMENDMENT AFTER ALLOWANCE UNDER 37 CFR 1.312

Mail Stop Non Fee Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

This is an amendment after allowance under 37 CFR 1.312.

This amendment was previously sent July 8, 2003 to the non-after final fax number for the Art Unit, and is now being sent to the after-final fax number per examiner request.

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number 703-746-7238 addressed the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: 7/14/03 Signed:

Name: Dov Rosenfeld, Reg. No. 38687

Received from <+1 510 291 2985> at 7/14/03 2:15:18 PM [Eastern Daylight Time]

#10/C (N/E.) LDD 10-22-03 entered LDD 10-27-03

OK to enter K.D.N.P 10/23/03

Match and Return

INTRODUCTORY REMARKS:

Kindly amend this application as follows and kindly consider the following remarks.

AMENDMENT TO THE TITLE

Kindly delete the title of record and substitute the following title therefor:

--PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE--

REMARKS

Status of the Application:

A Notice of Allowance was mailed on July 1, 2003.

Amendment to the Title:

Upon receipt of the Notice of Allowance, it was noticed that the title cited on the Notice was wrongly written as

METHOD AND APPRATUS FOR MONITIRING TRAFFIC IN A NETWORK

This is not the title of the invention as filed. A filing receipt was issued on November 7, 2000 with this wrong title. The application was filed on June 20, 2000 with the correct title, which is:

PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE

Correction of the title is respectfully requested.

Applicants understand that an amendment after allowance under 37 CFR 1.312 is not a right, but is discretionary. The original error was the error of the Patent Office. Applicants respectfully request that this amendment be entered.

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,

7/14/03
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
Email: dov@inventek.com

TRANSMITTAL FORM

(to be used for all correspondence after initial filing)

Application Number	09/609179
Filing Dat	30 Jun 2000
First Named Invent r	Dietz, Russell S.
Group Art Unit	2155
Examiner Name	Dinh, Khanh Q.
Attorney Docket Number	APPT-001-2

ENCLOSURES *(check all that apply)*

- ☐ Fee Transmittal Form
- ☐ Fee Attached
- ☒ 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
- ☐ 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
- ☐
- ☐

Remarks

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT/ CORRESPONDENCE ADDRESS

Firm or Individual name	Dov Rosenfeld, Reg. No. 38667
Signature	
Date	July 14, 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 703-746-7238 addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on this date: July 14, 2003

Type or printed name	Dov Rosenfeld, Reg. No. 38667		
Signature		Date	July 14, 2003

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.
09/609,179	06/30/2000	Russell S. Dietz	APPT-001-2	2668

7590 10/27/2003

Dov Rosenfeld
5507 College Avenue
Suite 2
Oakland, CA 94618

EXAMINER	
DINH, KHANH Q	

ART UNIT	PAPER NUMBER
2155	11

DATE MAILED: 10/27/2003

Please find below and/or attached an Office communication concerning this application or proceeding.

PR4

Response to Rule 312 Communication	Application No.	Applicant(s)	
	09/609,179	DIETZ ET AL.	
	Examiner	Art Unit	
	Khanh Dinh	2155	

-- *The MAILING DATE of this communication appears on the cover sheet with the correspondence address* --

1. ☒ The amendment filed on *14 July 2003* under 37 CFR 1.312 has been considered, and has been:

a) ☒ entered.

b) ☐ entered as directed to matters of form not affecting the scope of the invention.

c) ☐ disapproved because the amendment was filed after the payment of the issue fee.

Any amendment filed after the date the issue fee is paid must be accompanied by a petition under 37 CFR 1.313(c)(1) and the required fee to withdraw the application from issue.

d) ☐ disapproved. See explanation below.

e) ☐ entered in part. See explanation below.

HOSAIN ALAM
SUPERVISORY PATENT EXAMINER

K.DINH
10/23/03
A.U. 2155

Our Ref./Docket No: APPT-001-2 Patent

1 of 2 C. #12 ttk

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

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

Assignee: Hi/fn, Inc.

Patent No: 6,665,725 B1

Issue Date: December, 16, 2003

Application No.: 09/609,179

Filed: June 30, 2000

Title: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE

Certificate
APR 1 2 2004
of Correction

REQUEST FOR CERTIFICATE OF CORRECTIONS

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

The above patent contains significant errors as indicated on the attached Certificate of Correction form (submitted in duplicate).

__X__ Such errors arose through the fault of the Patent and Trademark Office. It is requested that the certificate be issued at no cost to the applicant.

However, if it is determined that the error(s) arose through the fault of applicant(s), please note that such error is of clerical error or minor nature and occurred in good faith and therefore issuance of the certificate of Correction is respectfully requested. The Commissioner is authorized to charge Deposit Account No. 50-0292 any required fee. A duplicate of this request is attached.

_____ Such error arose through the fault of applicant(s). A credit card charge form for the fee is enclosed. Such error is of clerical error or minor nature and occurred in good faith and therefore issuance of the certificate of Correction is respectfully requested.

Such errors specifically:

In col. 6, line 47 change "NBTBIOS" to --NETBIOS--.

Certificate of Mailing under 37 CFR 1.8

I hereby certify that this response is being deposited with the United States Postal Service as first class mail in an envelope addressed to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: Apr 5, 2004 Signed: Amy Drury

Name: Amy Drury

In col. 6, line 55 change "Diferent" to --Different--.

In col. 16, line 27 change "FIG. 6 FIG 6" to
--FIG 6.
FIG6--.

In col. 18, line 17 change "updatelookup" to --update-lookup--.

In col. 25, line 38 change "server-say" to --server—say--.

In col. 53, line 4 change ""Default"" to --"Default" :--.

In col. 53, line 45 shift "DISPLAY-HINT" to the right so its beginning lines up with the beginning of "SYNTAX" in line 42 and with the beginning of "LENGTH" in line 43.

In col. 53 line 46 shift "FLAGS" to the right so its beginning lines up with the beginning of "SYNTAX" in line 42 and with the beginning of "LENGTH" in line 43.

In col. 61, aprox. line 32 change "rip" to --r1p--.

In col 71, 9th line from the bottom change "netbios (0x3c00," to --netbios (0x3c00)--.

In col. 73, aprox. line 25 change "tyop" to --type--.

In col. 79, 4th line from the bottom change "SYNTAXINT(8)" to --SYNTAX INT (8)--.

In col. 81, approx. line 41 change "SYNTAXBITSRING(12)" to --SYNTAX BITSTRING (12)--.

In col. 83, approx. line 36 change "LOOKUPFILE" to --LOOKUP FILE--.

In col. 93, approx. line 45 change "'vnd.m-relaudio" to --'vnd.rn-realaudio'--.

In col. 96, line 38, change "In" to --in--.

The undersigned requests being contacted at (510) 547-3378 if there are any questions or clarifications, or if there are any problems with issuance of the Certificate of Correction.

Respectfully Submitted,

Apr. 5, 2004
Date

Dov Rosenfeld, Reg. No. 38687
Agent of Record.

Address for correspondence:
Dov Rosenfeld
5507 College Avenue, Suite 2,
Oakland, CA 94618 Tel. (510)547-3378; Fax: (510)291-2985

PTO/SB/44 (10-96)
Approved for use through 6/30/99. OMB 0651-0033
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO : 6,665,725 B1

DATED : December 16, 2003

INVENTOR(S) : Dietz, et al.

It is certified that an error appears in the above-identified patent and that said Letters Patent are hereby corrected as shown below:

In col. 6, line 47 change "NBTBIOS" to --NETBIOS--.

In col. 6, line 55 change "Diferent" to --Different--.

In col. 16, line 27 change "FIG. 6 FIG 6" to
--FIG 6.
FIG6--.

In col. 18, line 17 change "updatelookup" to --update-lookup--.

In col. 25, line 38 change "server-say" to --server—say--.

In col. 53, line 4 change ""Default"" to --"Default" :--.

In col. 53, line 45 shift "DISPLAY-HINT" to the right so its beginning lines up with the beginning of "SYNTAX" in line 42 and with the beginning of "LENGTH" in line 43.

In col. 53 line 46 shift "FLAGS" to the right so its beginning lines up with the beginning of "SYNTAX" in line 42 and with the beginning of "LENGTH" in line 43.

In col. 61, aprox. line 32 change "rip" to --r1p--.

In col 71, 9th line from the bottom change "netbios (0x3c00," to --netbios (0x3c00)--.

In col. 73, aprox. line 25 change "tyop" to --type--.

In col. 79, 4th line from the bottom change "SYNTAXINT(8)" to --SYNTAX INT (8)--.

In col. 81, approx. line 41 change "SYNTAXBITSRING(12)" to --SYNTAX BITSTRING (12)--.

In col. 83, approx. line 36 change "LOOKUPFILE" to --LOOKUP FILE--.

In col. 93, approx. line 45 change "'vnd.m-relaudio" to --'vnd.rn-realaudio'--.

In col. 96, line 38, change "In" to --in--.

MAILING ADDRESS OF SENDER (Atty/Agent of Record):
Dov Rosenfeld, Reg. No. 38687
5507 College Avenue, Suite 2
Oakland, CA 94618

PATENT NO: 6,665,725
No. of additional copies

APR 12 2004

PTO/SB/44 (10-96)
Approved for use through 6/30/99. OMB 0651-0033
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO : 6,665,725 B1

DATED : December 16, 2003

INVENTOR(S) : Dietz, et al.

It is certified that an error appears in the above-identified patent and that said Letters Patent are hereby corrected as shown below:

In col. 6, line 47 change "NBTBIOS" to --NETBIOS--.

In col. 6, line 55 change "Diferent" to --Different--.

In col. 16, line 27 change "FIG. 6 FIG 6" to
--FIG 6.
FIG6--.

In col. 18, line 17 change "updatelookup" to --update-lookup--.

In col. 25, line 38 change "server-say" to --server—say--.

In col. 53, line 4 change ""Default"" to --"Default" :--.

In col. 53, line 45 shift "DISPLAY-HINT" to the right so its beginning lines up with the beginning of "SYNTAX" in line 42 and with the beginning of "LENGTH" in line 43.

In col. 53 line 46 shift "FLAGS" to the right so its beginning lines up with the beginning of "SYNTAX" in line 42 and with the beginning of "LENGTH" in line 43.

In col. 61, aprox. line 32 change "rip" to --r1p--.

In col 71, 9th line from the bottom change "netbios (0x3c00," to --netbios (0x3c00)--.

In col. 73, aprox. line 25 change "tyop" to --type--.

In col. 79, 4th line from the bottom change "SYNTAXINT(8)" to --SYNTAX INT (8)--.

In col. 81, approx. line 41 change "SYNTAXBITSRING(12)" to --SYNTAX BITSTRING (12)--.

In col. 83, approx. line 36 change "LOOKUPFILE" to --LOOKUP FILE--.

In col. 93, approx. line 45 change "'vnd.m-relaudio" to --'vnd.rn-realaudio'--.

In col. 96, line 38, change "In" to --in--.

MAILING ADDRESS OF SENDER (Atty/Agent of Record):
Dov Rosenfeld, Reg. No. 38687
5507 College Avenue, Suite 2
Oakland, CA 94618

PATENT NO: 6,665,725
No. of additional copies

APR 13 2004

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,665,725 B1
DATED : December 16, 2003
INVENTOR(S) : Dietz et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 6,
Line 47, change "NBTBIOS" to -- NETBIOS --.
Line 55, change "Diferent" to -- Different --.

Column 16,
Line 27, change "FIG. 6 FIG 6" to
-- FIG. 6.
FIG6 --.

Column 18,
Line 17, change "updatelookup" to -- update-lookup --.

Column 25,
Line 38, change "server-say" to -- server—say --.

Column 53,
Line 4, change ""Default"" to -- "Default" : --.
Line 45, shift "DISPLAY-HINT" to the right so its beginning lines up with the beginning of "SYNTAX" in line 42 and with the beginning of "LENGTH" in line 43.
Line 46, shift "FLAGS" to the right so its beginning lines up with the beginning of "SYNTAX" in line 42 and with the beginning of "LENGTH" in line 43.

Column 61,
Aprox. line 32, change "rip" to -- r1p --.

Column 71,
Line 9, from the bottom, change "netbios (0x3c00," to -- netbios (0x3c00) --.

Column 73,
Aprox. Line 25, change "tyop" to -- type --.

Column 79,
Line 4 from the bottom, change "SYNTAXINT(8)" to -- SYNTAX INT (8) --.

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,665,725 B1
DATED : December 16, 2003
INVENTOR(S) : Dietz et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 81,
Approx. line 41, change "SYNTAXBITSRING(12)" to -- SYNTAX BITSTRING (12) --.

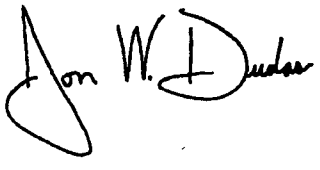
Column 83,
Approx. line 36, change "LOOKUPFILE" to -- LOOKUP FILE --.

Column 93,
Approx. line 45, change "vnd.m-relaudio" to -- 'vnd.rn-realaudio' --.

Column 96,
Line 38, change "In" to -- in --.

Signed and Sealed this

Twenty-ninth Day of June, 2004



JON W. DUDAS
Acting Director of the United States Patent and Trademark Office

Our Ref./Docket No: APPT-001-2 Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

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

Assignee: Hi/fn, Inc.

Patent No: 6,665,725 B1

Issue Date: December, 16, 2003

Application No.: 09/609,179

Filed: June 30, 2000

Title: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE

REQUEST FOR CERTIFICATE OF CORRECTIONS

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

The above patent contains significant errors as indicated on the attached Certificate of Correction form (submitted in duplicate).

__X__ Such errors arose through the fault of the Patent and Trademark Office. It is requested that the certificate be issued at no cost to the applicant.

However, if it is determined that the error(s) arose through the fault of applicant(s), please note that such error is of clerical error or minor nature and occurred in good faith and therefore issuance of the certificate of Correction is respectfully requested. The Commissioner is authorized to charge Deposit Account No. 50-0292 any required fee. A duplicate of this request is attached.

____ Such error arose through the fault of applicant(s). A credit card charge form for the fee is enclosed. Such error is of clerical error or minor nature and occurred in good faith and therefore issuance of the certificate of Correction is respectfully requested.

Such errors specifically:

In col. 6, line 47 change "NBTBIOS" to --NETBIOS--.

Certificate of Mailing under 37 CFR 1.8

I hereby certify that this response is being deposited with the United States Postal Service as first class mail in an envelope addressed to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: Apr. 5, 2004 Signed: Amy Drury

Name: Amy Drury

In col. 6, line 55 change "Diferent" to --Different--.

In col. 16, line 27 change "FIG. 6 FIG 6" to
--FIG 6.
FIG6--.

In col. 18, line 17 change "updatelookup" to --update-lookup--.

In col. 25, line 38 change "server-say" to --server—say--.

In col. 53, line 4 change ""Default"" to --"Default" :--.

In col. 53, line 45 shift "DISPLAY-HINT" to the right so its beginning lines up with the beginning of "SYNTAX" in line 42 and with the beginning of "LENGTH" in line 43.

In col. 53 line 46 shift "FLAGS" to the right so its beginning lines up with the beginning of "SYNTAX" in line 42 and with the beginning of "LENGTH" in line 43.

In col. 61, aprox. line 32 change "rip" to --r1p--.

In col 71, 9th line from the bottom change "netbios (0x3c00," to --netbios (0x3c00)--.

In col. 73, aprox. line 25 change "tyop" to --type--.

In col. 79, 4th line from the bottom change "SYNTAXINT(8)" to --SYNTAX INT (8)--.

In col. 81, approx. line 41 change "SYNTAXBITSRING(12)" to --SYNTAX BITSTRING (12)--.

In col. 83, approx. line 36 change "LOOKUPFILE" to --LOOKUP FILE--.

In col. 93, approx. line 45 change "'vnd.m-relaudio" to --'vnd.rn-realaudio'--.

In col. 96, line 38, change "In" to --in--.

The undersigned requests being contacted at (510) 547-3378 if there are any questions or clarifications, or if there are any problems with issuance of the Certificate of Correction.

Respectfully Submitted,

Apr. 5, 2004
Date

Dov Rosenfeld, Reg. No. 38687
Agent of Record.

Address for correspondence:
Dov Rosenfeld
5507 College Avenue, Suite 2,
Oakland, CA 94618 Tel. (510)547-3378; Fax: (510)291-2985

DOCKET NO.: 10354-001GEN PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In Re Application of:

Russell S. Dietz, Andrew A. Koppenhaver, James F. Torgerson

Confirmation No.: 2668

Application No.: 09/609,179

Group Art Unit: 2155

Patent No.: 6,665,725

Issue Date: December 16, 2003

Filing Date: June 30, 2000

Examiner: Khanh Q. Dinh

For: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE

Commissioner for Patents
Office of Patent Publications
ATTN: Certificate of Correction Branch
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

REQUEST FOR CERTIFICATE OF CORRECTION PURSUANT TO 37 CFR § 1.322 & 37 CFR § 1.323

It is respectfully requested that a Certificate of Correction be issued for the above-identified patent. The patent has **one (1)** error that is the fault of the applicant. Applicant's error occurred in good faith and is of a clerical or typographical nature, or minor character, and is not believed to constitute new matter or require examination.

Enclosed herewith please find a completed Certificate of Correction form.

The fee in the amount of **$100.00** is attached.

Respectfully submitted,

Date: September 4, 2013

/Lawrence A. Aaronson/
Lawrence Aaronson
Reg. No. 38,369

Meunier Carlin & Curfman, LLC
817 W. Peachtree St., NW
Suite 500
Atlanta, GA 30308
phone: (404) 645-7713
fax: (404) 645-7707

PTO/SB/44 (09-07)
Approved for use through 08/31/2013. OMB 0651-0033
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 6,665,725

APPLICATION NO.: 09/609,179

ISSUE DATE : December 16, 2003

INVENTOR(S) : Russell S. Dietz, Andrew A. Koppenhaver, James F. Torgerson

It is certified that an error appears or errors appear in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

IN THE CLAIMS:

Column 1, lines 15 and 16, claim 14, change "searching the packet at the particular protocol" to --searching the packet at the particular protocol level--.

MAILING ADDRESS OF SENDER (Please do not use customer number below):

Meunier Carlin & Curfman, LLC
817 W. Peachtree St., NW, Suite 500
Atlanta, GA 30308

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,665,725 B1
APPLICATION NO. : 09/609179
DATED : December 16, 2003
INVENTOR(S) : Russell S. Dietz, Andrew A. Koppenhaver and James F. Torgerson

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

IN THE CLAIMS:

Column 1, lines 15 and 16, claim 14, change "searching the packet at the particular protocol" to --searching the packet at the particular protocol level--.

Signed and Sealed this
Eighth Day of October, 2013

Teresa Stanek Rea

Teresa Stanek Rea
Deputy Director of the United States Patent and Trademark Office

INVENTEK *Fax*

Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618, USA
Phone: (510)547-3378; Fax: (510)653-7992
dov@inventek.com

OUR REF: APPT-001-2

TO: Mail Stop Issue Fee
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

FAX No.: (703) 746-4000

DATE: September 24, 2003

FROM: Dov Rosenfeld, Reg. No., 38,687

RE: Issue Fee for Application No.: 09/609,179

Number of pages including cover: 5

OFFICIAL COMMUNICATION

ISSUE FEE PAYMENT

Included herewith are:

- A transmittal letter and copy
- Fee(s) Transmittal (form PTOL-85)
- Credit Card charge form for issue fee

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number (703) 746-4000 addressed to Mail Stop Issue Fee, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: September 24, 2003 Signed: _____

Name: Dov Rosenfeld, Reg. No. 38687

Match and Return

Our Ref./Docket No: APPT-001-2 Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Dietz, *et al.*	Group Art Unit: 2756
Application No.: 09/609,179	Examiner:
Filed: June 30, 2000	Notice of Allowance Mailed: July, 1, 2003
Title: PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE	Confirmation No: 2668

SUBMISSION OF ISSUE FEE

Mail Stop ISSUE FEE
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

Transmitted herewith is a completed "Issue Fee Transmittal" Form. Included with the form are:

_X_ A credit card payment form for the issue fee and any advance order of copies;
____ drawing corrections (with separate letter);
____ formal drawings (with separate letter);

_X_ The Commissioner is hereby authorized to charge payment of the any missing fee or credit any overpayment to Deposit Account No. 50-0292
(A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED):

Respectfully Submitted,

24 Sep 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: +1-413-638-1280

Certificate of Facsimile Transmission under 37 CFR 1.8

I hereby certify that this response is being facsimile transmitted to the United States Patent and Trademark Office at telephone number (703) 746-4000 addressed to Mail Stop Issue Fee, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on.

Date: September 24, 2003

Signed:
Name: Dov Rosenfeld, Reg. No. 38687

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: Mail Mail Stop ISSUE FEE
Commissioner for Patents
Alexandria, Virginia 22313-1450
Fax (703)746-4000

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 4 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Legibly mark-up with any corrections or use block 1)

7590 07/01/2003

Dov Rosenfeld
5507 College Avenue
Suite 2
Oakland, CA 94618

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission

I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Box Issue Fee address above, or being facsimile transmitted to the USPTO, on the date indicated below.

Dov Rosenfeld	(Depositor's name)
	(Signature)
24 Sep 03	(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/609,179	06/30/2000	Russell S. Dietz	APPT-001-2	2668

TITLE OF INVENTION: METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK

APPLN. TYPE	SMALL ENTITY	ISSUE FEE	PUBLICATION FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	NO	$1300	$0	$1300	10/01/2003

EXAMINER	ART UNIT	CLASS-SUBCLASS
DINH, KHANH Q	2155	709-230000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).

☐ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.

☐ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. Use of a Customer Number is required.

2. For printing on the patent front page, list (1) the names of up to 3 registered patent attorneys or agents OR, alternatively, (2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.

1 Dov Rosenfeld
2 Inventek
3

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. Inclusion of assignee data is only appropriate when an assignment has been previously submitted to the USPTO or is being submitted under separate cover. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE: Hi/fn, Inc.

(B) RESIDENCE: (CITY and STATE OR COUNTRY): Los Gatos, CA

Please check the appropriate assignee category or categories (will not be printed on the patent) ☐ individual ☒ corporation or other private group entity ☐ government

4a. The following fee(s) are enclosed:

☒ Issue Fee
☐ Publication Fee
☒ Advance Order - # of Copies 10

4b. Payment of Fee(s):

☐ A check in the amount of the fee(s) is enclosed.
☒ Payment by credit card. Form PTO-2038 is attached.
☒ The Commissioner is hereby authorized by charge the required fee(s), or credit any overpayment, to Deposit Account Number 50-0392. (enclose an extra copy of this form).

Commissioner for Patents is requested to apply the Issue Fee and Publication Fee (if any) or to re-apply any previously paid issue fee to the application identified above.

(Authorized Signature) (Date) 24 Sep 03

NOTE: The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office.

This collection of information is required by 37 CFR 1.311. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, Alexandria, Virginia 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, Alexandria, Virginia 22313-1450.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

09/26/2003 TRANG2 00000067 09609179
01 FC:1501 1300.00 DP
02 FC:8001 30.00 DP

TRANSMIT THIS FORM WITH FEE(S)

PATENT APPLICATION FEE DETERMINATION RECORD
Effective December 29, 1999

Application or Docket Number

CLAIMS AS FILED - PART I

	(Column 1) NUMBER FILED	(Column 2) NUMBER EXTRA
FOR		
BASIC FEE		
TOTAL CLAIMS	18 minus 20=	*
INDEPENDENT CLAIMS	1 minus 3 =	*
MULTIPLE DEPENDENT CLAIM PRESENT		

SMALL ENTITY TYPE ☐		OR	OTHER THAN SMALL ENTITY	
RATE	FEE		RATE	FEE
	345.00	OR		690.00
X$ 9=		OR	X$18=	
X39=		OR	X78=	
+130=		OR	+260=	
TOTAL		OR	TOTAL	

* If the difference in column 1 is less than zero, enter "0" in column 2

CLAIMS AS AMENDED - PART II

AMENDMENT A	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				

SMALL ENTITY		OR	OTHER THAN SMALL ENTITY	
RATE	ADDI-TIONAL FEE		RATE	ADDI-TIONAL FEE
X$ 9=		OR	X$18=	
X39=		OR	X78=	
+130=		OR	+260=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT B	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				

RATE	ADDI-TIONAL FEE		RATE	ADDI-TIONAL FEE
X$ 9=		OR	X$18=	
X39=		OR	X78=	
+130=		OR	+260=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT C	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				

RATE	ADDI-TIONAL FEE		RATE	ADDI-TIONAL FEE
X$ 9=		OR	X$18=	
X39=		OR	X78=	
+130=		OR	+260=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."
***If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3."
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

*U.S GPO: 2000-463-433/29044

PATENT APPLICATION FEE DETERMINATION RECORD
Effective October 1, 2000

Application or Docket Number

CLAIMS AS FILED - PART I

	(Column 1)	(Column 2)
TOTAL CLAIMS		
FOR	NUMBER FILED	NUMBER EXTRA
TOTAL CHARGEABLE CLAIMS	18 minus 20=	*
INDEPENDENT CLAIMS	1 minus 3 =	*
MULTIPLE DEPENDENT CLAIM PRESENT		☐

* If the difference in column 1 is less than zero, enter "0" in column 2

SMALL ENTITY TYPE ☐		OR	OTHER THAN SMALL ENTITY	
RATE	FEE		RATE	FEE
BASIC FEE	355.00	OR	BASIC FEE	710.00
X$ 9=		OR	X$18=	
X40=		OR	X80=	
+135=		OR	+270=	
TOTAL		OR	TOTAL	710

CLAIMS AS AMENDED - PART II

AMENDMENT A	(Column 1)		(Column 2)	(Column 3)
	CLAIMS REMAINING AFTER AMENDMENT		HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
Total	* 18	Minus	** 20	=
Independent	* 3	Minus	*** 3	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

SMALL ENTITY		OR	OTHER THAN SMALL ENTITY	
RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X40=		OR	X80=	
+135=		OR	+270=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT B	(Column 1)		(Column 2)	(Column 3)
	CLAIMS REMAINING AFTER AMENDMENT		HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

SMALL ENTITY		OR	OTHER THAN SMALL ENTITY	
RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X40=		OR	X80=	
+135=		OR	+270=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT C	(Column 1)		(Column 2)	(Column 3)
	CLAIMS REMAINING AFTER AMENDMENT		HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

SMALL ENTITY		OR	OTHER THAN SMALL ENTITY	
RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X40=		OR	X80=	
+135=		OR	+270=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."
***If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3."
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=ADJ			
L24	l11 and L23	29	L24
L23	l14 and L22	29	L23
L22	l13 and l20	280	L22
L21	l15 and L20	0	L21
L20	l12 and l13	280	L20
L19	l17 and L18	0	L19
L18	l10 and l11	1453	L18
L17	L16 and l9	2	L17
L16	l6 and L15	62	L16
L15	l4 and l5	141	L15
L14	l2 and l3	5387	L14
L13	pars$4 and compil$3	3782	L13
L12	compress$3 and L11	3331	L12
L11	index$3 same entry	12455	L11
L10	client and server	16935	L10
L9	child$4 and protocol	8849	L9
L8	child$4 (4a) protocol	0	L8
L7	child$4 4a protocol	0	L7
L6	layer$2 and L5	10625	L6
L5	protocol same operat$4	31181	L5
L4	pdl or protocol definition language	1895	L4
L3	monitor$4 same l1	31281	L3
L2	ip or internet protocol	39482	L2
L1	network or internet	263481	L1

END OF SEARCH HISTORY

WEST

Help | Logout | Interrupt

Main Menu | Search Form | Posting Counts | Show S Numbers | Edit S Numbers | Preferences | Cases

Search Results -

Term	Documents
(23 AND 11).USPT.	29
(L11 AND L23).USPT.	29

Database: US Patents Full-Text Database; US Pre-Grant Publication Full-Text Database; JPO Abstracts Database; EPO Abstracts Database; Derwent World Patents Index; IBM Technical Disclosure Bulletins

Search: L24

Refine Search

Recall Text | Clear

Search History

DATE: Thursday, May 29, 2003 Printable Copy Create Case

WEST

Generate Collection | Print

Search Results - Record(s) 1 through 29 of 29 returned.

☑ 1. Document ID: US 6571285 B1

L24: Entry 1 of 29 File: USPT May 27, 2003

US-PAT-NO: 6571285
DOCUMENT-IDENTIFIER: US 6571285 B1

TITLE: Providing an integrated service assurance environment for a network

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw Desc | Image

☑ 2. Document ID: US 6519568 B1

L24: Entry 2 of 29 File: USPT Feb 11, 2003

US-PAT-NO: 6519568
DOCUMENT-IDENTIFIER: US 6519568 B1

TITLE: System and method for electronic data delivery

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

☑ 3. Document ID: US 6516337 B1

L24: Entry 3 of 29 File: USPT Feb 4, 2003

US-PAT-NO: 6516337
DOCUMENT-IDENTIFIER: US 6516337 B1

TITLE: Sending to a central indexing site meta data or signatures from objects on a computer network

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

☑ 4. Document ID: US 6430409 B1

L24: Entry 4 of 29 File: USPT Aug 6, 2002

US-PAT-NO: 6430409
DOCUMENT-IDENTIFIER: US 6430409 B1

TITLE: Method and architecture for an interactive two-way data communication network

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☐ 5. Document ID: US 6421730 B1

L24: Entry 5 of 29 File: USPT Jul 16, 2002

US-PAT-NO: 6421730
DOCUMENT-IDENTIFIER: US 6421730 B1

TITLE: Programmable system for processing a partitioned network infrastructure

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☑ 6. Document ID: US 6405037 B1

L24: Entry 6 of 29 File: USPT Jun 11, 2002

US-PAT-NO: 6405037
DOCUMENT-IDENTIFIER: US 6405037 B1

TITLE: Method and architecture for an interactive two-way data communication network

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☑ 7. Document ID: US 6401117 B1

L24: Entry 7 of 29 File: USPT Jun 4, 2002

US-PAT-NO: 6401117
DOCUMENT-IDENTIFIER: US 6401117 B1

TITLE: Platform permitting execution of multiple network infrastructure applications

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☑ 8. Document ID: US 6393487 B2

L24: Entry 8 of 29 File: USPT May 21, 2002

US-PAT-NO: 6393487
DOCUMENT-IDENTIFIER: US 6393487 B2

TITLE: Passing a communication control block to a local device such that a message

is processed on the device

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☑ 9. Document ID: US 6334153 B1

L24: Entry 9 of 29 File: USPT Dec 25, 2001

US-PAT-NO: 6334153
DOCUMENT-IDENTIFIER: US 6334153 B1

TITLE: Passing a communication control block from host to a local device such that a message is processed on the device

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☐ 10. Document ID: US 6304915 B1

L24: Entry 10 of 29 File: USPT Oct 16, 2001

US-PAT-NO: 6304915
DOCUMENT-IDENTIFIER: US 6304915 B1

TITLE: System, method and article of manufacture for a gateway system architecture with system administration information accessible from a browser

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☑ 11. Document ID: US 6272151 B1

L24: Entry 11 of 29 File: USPT Aug 7, 2001

US-PAT-NO: 6272151
DOCUMENT-IDENTIFIER: US 6272151 B1

TITLE: Scalable multimedia network

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☑ 12. Document ID: US 6247060 B1

L24: Entry 12 of 29 File: USPT Jun 12, 2001

US-PAT-NO: 6247060
DOCUMENT-IDENTIFIER: US 6247060 B1

TITLE: Passing a communication control block from host to a local device such that a message is processed on the device

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☐ 13. Document ID: US 6202060 B1

L24: Entry 13 of 29 File: USPT Mar 13, 2001

US-PAT-NO: 6202060
DOCUMENT-IDENTIFIER: US 6202060 B1

TITLE: Data management system

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☐ 14. Document ID: US 6199076 B1

L24: Entry 14 of 29 File: USPT Mar 6, 2001

US-PAT-NO: 6199076
DOCUMENT-IDENTIFIER: US 6199076 B1

TITLE: Audio program player including a dynamic program selection controller

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☑ 15. Document ID: US 6157955 A

L24: Entry 15 of 29 File: USPT Dec 5, 2000

US-PAT-NO: 6157955
DOCUMENT-IDENTIFIER: US 6157955 A

TITLE: Packet processing system including a policy engine having a classification unit

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☑ 16. Document ID: US 6157935 A

L24: Entry 16 of 29 File: USPT Dec 5, 2000

US-PAT-NO: 6157935
DOCUMENT-IDENTIFIER: US 6157935 A

TITLE: Remote data access and management system

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | DWIC
Draw Desc | Image

☐ 17. Document ID: US 6150962 A

L24: Entry 17 of 29 File: USPT Nov 21, 2000

US-PAT-NO: 6150962
DOCUMENT-IDENTIFIER: US 6150962 A

TITLE: Predictive data entry method for a keyboard

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | DWIC
Draw Desc | Image

☐ 18. Document ID: US 6085233 A

L24: Entry 18 of 29 File: USPT Jul 4, 2000

US-PAT-NO: 6085233
DOCUMENT-IDENTIFIER: US 6085233 A

TITLE: System and method for cellular network computing and communications

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | DWIC
Draw Desc | Image

☑ 19. Document ID: US 5978840 A

L24: Entry 19 of 29 File: USPT Nov 2, 1999

US-PAT-NO: 5978840
DOCUMENT-IDENTIFIER: US 5978840 A

TITLE: System, method and article of manufacture for a payment gateway system architecture for processing encrypted payment transactions utilizing a multichannel, extensible, flexible architecture

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | DWIC
Draw Desc | Image

☐ 20. Document ID: US 5931917 A

L24: Entry 20 of 29 File: USPT Aug 3, 1999

US-PAT-NO: 5931917
DOCUMENT-IDENTIFIER: US 5931917 A

TITLE: System, method and article of manufacture for a gateway system architecture with system administration information accessible from a browser

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☐ 21. Document ID: US 5911485 A

L24: Entry 21 of 29 File: USPT Jun 15, 1999

US-PAT-NO: 5911485
DOCUMENT-IDENTIFIER: US 5911485 A

TITLE: Predictive data entry method for a keypad

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☐ 22. Document ID: US 5864542 A

L24: Entry 22 of 29 File: USPT Jan 26, 1999

US-PAT-NO: 5864542
DOCUMENT-IDENTIFIER: US 5864542 A

TITLE: Scalable multimedia network

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☐ 23. Document ID: US 5809415 A

L24: Entry 23 of 29 File: USPT Sep 15, 1998

US-PAT-NO: 5809415
DOCUMENT-IDENTIFIER: US 5809415 A

TITLE: Method and architecture for an interactive two-way data communication network

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☐ 24. Document ID: US 5799017 A

L24: Entry 24 of 29 File: USPT Aug 25, 1998

US-PAT-NO: 5799017
DOCUMENT-IDENTIFIER: US 5799017 A

TITLE: Scalable multimedia network

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☐ 25. Document ID: US 5740176 A

L24: Entry 25 of 29 File: USPT Apr 14, 1998

US-PAT-NO: 5740176
DOCUMENT-IDENTIFIER: US 5740176 A

TITLE: Scalable multimedia network

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☐ 26. Document ID: US 5732216 A

L24: Entry 26 of 29 File: USPT Mar 24, 1998

US-PAT-NO: 5732216
DOCUMENT-IDENTIFIER: US 5732216 A

TITLE: Audio message exchange system

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☑ 27. Document ID: US 5721827 A

L24: Entry 27 of 29 File: USPT Feb 24, 1998

US-PAT-NO: 5721827
DOCUMENT-IDENTIFIER: US 5721827 A

TITLE: System for electrically distributing personalized information

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☐ 28. Document ID: US 5673265 A

L24: Entry 28 of 29 File: USPT Sep 30, 1997

US-PAT-NO: 5673265
DOCUMENT-IDENTIFIER: US 5673265 A

TITLE: Scalable multimedia network

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC
Draw Desc | Image

☐ 29. Document ID: US 5555244 A

L24: Entry 29 of 29 File: USPT Sep 10, 1996

US-PAT-NO: 5555244
DOCUMENT-IDENTIFIER: US 5555244 A

TITLE: Scalable multimedia network

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw. Desc | Image

Generate Collection | Print

Term	Documents
(23 AND 11).USPT.	29
(L11 AND L23).USPT.	29

Display Format: TI | Change Format

Previous Page Next Page

PALM INTRANET

Day : Monday
Date: 6/16/2003
Time: 11:49:19

Application Number Information

Application Number: **09/609179**
Assignments
Filing Date: **06/30/2000**
Effective Date: **06/30/2000**
Application Received: **07/03/2000**
Patent Number:
Issue Date: **00/00/0000**
Date of Abandonment: **00/00/0000**
Attorney Docket Number: **APPT-001-2**
Status: **41 /NON FINAL ACTION MAILED**
Confirmation Number: **2668**
Examiner Number: **74865 / DINH, KHANH**
Group Art Unit: **2155**
Class/Subclass: **709/236.000**
Lost Case: **NO**
Interference Number:
Unmatched Petition: **NO**
L&R Code: Secrecy Code:**1**
Third Level Review: **NO**
Oral Hearing: **NO**
Waiting for Response Desc.
Mail Non Final
Secrecy Order: **NO**
Status Date: **06/04/2003**
Title of Invention: **METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK**

Bar Code	PALM Location	Location Date	Charge to Loc	Charge to Name	Employee Name	Location
09609179	21C1	06/05/2003	No Charge to Location	No Charge to Name	SAID,ABUDULKADAR	PK2/06/C 09

Appln Info | Contents | Petition Info | Atty/Agent Info | Continuity Data | Foreign Data | Inve

Search Another: Application# [ ] Search **or Patent#** [ ] Search

PCT / [ ] / [ ] Search **or PG PUBS #** [ ] Search

Attorney Docket # [ ] Search

Bar Code # [ ] Search

To go back use Back button on your browser toolbar.

Back to PALM | ASSIGNMENT | OASIS | Home page

PALM INTRANET

Day : Monday
Date: 6/16/2003
Time: 11:49:19

Application Number Information

Application Number: **09/609179**
Assignments
Filing Date: **06/30/2000**
Effective Date: **06/30/2000**
Application Received: **07/03/2000**
Patent Number:
Issue Date: **00/00/0000**
Date of Abandonment: **00/00/0000**
Attorney Docket Number: **APPT-001-2**
Status: **41 /NON FINAL ACTION MAILED**
Confirmation Number: **2668**
Examiner Number: **74865 / DINH, KHANH**
Group Art Unit: **2155**
Class/Subclass: **709/236.000**
Lost Case: **NO**
Interference Number:
Unmatched Petition: **NO**
L&R Code: Secrecy Code:**1**
Third Level Review: **NO**
Oral Hearing: **NO**
Waiting for Response Desc.
Mail Non Final
Secrecy Order: **NO**
Status Date: **06/04/2003**
Title of Invention: **METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK**

Bar Code	PALM Location	Location Date	Charge to Loc	Charge to Name	Employee Name	Location
09609179	21C1	06/05/2003	No Charge to Location	No Charge to Name	SAID,ABUDULKADAR	PK2/06/C 09

Appln Info | Contents | Petition Info | Atty/Agent Info | Continuity Data | Foreign Data | Inve

Search Another: Application# [ ] Search **or Patent#** [ ] Search

PCT / [ ] / [ ] Search **or PG PUBS #** [ ] Search

Attorney Docket # [ ] Search

Bar Code # [ ] Search

To go back use Back button on your browser toolbar.

Back to PALM | ASSIGNMENT | OASIS | Home page

6/16/03 11:49 AM

This Form is for INTERNAL PTO USE ONLY
It does NOT get mailed to the applicant.

NOTICE OF FILING / CLAIM FEE(S) DUE (CALCULATION SHEET)

APPLICATION NUMBER: 9/609,179

Total Fee Calculation

	Fee Code Sm./Lg	Total # Claims		Number Extra	X	Fee Sm Entity	Fee Lg Entity	=	Total
Basic Filing Fee	201/101						690	=	690
Total Claims >20	203/103	18	-20 =		X			=	
Independent Claims >3	202/102	1	-3 =		X			=	
Mult. Dep Claim Present	204/104							=	
Surcharge	205/105						130	=	130
English Translation	139								
TOTAL FEE CALCULATION									820

Fees due upon filing the application

Total Filing Fees Due = $ 820.00

Less Filing Fees Submitted - $ ✓

BALANCE DUE = $ 820.00

J. Artis
Office of Initial Patent Examination

Figure 7

FORM OIPE-RAM-01 (Rev. 12/97)

US006665725B1

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

(10) **Patent No.: US 6,665,725 B1**
(45) **Date of Patent: Dec. 16, 2003**

(54) **PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE**

(75) Inventors: **Russell S. Dietz**, San Jose, CA (US); **Andrew A. Koppenhaver**, Littleton, CO (US); **James F. Torgerson**, Andover, MN (US)

(73) Assignee: **Hl/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 537 days.

(21) Appl. No.: **09/609,179**

(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.** **709/230**; 709/246; 709/228; 370/389

(58) **Field of Search** 709/203, 206, 709/216, 217, 222, 246, 225, 228, 230, 232; 703/26; 370/489, 13, 17

(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	709/203
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
5,414,650	A	5/1995	Hekhuis	364/715.02
5,414,704	A	5/1995	Spinney	370/60

(List continued on next page.)

OTHER PUBLICATIONS

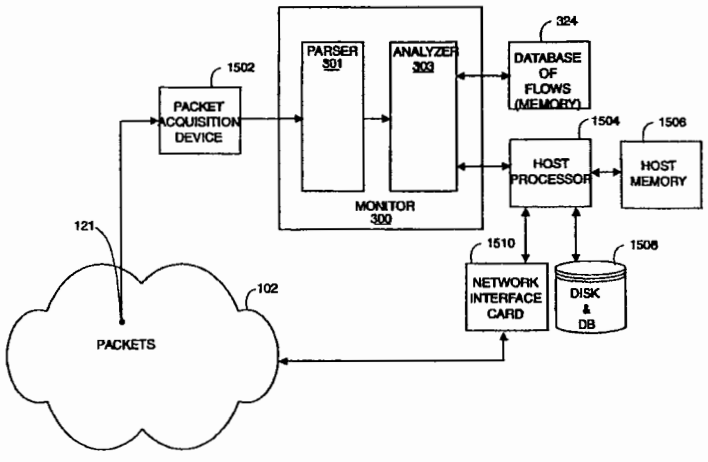
"Technical Note: the Narus System," Downloaded Apr. 29, 1999 from www.narus.com, Narus Corporation, Redwood City California.

Primary Examiner—Hosain T. Alam
Assistant Examiner—Khanh Quang Dinh
(74) *Attorney, Agent, or Firm*—Dov Rosenfeld; Inventek

(57) **ABSTRACT**

A method of performing protocol specific operations on a packet passing through a connection point on a computer network. The packet contents conform to protocols of a layered model wherein the protocol at a at a particular layer level may include one or a set of child protocols defined for that level. The method includes receiving the packet and receiving a set of protocol descriptions for protocols may be used in the packet. A protocol description for a particular protocol at a particular layer level includes any child protocols of the particular protocol, and for any child protocol, where in the packet information related to the particular child protocol may be found. A protocol description also includes any protocol specific operations to be performed on the packet for the particular protocol at the particular layer level. The method includes performing the protocol specific operations on the packet specified by the set of protocol descriptions based on the base protocol of the packet and the children of the protocols used in the packet. A particular embodiment includes providing the protocol descriptions in a high-level protocol description language, and compiling to the descriptions into a data structure. The compiling may further include compressing the data structure into a compressed data structure. The protocol specific operations may include parsing and extraction operations to extract identifying information. The protocol specific operations may also include state processing operations defined for a particular state of a conversational flow of the packet.

17 Claims, 20 Drawing Sheets



U.S. PATENT DOCUMENTS

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. 709/206
5,500,855	A		3/1996	Hershey et al. 370/17
5,511,215	A		4/1996	Terasaka et al. 709/246
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. 709/216
5,606,668	A		2/1997	Shwed 709/216
5,608,662	A		3/1997	Large et al. 364/724.01
5,634,009	A		5/1997	Iddon et al. 709/206
5,651,002	A		7/1997	Van Seters et al. 370/392
5,680,585	A	*	10/1997	Bruell 703/26
5,684,954	A		11/1997	Kaiserswerth et al. 709/203
5,703,877	A		12/1997	Nuber et al. 370/395
5,721,827	A	*	2/1998	Logan et al. 709/217
5,732,213	A		3/1998	Gessel et al. 709/216
5,740,355	A		4/1998	Watanabe et al. 395/183.21
5,761,424	A		6/1998	Adams et al. 709/232
5,764,638	A		6/1998	Ketchum 370/401
5,781,735	A		7/1998	Southard 709/238
5,784,298	A		7/1998	Hershey et al. 364/557
5,787,253	A		7/1998	McCreery et al. 709/227
5,805,808	A		9/1998	Hansani et al. 709/203
5,812,529	A		9/1998	Czarnik et al. 370/245
5,819,028	A		10/1998	Manghirmalani et al. ... 709/203
5,825,774	A		10/1998	Ready et al. 370/401
5,826,017	A		10/1998	Holzmann 709/206
5,835,726	A		11/1998	Shwed et al. 709/228
5,838,919	A		11/1998	Schwaller et al. 709/208
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. 709/232
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 709/238
5,917,821	A		6/1999	Gobuyan et al. 370/392
6,014,380	A		1/2000	Hendel et al. 370/392
6,272,151	B1	*	8/2001	Gupta et al. 370/489
6,430,409	B1	*	8/2002	Rossmann 455/422.1
6,516,337	B1	*	2/2003	Tripp et al. 709/202
6,519,568	B1	*	2/2003	Harvey et al. 705/1

* cited by examiner

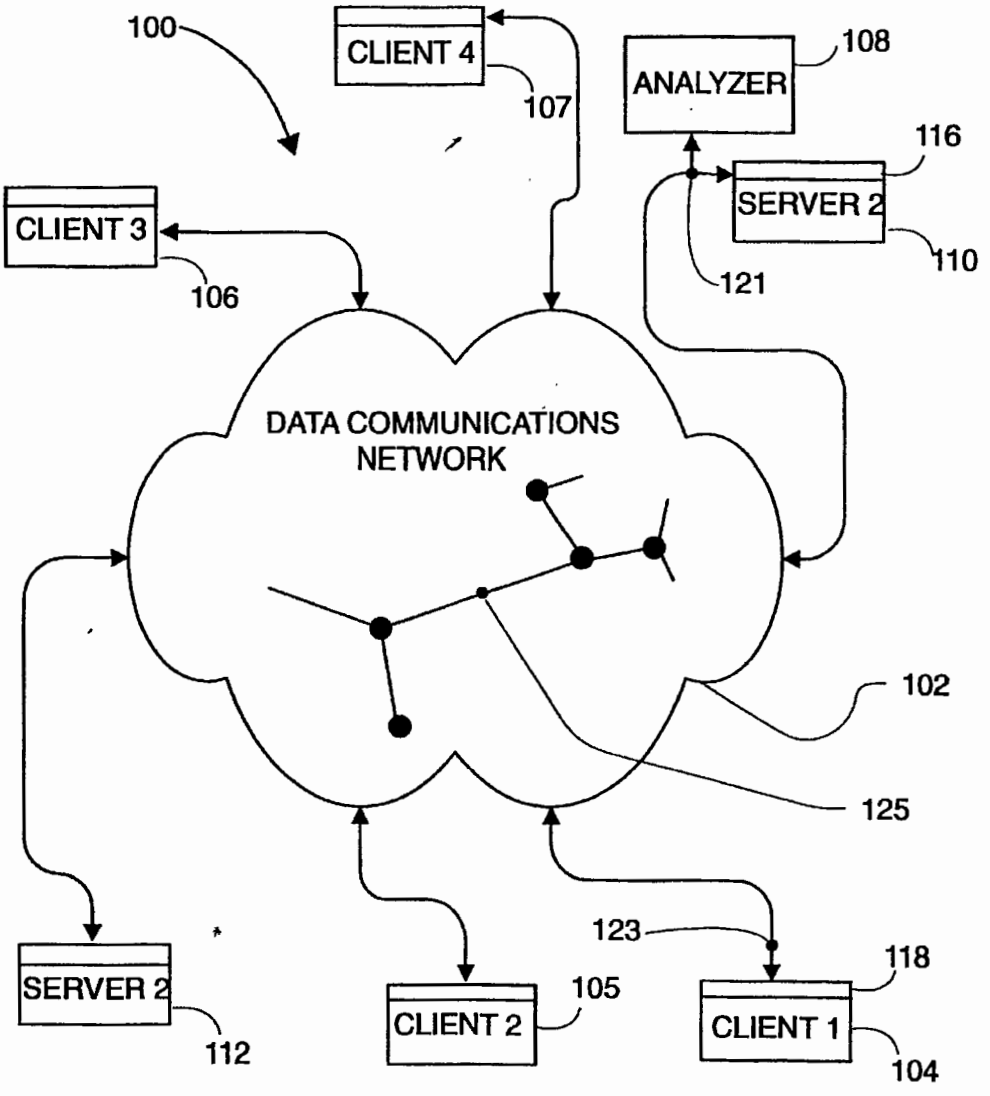

100
CLIENT 4
107
ANALYZER
108
116
SERVER 2
110
121
CLIENT 3
106
DATA COMMUNICATIONS NETWORK
102
125
123
118
SERVER 2
112
CLIENT 2
105
CLIENT 1
104

FIG. 1

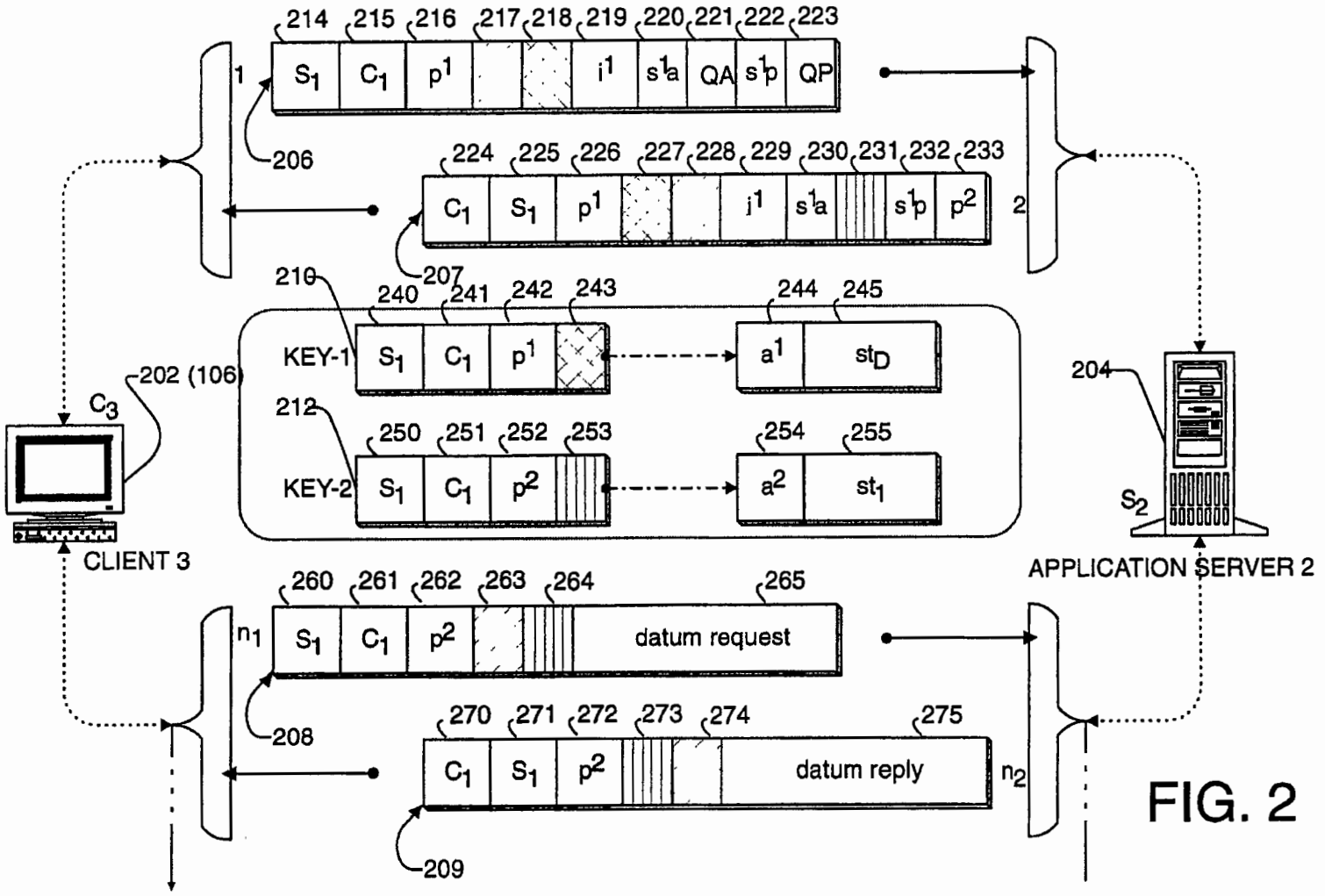

214 215 216 217 218 219 220 221 222 223
S1 C1 p1 i1 s1a QA s1p QP
1
206
224 225 226 227 228 229 230 231 232 233
C1 S1 p1 i1 s1a s1p p2
2
207
210
240 241 242 243 244 245
KEY-1 S1 C1 p1 a1 stD
212
250 251 252 253 254 255
KEY-2 S1 C1 p2 a2 st1
202 (106)
C3
CLIENT 3
204
S2
APPLICATION SERVER 2
260 261 262 263 264 265
n1
S1 C1 p2 datum request
208
270 271 272 273 274 275
C1 S1 p2 datum reply
n2
209

FIG. 2

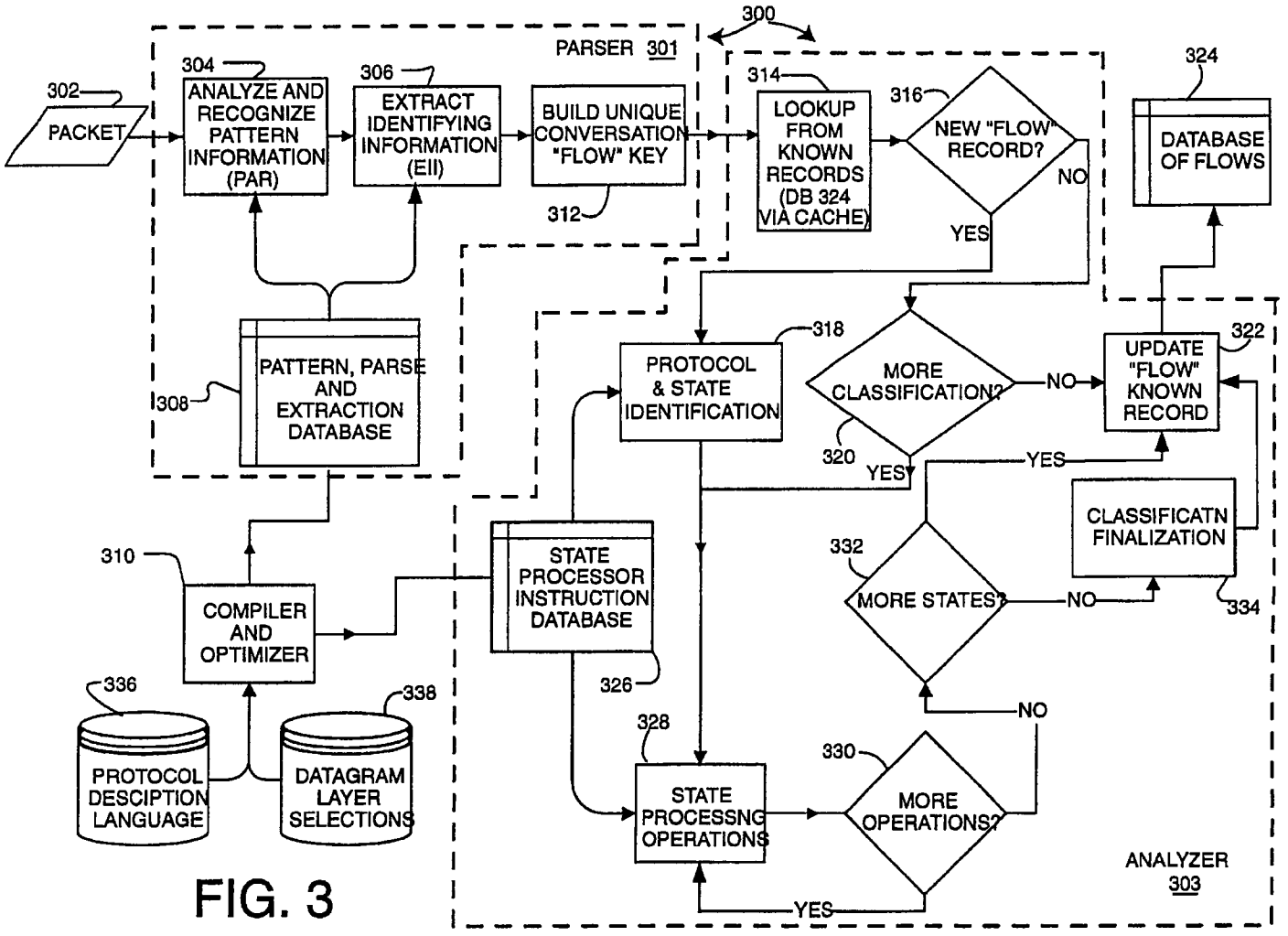

300
PARSER 301
302
PACKET
304
ANALYZE AND RECOGNIZE PATTERN INFORMATION (PAR)
306
EXTRACT IDENTIFYING INFORMATION (EII)
BUILD UNIQUE CONVERSATION "FLOW" KEY
312
314
LOOKUP FROM KNOWN RECORDS (DB 324 VIA CACHE)
316
NEW "FLOW" RECORD?
NO
YES
324
DATABASE OF FLOWS
308
PATTERN, PARSE AND EXTRACTION DATABASE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

FIG. 3

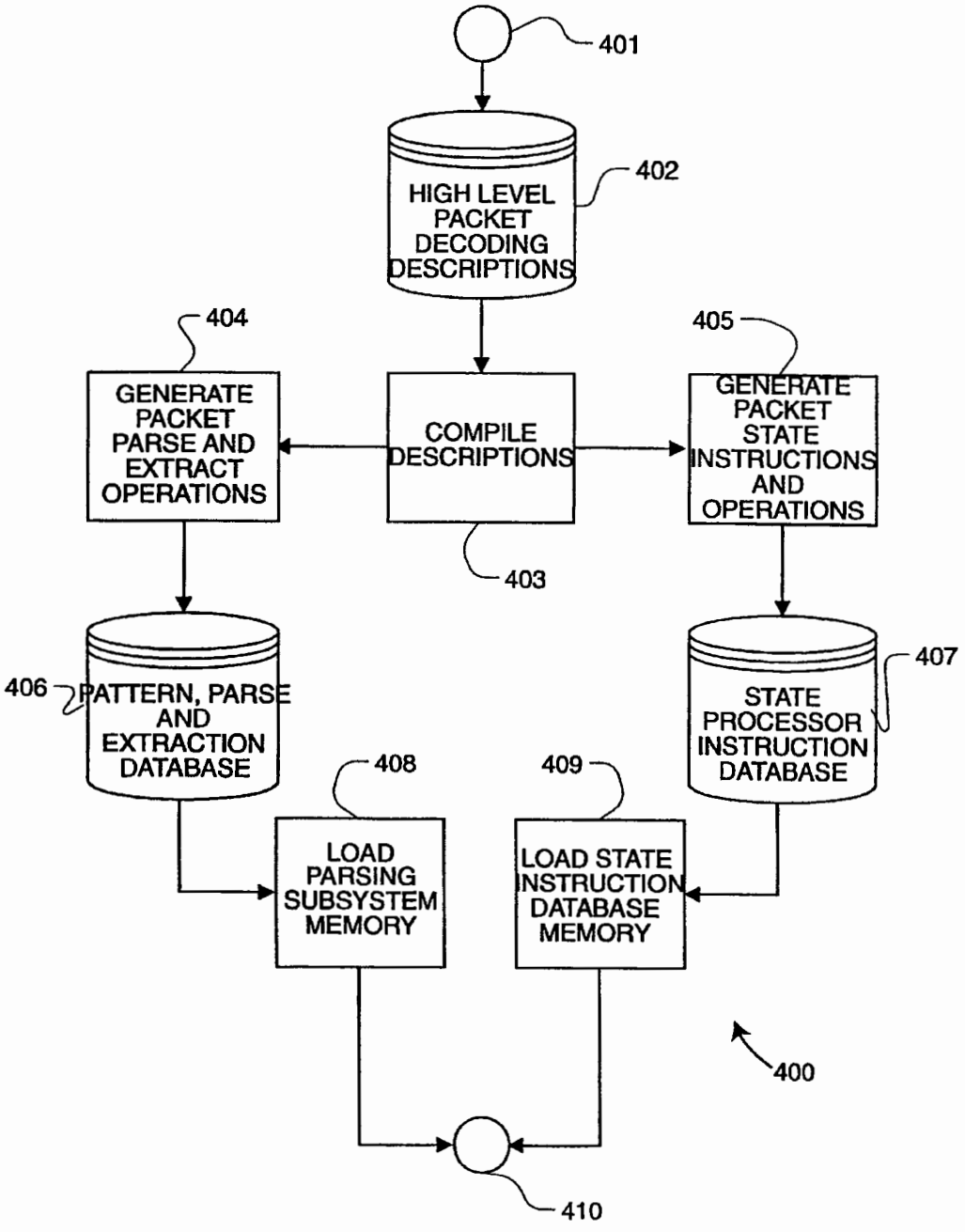

401
HIGH LEVEL PACKET DECODING DESCRIPTIONS
402
404
GENERATE PACKET PARSE AND EXTRACT OPERATIONS
COMPILE DESCRIPTIONS
403
405
GENERATE PACKET STATE INSTRUCTIONS AND OPERATIONS
406
PATTERN, PARSE AND EXTRACTION DATABASE
407
STATE PROCESSOR INSTRUCTION DATABASE
408
LOAD PARSING SUBSYSTEM MEMORY
409
LOAD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

FIG. 4

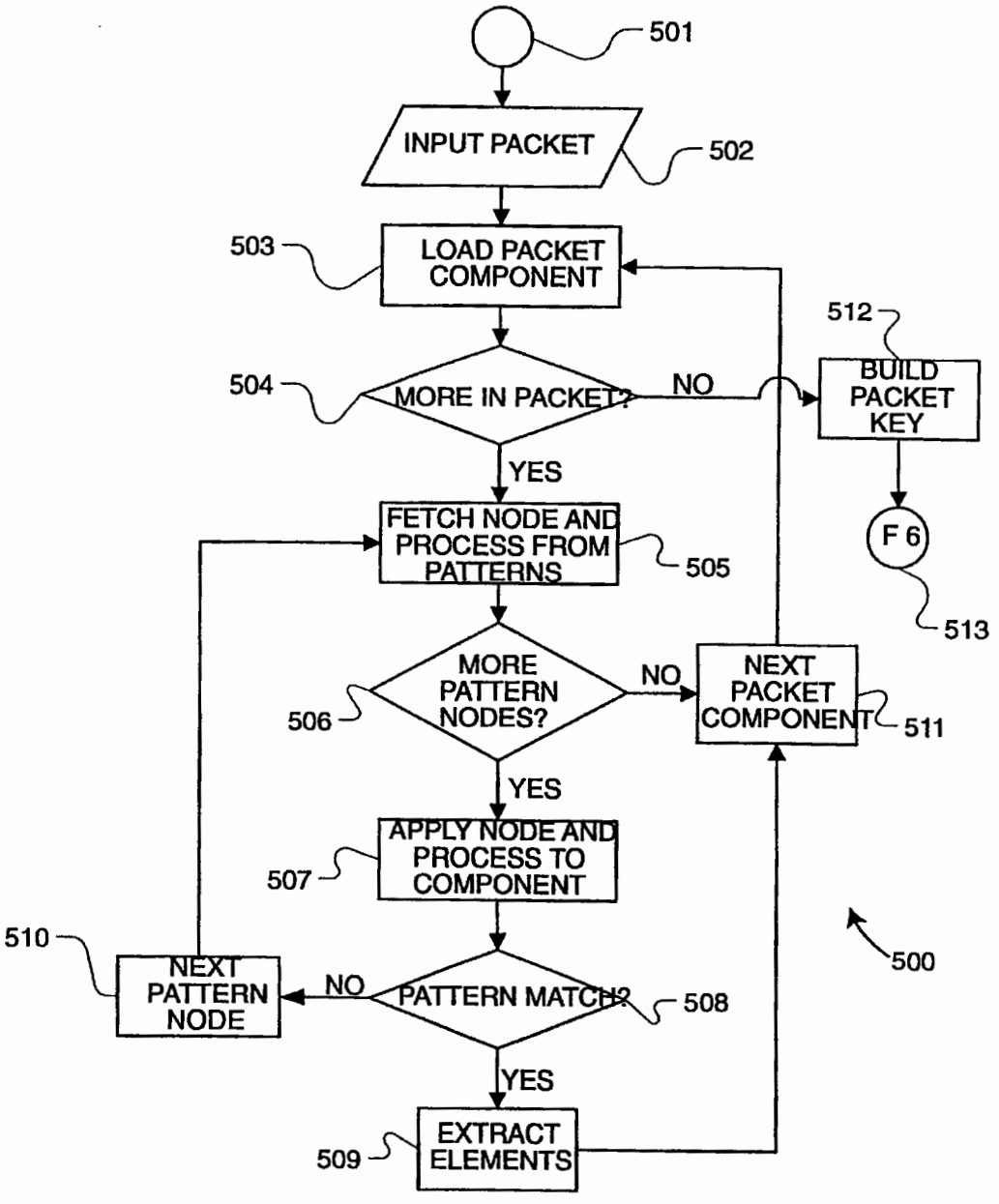

501
INPUT PACKET
502
503
LOAD PACKET COMPONENT
512
504
MORE IN PACKET?
NO
BUILD PACKET KEY
YES
F 6
513
FETCH NODE AND PROCESS FROM PATTERNS
505
MORE PATTERN NODES?
506
NO
NEXT PACKET COMPONENT
511
YES
APPLY NODE AND PROCESS TO COMPONENT
507
500
510
NEXT PATTERN NODE
NO
PATTERN MATCH?
508
YES
EXTRACT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

FIG. 5

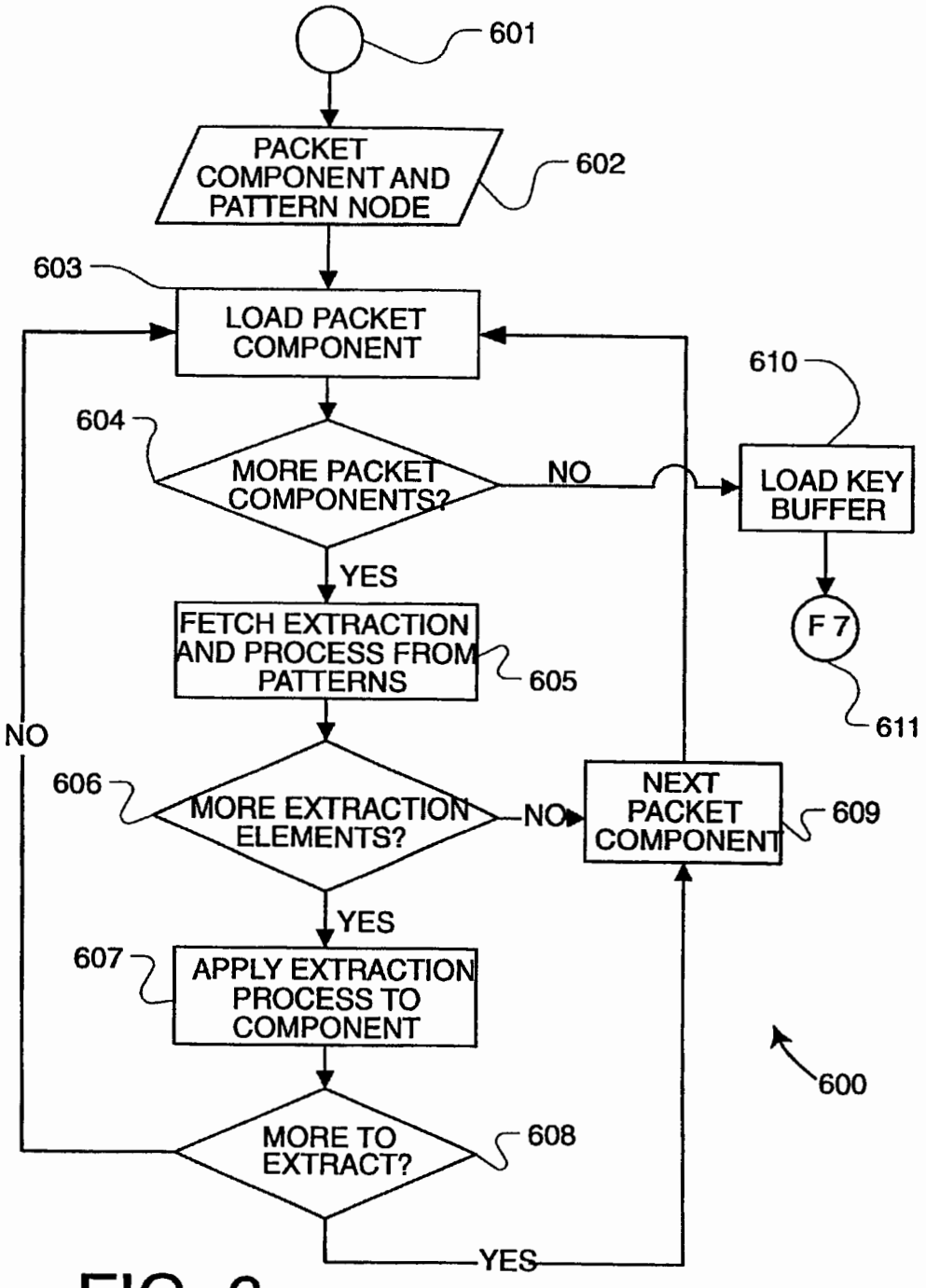

601
PACKET COMPONENT AND PATTERN NODE
602
603
LOAD PACKET COMPONENT
610
604
MORE PACKET COMPONENTS?
NO
LOAD KEY BUFFER
YES
F 7
FETCH EXTRACTION AND PROCESS FROM PATTERNS
605
611
NO
606
MORE EXTRACTION ELEMENTS?
NO
NEXT PACKET COMPONENT
609
YES
607
APPLY EXTRACTION PROCESS TO COMPONENT
600
MORE TO EXTRACT?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

FIG. 6

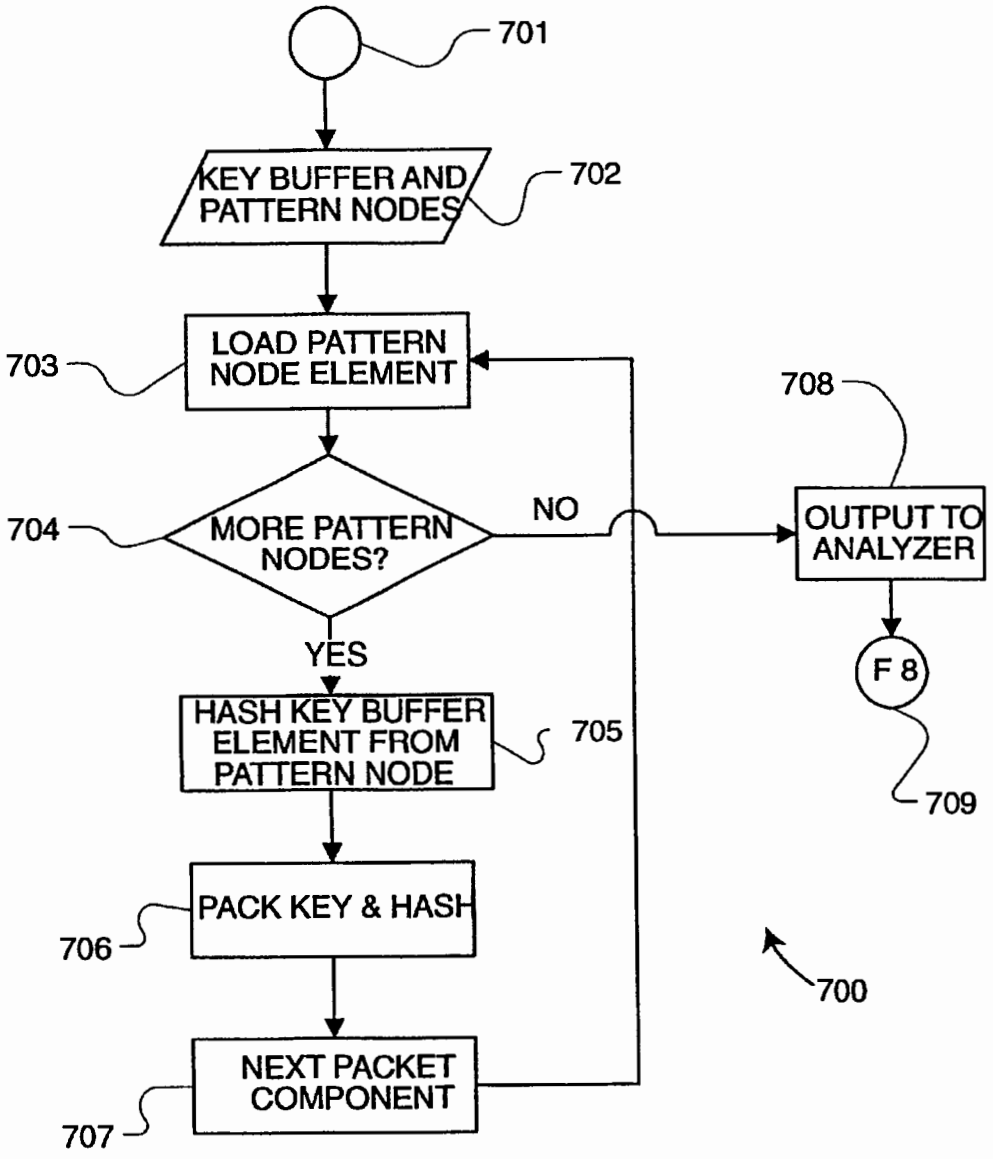

701
KEY BUFFER AND PATTERN NODES
702
LOAD PATTERN NODE ELEMENT
703
MORE PATTERN NODES?
704
NO
YES
OUTPUT TO ANALYZER
708
F 8
709
HASH KEY BUFFER ELEMENT FROM PATTERN NODE
705
PACK KEY & HASH
706
NEXT PACKET COMPONENT
707
700

FIG. 7

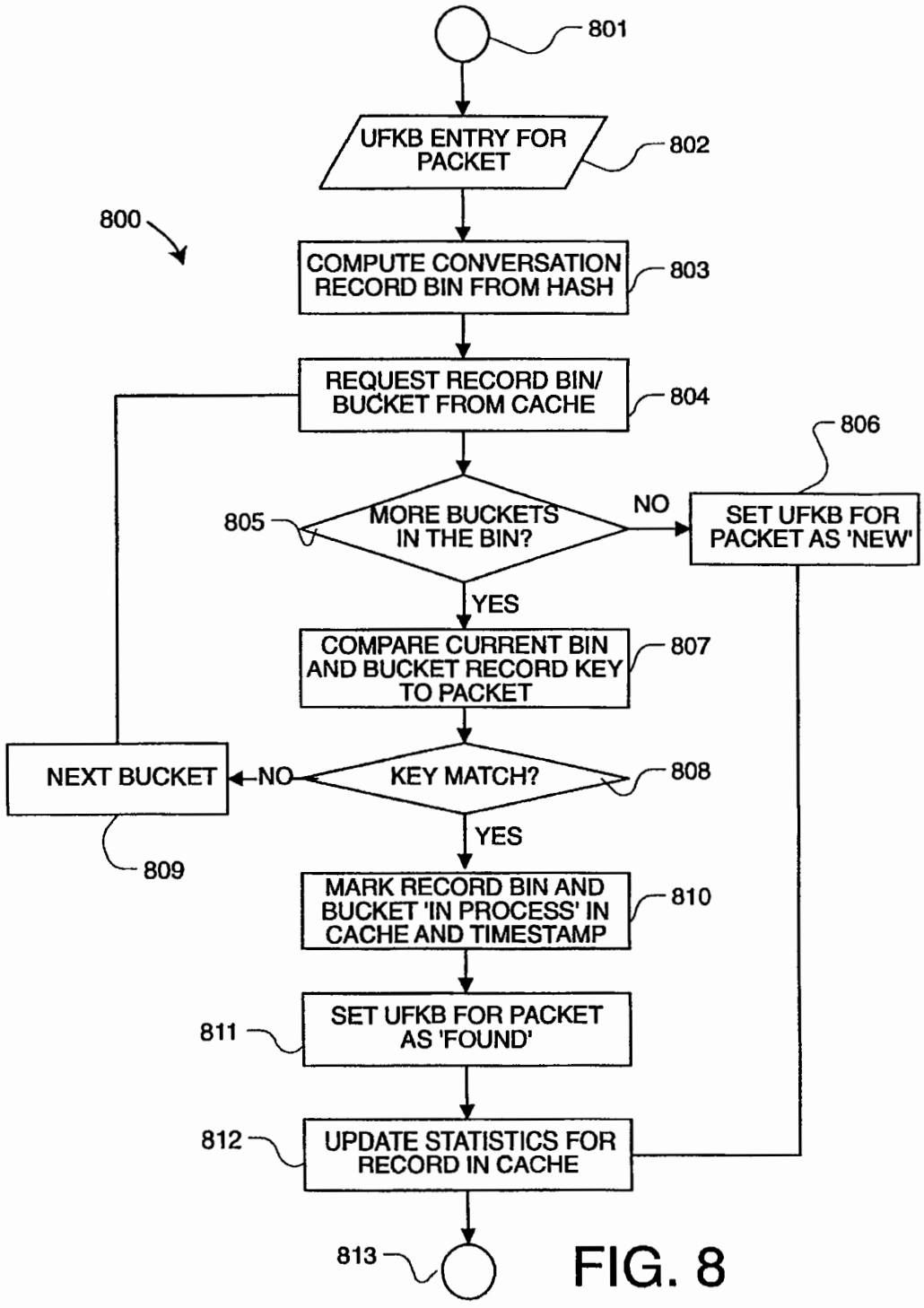

801
UFKB ENTRY FOR PACKET
802
800
COMPUTE CONVERSATION RECORD BIN FROM HASH
803
REQUEST RECORD BIN/ BUCKET FROM CACHE
804
806
805
MORE BUCKETS IN THE BIN?
NO
SET UFKB FOR PACKET AS 'NEW'
YES
COMPARE CURRENT BIN AND BUCKET RECORD KEY TO PACKET
807
NEXT BUCKET
NO
KEY MATCH?
808
YES
809
MARK RECORD BIN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

FIG. 8

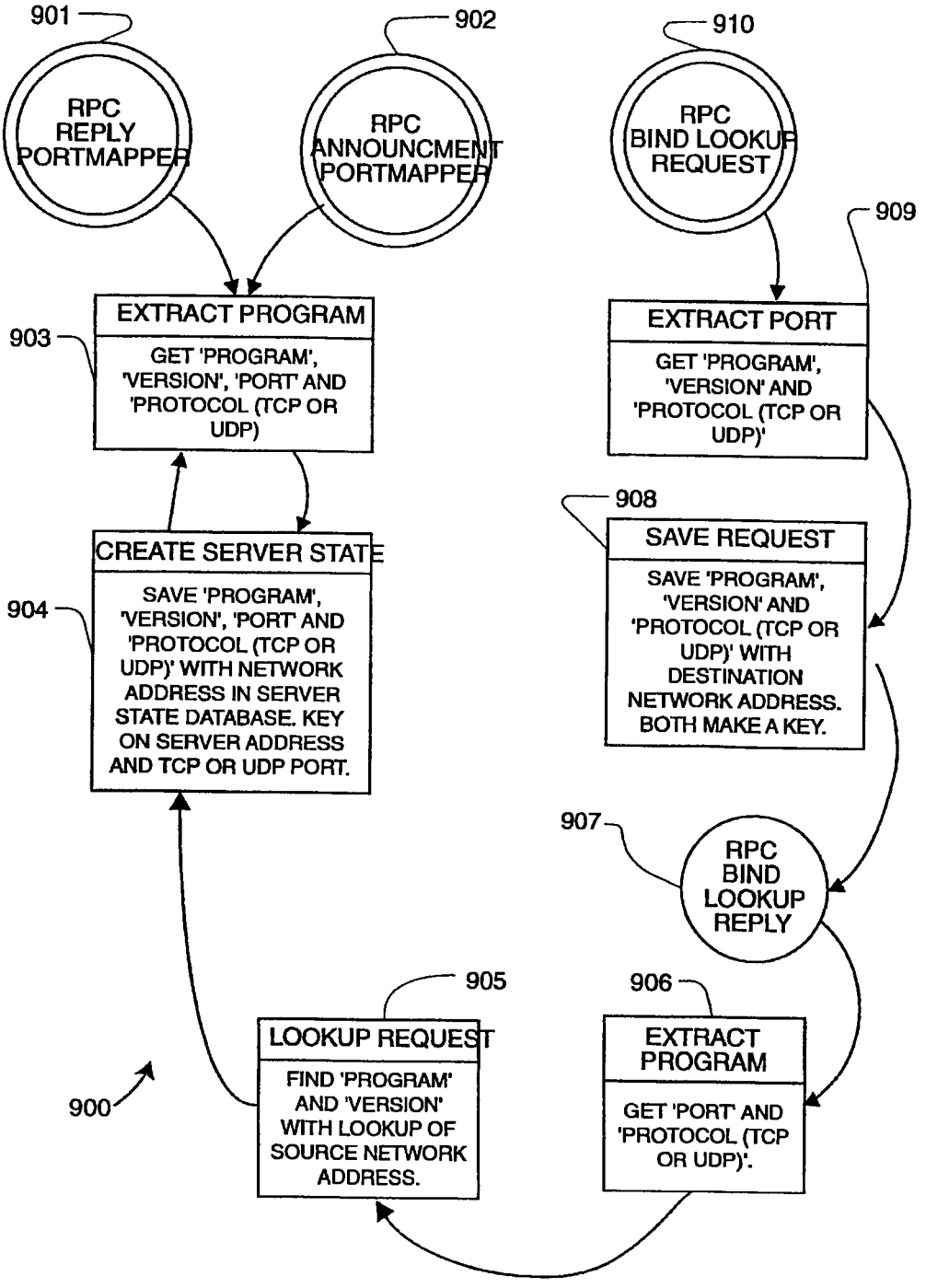

901
RPC REPLY PORTMAPPER
902
RPC ANNOUNCMENT PORTMAPPER
910
RPC BIND LOOKUP REQUEST
909
EXTRACT PROGRAM
903
GET 'PROGRAM', 'VERSION', 'PORT' AND 'PROTOCOL (TCP OR UDP)
EXTRACT PORT
GET 'PROGRAM', 'VERSION' AND 'PROTOCOL (TCP OR UDP)'
908
SAVE REQUEST
SAVE 'PROGRAM', 'VERSION' AND 'PROTOCOL (TCP OR UDP)'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

FIG. 9

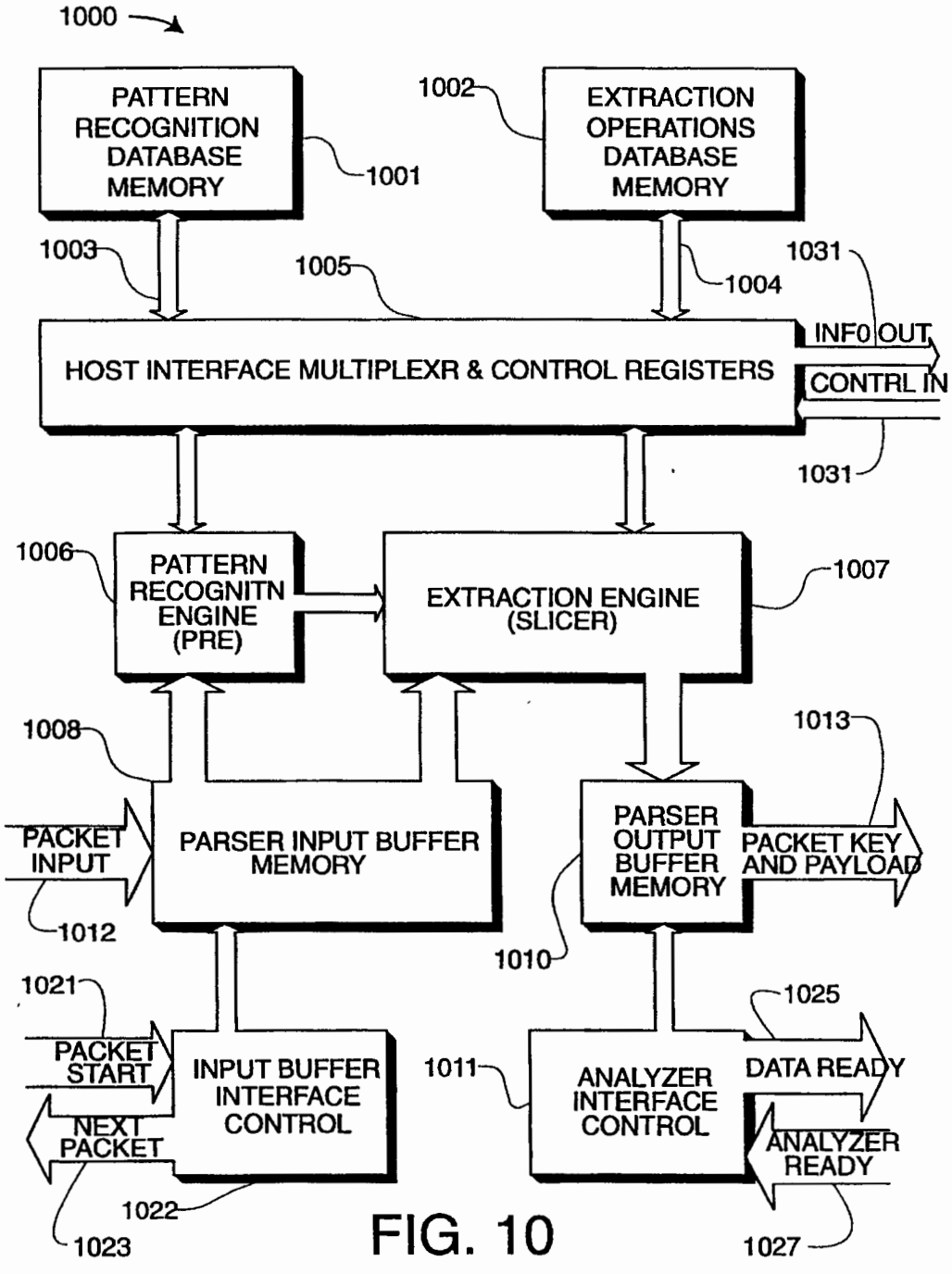

1000
PATTERN RECOGNITION DATABASE MEMORY
1001
1002
EXTRACTION OPERATIONS DATABASE MEMORY
1003
1005
1004
1031
HOST INTERFACE MULTIPLEXR & CONTROL REGISTERS
INFO OUT
CONTRL IN
1031
1006
PATTERN RECOGNITN ENGINE (PRE)
EXTRACTION ENGINE (SLICER)
1007
1008
1013
PACKET INPUT
PARSER INPUT BUFFER MEMORY
PARSER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

FIG. 10

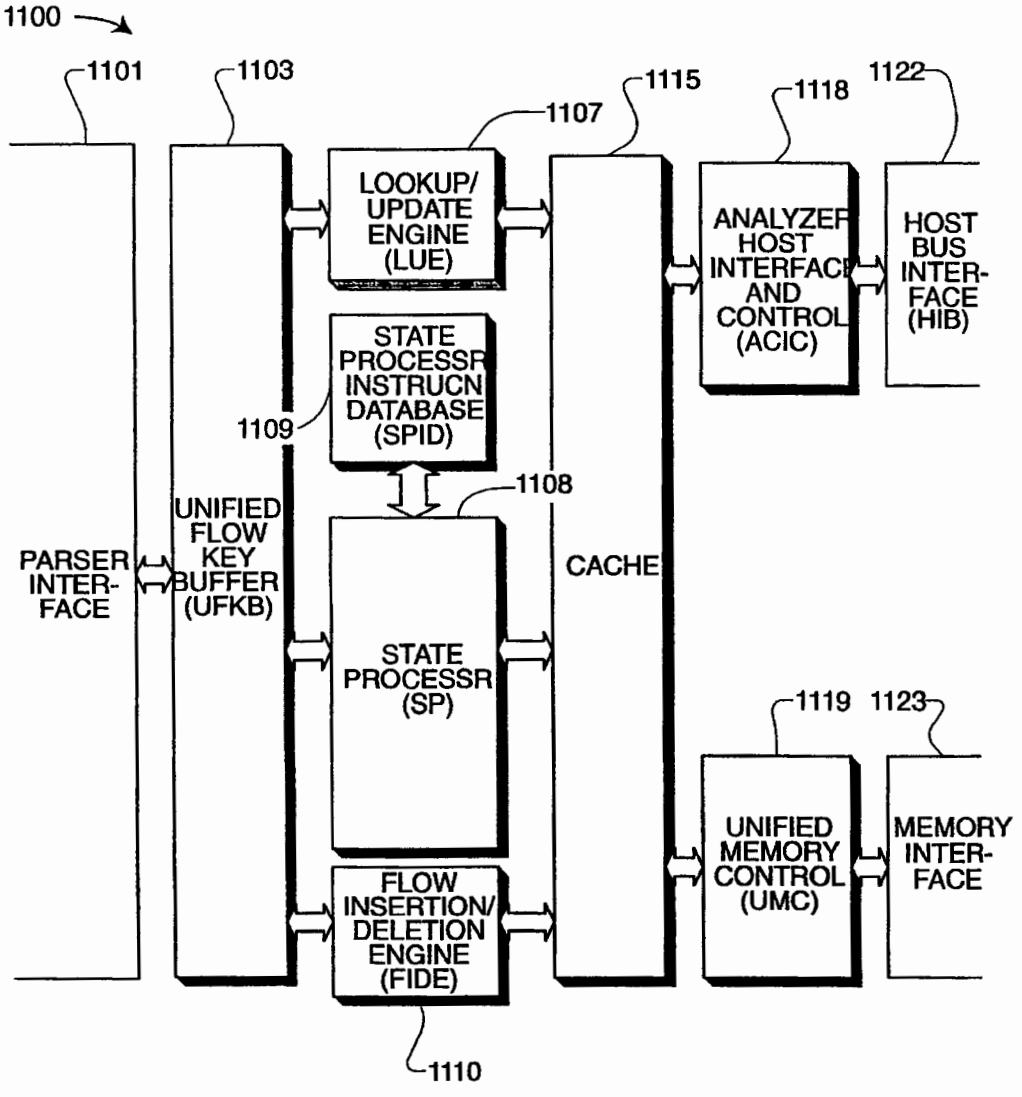

1100
1101
1103
1107
1115
1118
1122
LOOKUP/ UPDATE ENGINE (LUE)
ANALYZER HOST INTERFACE AND CONTROL (ACIC)
HOST BUS INTER-FACE (HIB)
STATE PROCESSR INSTRUCN DATABASE (SPID)
1109
1108
UNIFIED FLOW KEY BUFFER (UFKB)
PARSER INTER-FACE
CACHE
STATE PROCESSR (SP)
1119
1123
UNIFIED MEMORY CONTROL (UMC)
MEMORY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

FIG. 11

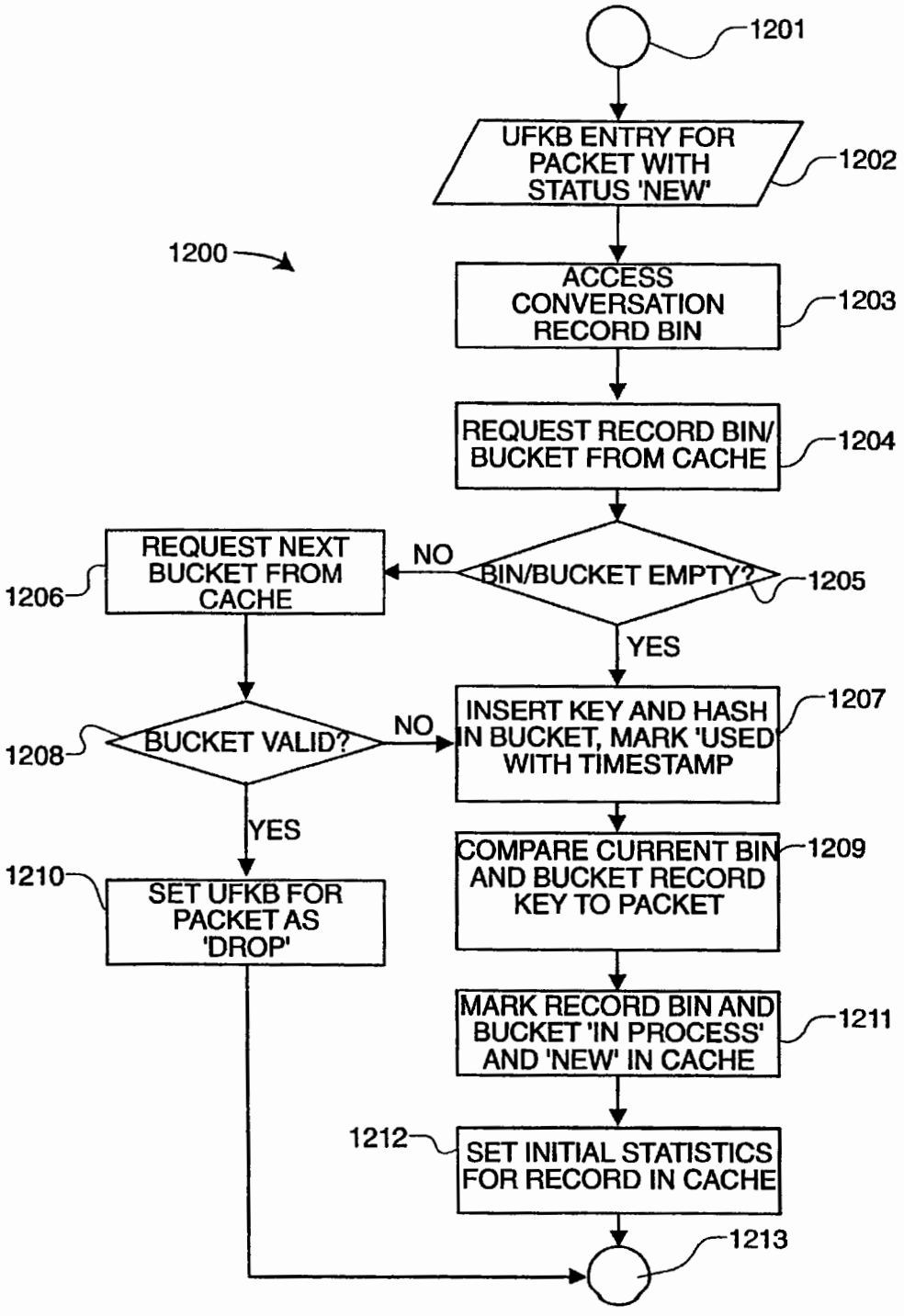

1200
1201
UFKB ENTRY FOR PACKET WITH STATUS 'NEW'
1202
ACCESS CONVERSATION RECORD BIN
1203
REQUEST RECORD BIN/ BUCKET FROM CACHE
1204
BIN/BUCKET EMPTY?
1205
NO
REQUEST NEXT BUCKET FROM CACHE
1206
YES
INSERT KEY AND HASH IN BUCKET, MARK 'USED WITH TIMESTAMP
1207
BUCKET VALID?
1208
NO
YES
COMPARE CURRENT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

FIG. 12

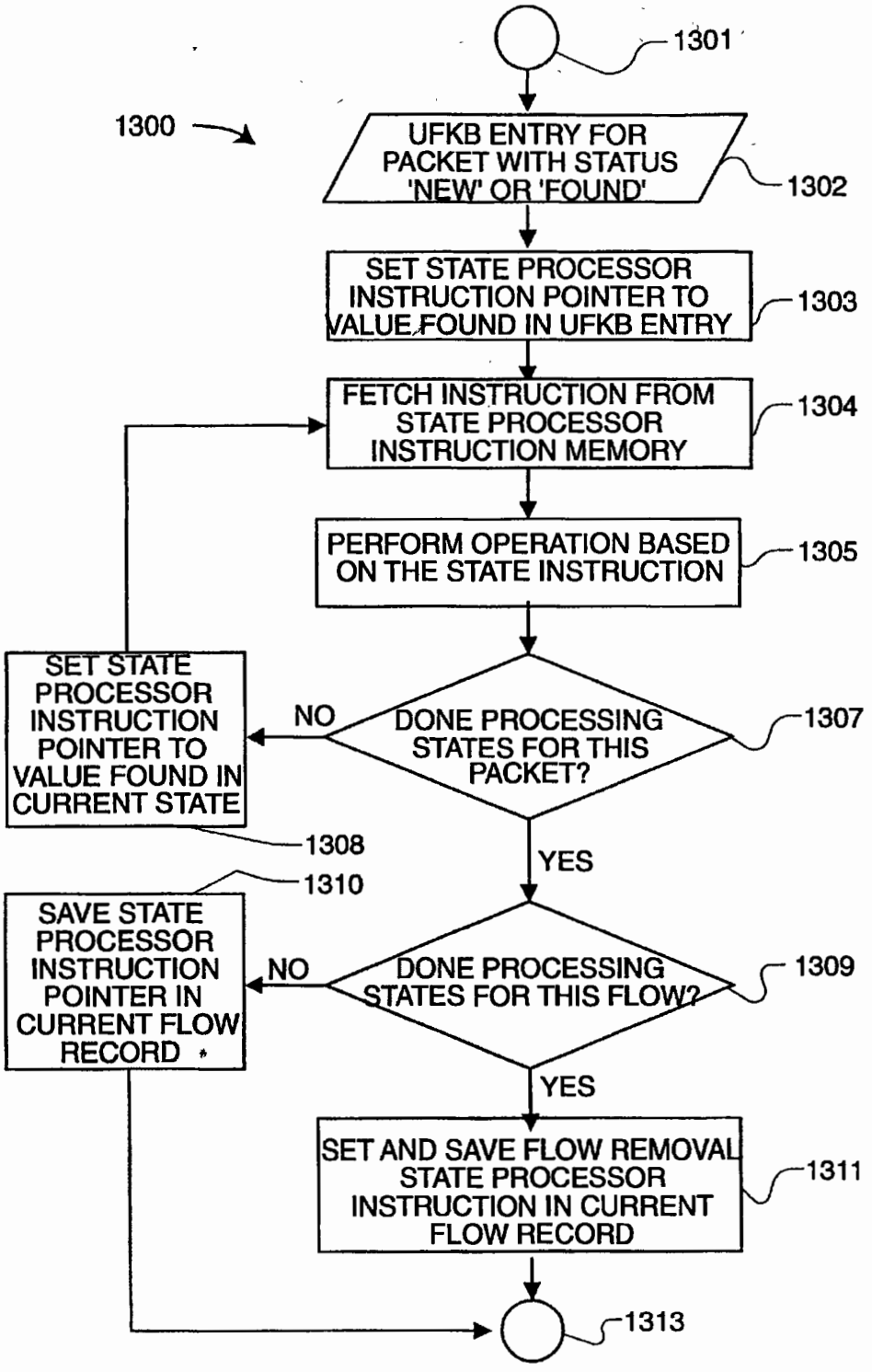

1301
1300
UFKB ENTRY FOR PACKET WITH STATUS 'NEW' OR 'FOUND'
1302
SET STATE PROCESSOR INSTRUCTION POINTER TO VALUE FOUND IN UFKB ENTRY
1303
FETCH INSTRUCTION FROM STATE PROCESSOR INSTRUCTION MEMORY
1304
PERFORM OPERATION BASED ON THE STATE INSTRUCTION
1305
DONE PROCESSING STATES FOR THIS PACKET?
1307
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

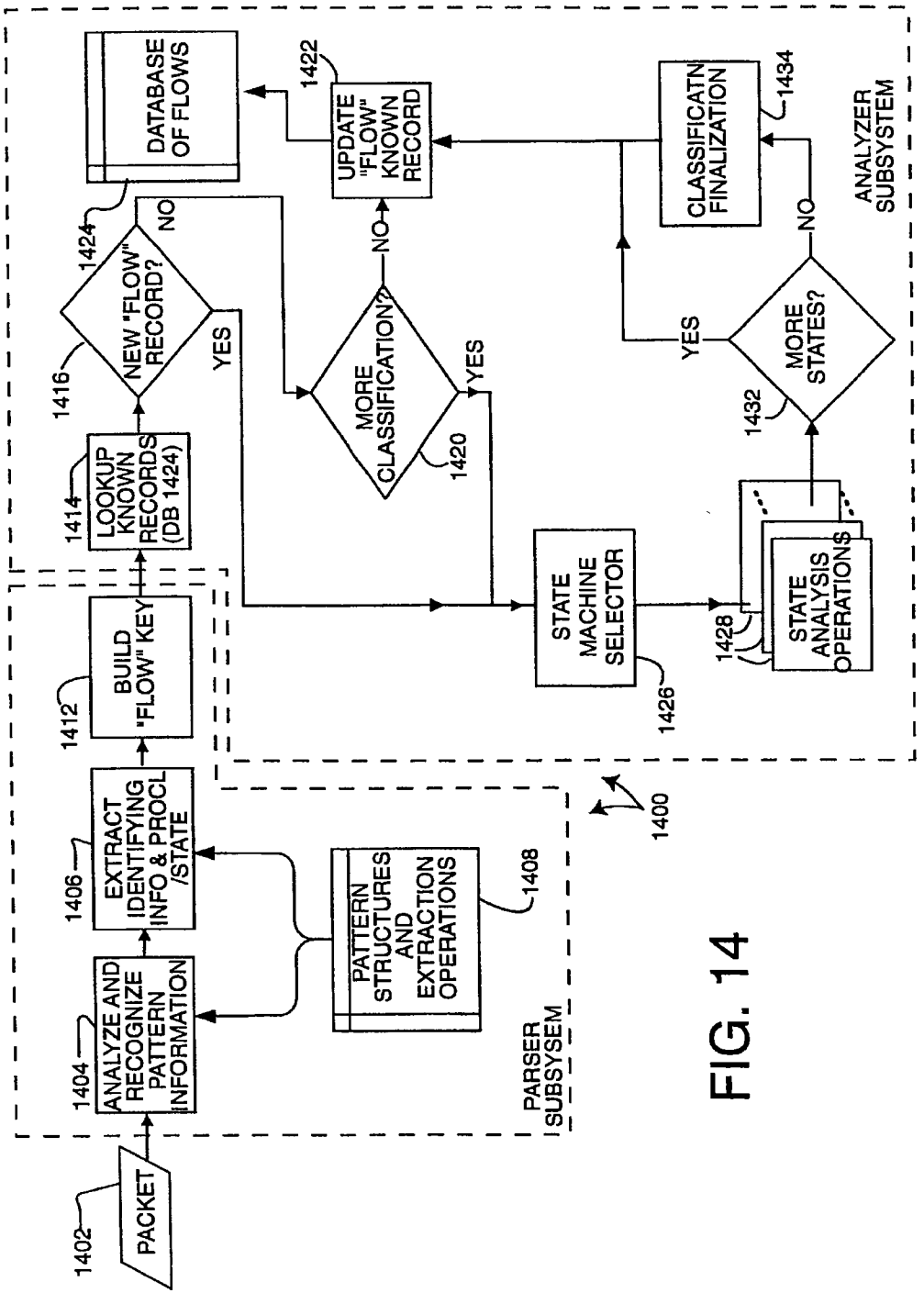

1402
PACKET
1404
ANALYZE AND RECOGNIZE PATTERN INFORMATION
1406
EXTRACT IDENTIFYING INFO & PROCL /STATE
1408
PATTERN STRUCTURES AND EXTRACTION OPERATIONS
PARSER SUBSYSEM
1412
BUILD "FLOW" KEY
1414
LOOKUP KNOWN RECORDS (DB 1424)
1416
NEW "FLOW" RECORD?
NO
YES
1420
MORE CLASSIFICATION?
NO
YES
1422
UPDATE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

FIG. 14

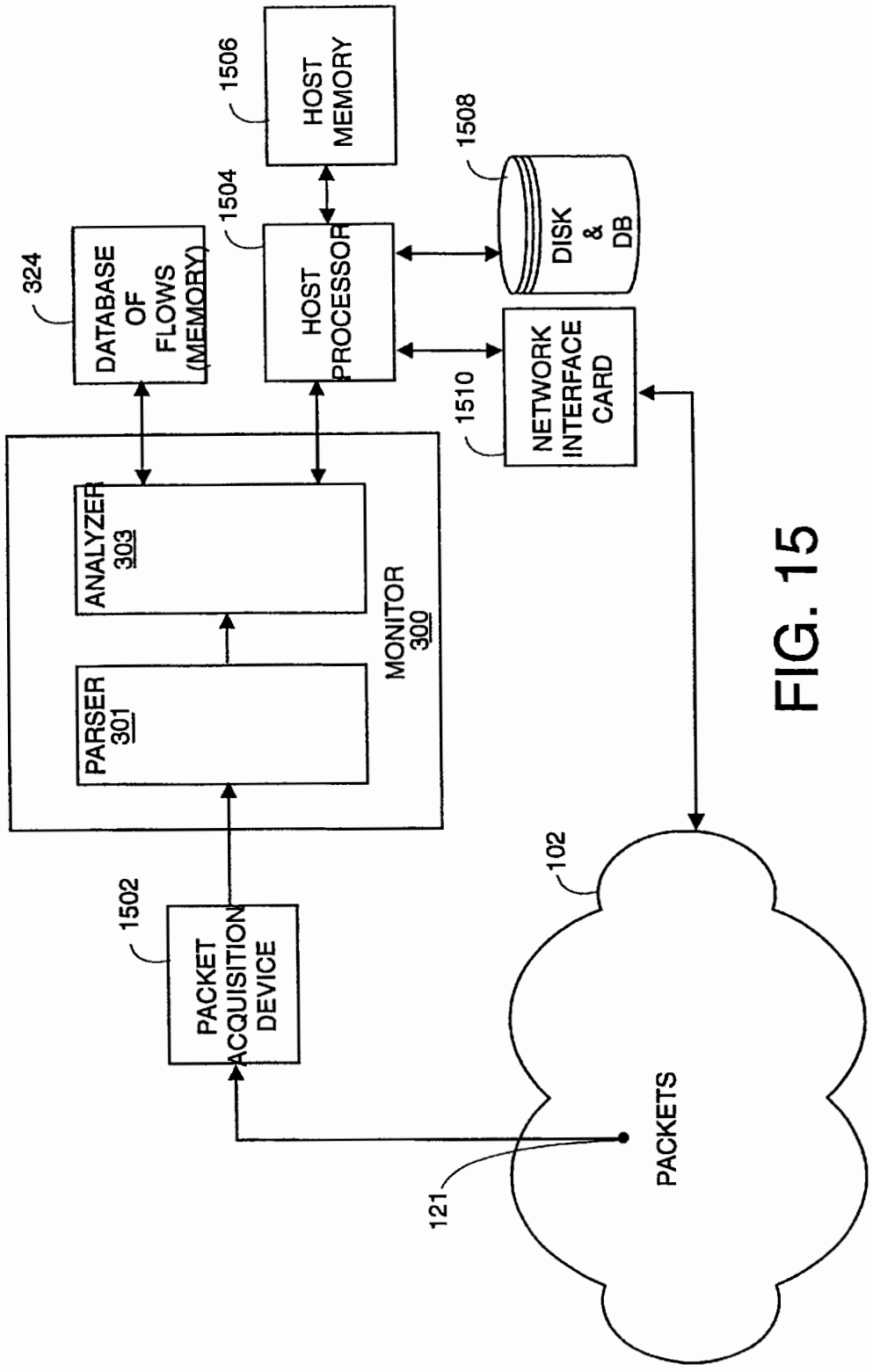

HOST MEMORY
1506
1508
DISK & DB
1504
HOST PROCESSOR
324
DATABASE OF FLOWS (MEMORY)
NETWORK INTERFACE CARD
1510
ANALYZER 303
MONITOR 300
PARSER 301
102
1502
PACKET ACQUISITION DEVICE
PACKETS
121

FIG. 15

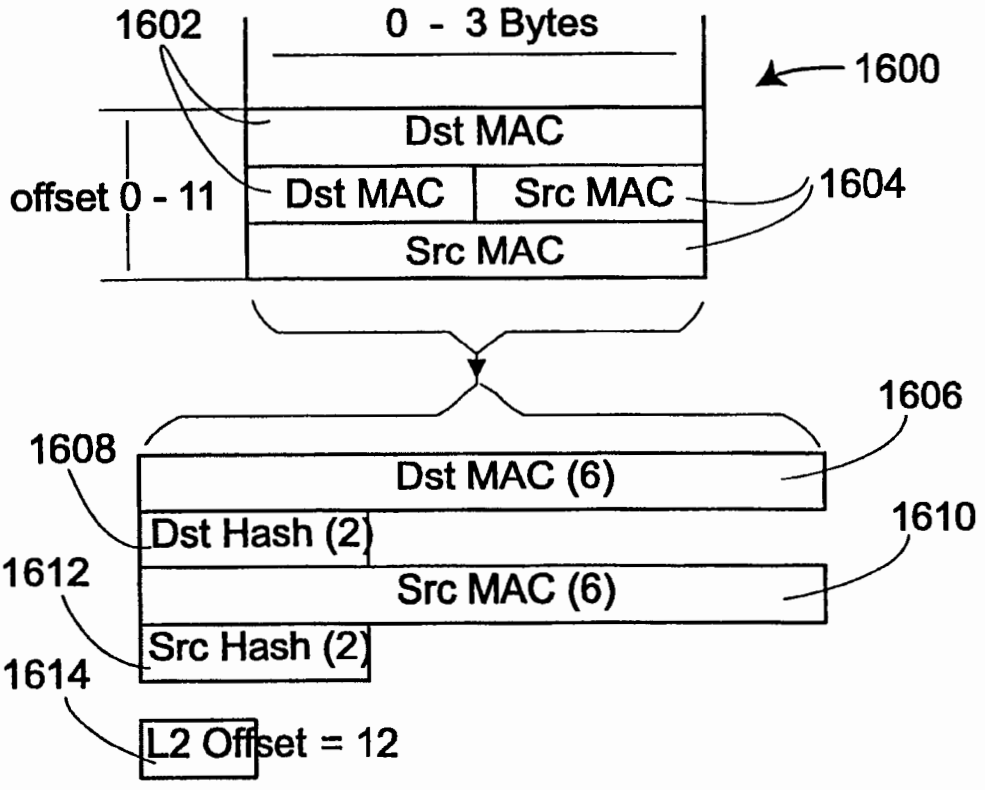

1602
0 - 3 Bytes
1600
Dst MAC
offset 0 - 11
Dst MAC
Src MAC
1604
Src MAC
1606
1608
Dst MAC (6)
Dst Hash (2)
1610
1612
Src MAC (6)
Src Hash (2)
1614
L2 Offset = 12

FIG. 16

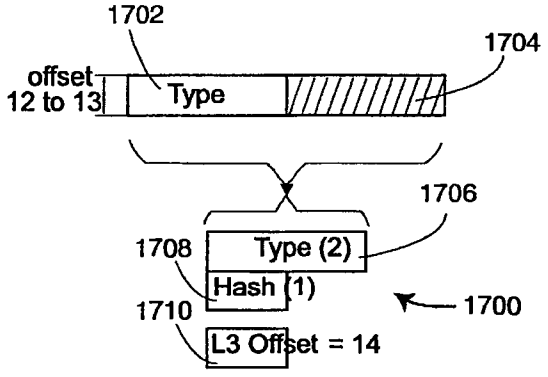


FIG. 17A

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

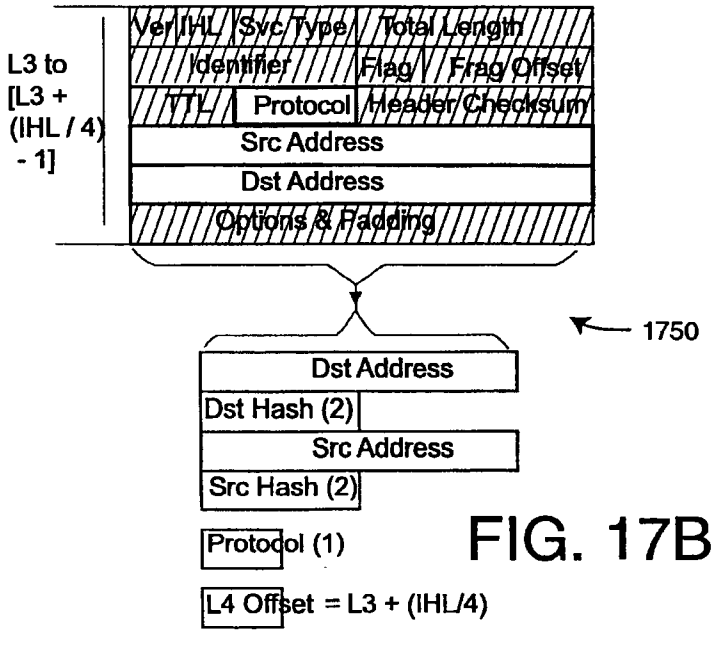


FIG. 17B

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

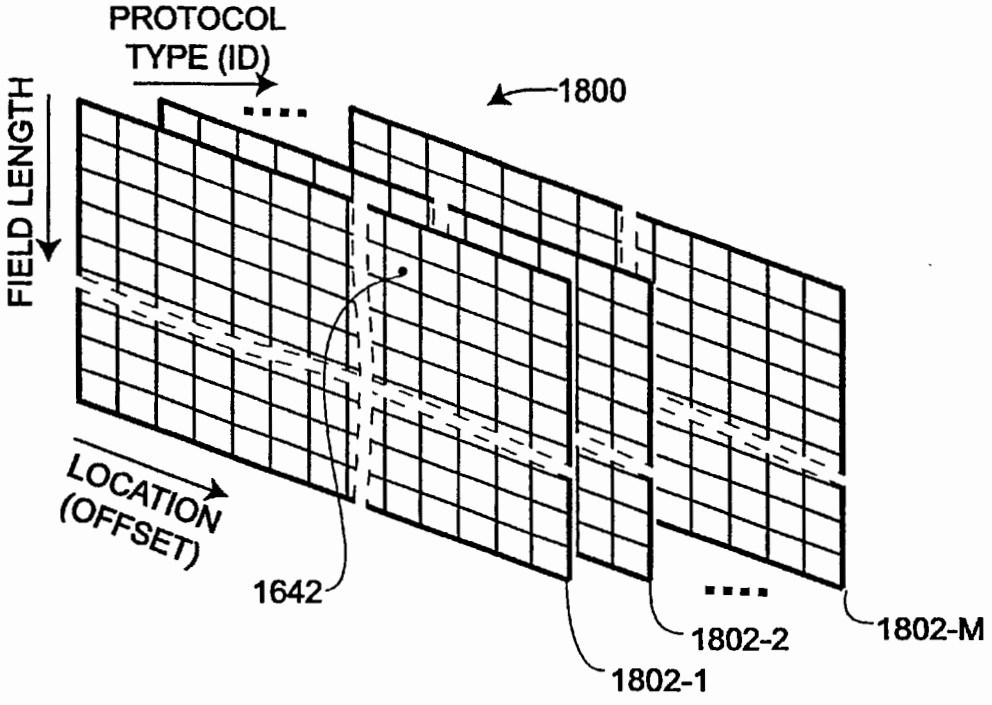


FIG. 18A

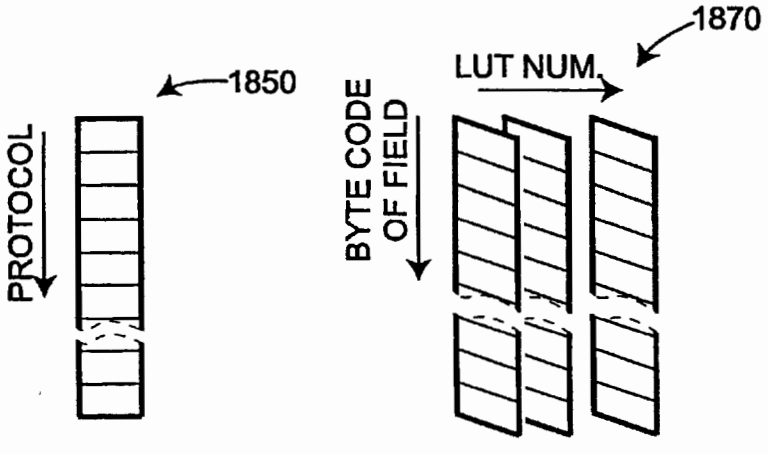


FIG. 18B

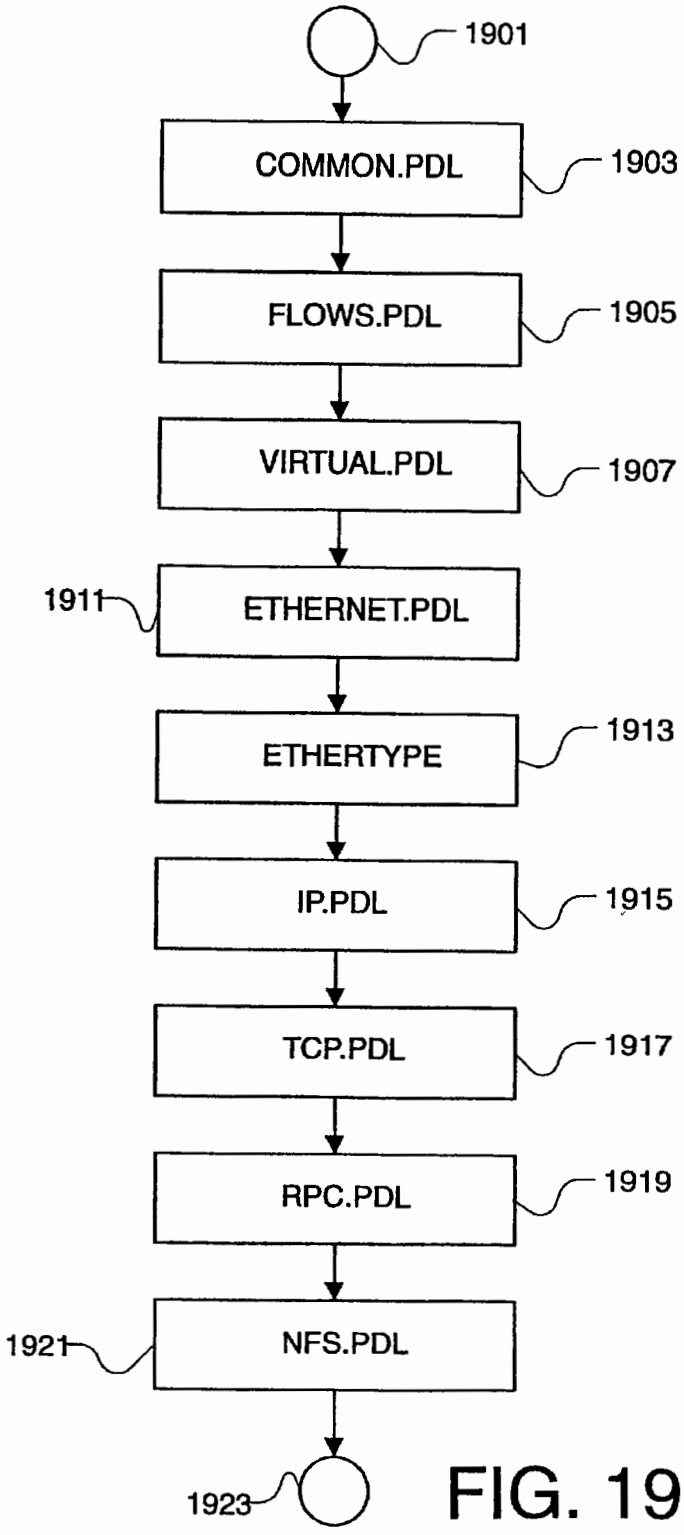

1901
COMMON.PDL
1903
FLOWS.PDL
1905
VIRTUAL.PDL
1907
1911
ETHERNET.PDL
ETHERTYPE
1913
IP.PDL
1915
TCP.PDL
1917
RPC.PDL
1919
1921
NFS.PDL
1923

FIG. 19

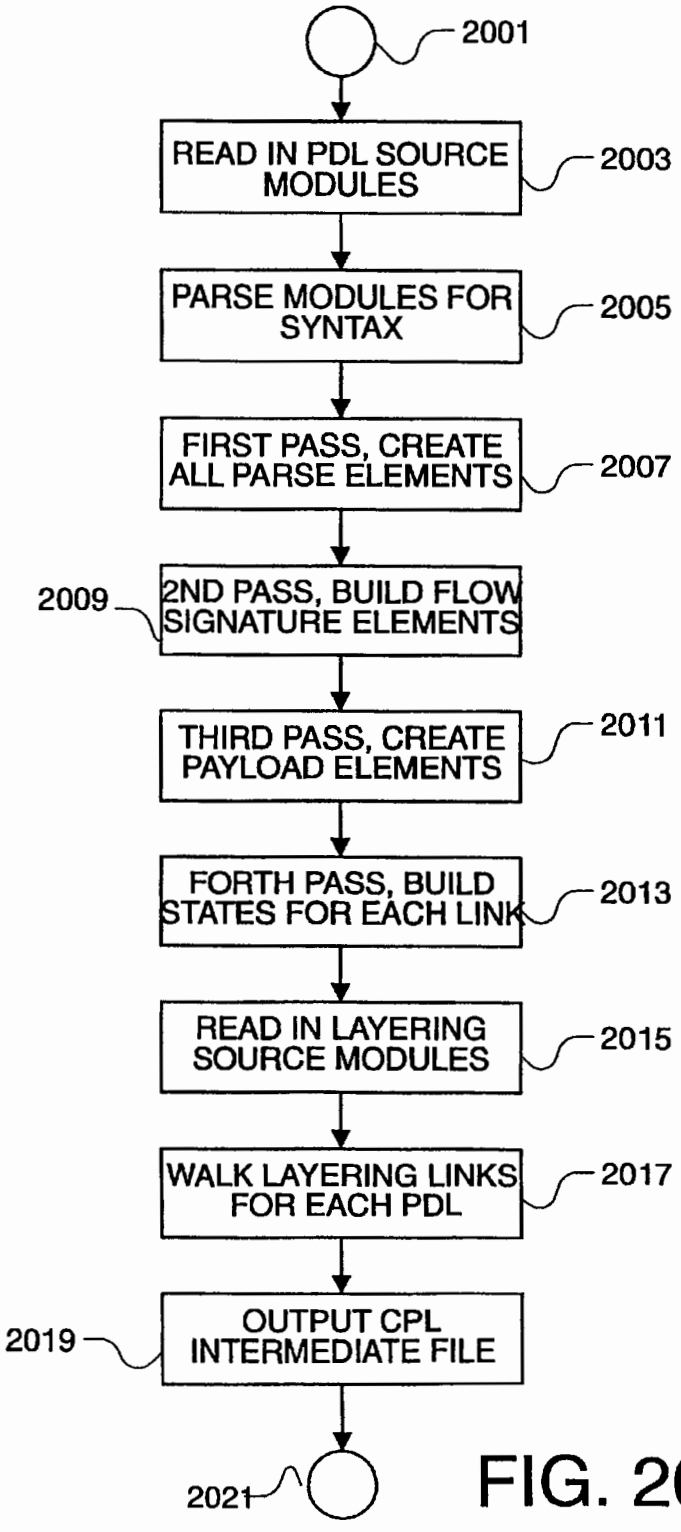

2001
READ IN PDL SOURCE MODULES
2003
PARSE MODULES FOR SYNTAX
2005
FIRST PASS, CREATE ALL PARSE ELEMENTS
2007
2009
2ND PASS, BUILD FLOW SIGNATURE ELEMENTS
THIRD PASS, CREATE PAYLOAD ELEMENTS
2011
FORTH PASS, BUILD STATES FOR EACH LINK
2013
READ IN LAYERING SOURCE MODULES
2015
WALK LAYERING LINKS FOR EACH PDL
2017
2019
OUTPUT CPL INTERMEDIATE FILE
2021

FIG. 20

PROCESSING PROTOCOL SPECIFIC INFORMATION IN PACKETS SPECIFIED BY A PROTOCOL DESCRIPTION LANGUAGE

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 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. 09/608,237 for METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK, to inventors Dietz, et al., filed Jun. 30, 2000, 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, 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, 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, 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

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.

The recognizing and classifying in such a network monitor should be at all protocol layer levels in conversational flows that pass in either direction at a point in a network. Furthermore, the monitor should provide for properly analyzing each of the packets exchanged between a client and a server, maintaining information relevant to the current state of each of these conversational flows.

Related and incorporated by reference U.S. patent application Ser. No. 09/608,237 for METHOD AND APPARATUS FOR MONITORING TRAFFIC IN A NETWORK, to inventors Dietz, et al, 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.

A network analyzer should be able to analyze many different protocols. At a base level, there are a number of standards used in digital telecommunications, including Ethernet, HDLC, ISDN, Lap B, ATM, X.25, Frame Relay, Digital Data Service, FDDI (Fiber Distributed Data Interface), T1, and others. Many of these standards employ different packet and/or frame formats. For example, data is

transmitted in ATM and frame-relay systems in the form of fixed length packets (called "cells") that are 53 octets (i.e., bytes) long. Several such cells may be needed to make up the information that might be included in the packet employed by some other protocol for the same payload information—for example in a conversational flow that uses the frame-relay standard or the Ethernet protocol.

In order for a network monitor to be able to analyze different packet or frame formats, the monitor needs to be able to perform protocol specific operations on each packet with each packet carrying information conforming to different protocols and related to different applications. For example, the monitor needs to be able to parse packets of different formats into fields to understand the data encapsulated in the different fields. As the number of possible packet formats or types increases, the amount of logic required to parse these different packet formats also increases.

Prior art network monitors exist that parse individual packets and look for information at different fields to use for building a signature for identifying packets. 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." In this patent, there are fixed locations specified for particular types of packets. For example, if a DECnet packet appears, the Chiu system 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 locations are examined. The system looks only at the lowest levels up to the protocol layer. There are fixed locations for each of the fields that specified the next level. 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.

It is desirable to be able to adaptively determine the locations and the information extracted from any packet for the particular type of packet. In this way, an optimal signature may be defined using a protocol-dependent and packet-content-dependent definition of what to look for and where to look for it in order to form a signature.

There thus is also a need for a network monitor that can be tailored or adapted for different protocols and for different application programs. There thus is also a need for a network monitor that can accommodate new protocols and for new application programs. There also is a need for means for specifying new protocols and new levels, including new applications. There also is a need for a mechanism to describe protocol specific operations, including, for example, what information is relevant to packets and packets that need to be decoded, and to include specifying parsing operations and extraction operations. There also is a need for a mechanism to describe state operations to perform on packets that are at a particular state of recognition of a flow in order to further recognize the flow.

SUMMARY

One embodiment of the invention is a method of performing protocol specific operations on a packet passing through a connection point on a computer network. The packet contents conform to protocols of a layered model wherein the protocol at a particular layer level may include one or a set of child protocols defined for that level. The method includes receiving the packet and receiving a set of protocol descriptions for protocols may be used in the packet. A protocol description for a particular protocol at a particular layer level includes any child protocols of the particular protocol, and for any child protocol, where in the packet information related to the particular child protocol may be found. A protocol description also includes any protocol specific operations to be performed on the packet for the particular protocol at the particular layer level. The method includes performing the protocol specific operations on the packet specified by the set of protocol descriptions based on the base protocol of the packet and the children of the protocols used in the packet. A particular embodiment includes providing the protocol descriptions in a high-level protocol description language, and compiling to the descriptions into a data structure. The compiling may further include compressing the data structure into a compressed data structure. The protocol specific operations may include parsing and extraction operations to extract identifying information. The protocol specific operations may also include state processing operations defined for a particular state of a conversational flow of the packet.

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.

FIG. **19** shows various PDL file modules to be compiled together by the compiling process illustrated in FIG. **20** as an example, in accordance with a compiling aspect of the invention.

FIG. **20** is a flowchart of the process of compiling high-level language files according to an aspect 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 from 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, NBTBIOS, 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)

Diferent communications 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 multi-level 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. **18**A 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. **18**B.

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 updatelookup 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 **110** 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 subsystem **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, 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.

The Pattern Parse and Extraction Database Format

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 base level). Each protocol is either a parent node of some other protocol at the next later 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.

As an example of the tree structure, consider an Ethernet 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. Another child of the IP may be the UDP protocol.

A packet includes at least one header for each protocol used. The child protocol of a particular protocol used in a packet is indicated by the contents at a location within the header of the particular protocol. The contents of the packet that specify the child are in the form of a child recognition pattern.

A network analyzer preferably can analyze many different protocols. At a base level, there are a number of packet types used in digital telecommunications, including Ethernet, HDLC, ISDN, Lap B, ATM, X.25, Frame Relay, Digital Data Service, FDDI (Fiber Distributed Data Interface), and T1, among others. Many of these packet types use different packet and/or frame formats. For example, data is transmitted in ATM and frame-relay systems in the form of fixed length packets (called "cells") that are 53 octets (i.e., bytes) long; several such cells may be needed to make up the information that might be included in a single packet of some other type.

Note that the term packet herein 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," "frame format" and "cell format" are generally synonymous.

The packet monitor **300** can analyze different protocols, and thus can perform different protocol specific operations on a packet wherein the protocol headers of any protocol are located at different locations depending on the parent protocol or protocols used in the packet. Thus, the packet monitor adapts to different protocols according to the contents of the packet. The locations and the information extracted from any packet are adaptively determined for the particular type of packet. For example, there is no fixed definition of what to look for or where to look in order to form the flow signature. In some prior art systems, such as that described in U.S. Pat. No. 5,101,402 to Chiu, et al., there are fixed locations specified for particular types of packets. With the proliferation of protocols, 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 number of levels is variable for any protocol and is whatever number

is sufficient to uniquely identify as high up the level system as we wish to go, all the way to the application level (in the OSI model).

Even the same protocol may have different variants. Ethernet packets for example, have several known variants, each having a basic format that remains substantially the same. 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 Ethernet (IEEE 803.x) packet. A monitor should be able to handle all types of Ethernet protocols. With the Ethertype protocol, the contents that indicate the child protocol is in one location, while with an IEEE type, the child protocol is specified in a different location. The child protocol is indicated by a child recognition pattern.

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. Such information is also to be specified in the parsing structures and extraction operations database **308**. This includes all of the header information at this level in the form of 6 bytes of Dst MAC information **1606** and 6 bytes of Src MAC information **1610**. Also specified are the source and destination address components, respectively, of the hash. These are shown as 2 byte Dst Hash **1608** from the Dst MAC address and the 2 byte Src Hash **1612** from the Src MAC address. Finally, information is included (**1614**) on where to the header starts for information related to the next layer level. In this case the next layer level (level 2) information starts at packet offset **12**.

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**.

Also shown is some of the extracted part used for the parser record and to locate the next header information. The signature part of the parser record includes extracted part **1702**. Also included is the 1-byte Hash component **1710** from this information.

An offset field **1710** provides the offset to go to the next level information, i.e., to locate the start of the next layer level header. For the Ethertype packet, the start of the next layer header 14 bytes from the start of the frame.

Other packet types are arranged differently. For example, in an ATM system, each ATM packet comprises a five-octet "header" segment followed by a forty-eight octet "payload" segment. The header segment of an ATM cell contains information relating to the routing of the data contained in the payload segment. The header segment also contains traffic control information. Eight or twelve bits of the header segment contain the Virtual Path Identifier (VPI), and sixteen bits of the header segment contain the Virtual Channel Identifier (VCI). Each ATM exchange translates the abstract routing information represented by the VPI and VCI bits into the addresses of physical or logical network links and routes each ATM cell appropriately.

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 header starts at a different location (L3) depending on the parent protocol. Also included in FIG. **17B** are some of the fields to be extracted for the signature, and an indication of where the next level's header would start in the packet.

Note that the information shown in FIGS. **16**, **17A**, and **17B** would be specified to the monitor in the form of PDL files and compiled into the database **308** of pattern structures and extraction operations.

The parsing subsystem **301** performs operations on the packet header data based on information stored in the database **308**. Because data related to protocols can be considered as organized in the form of a tree, it is required in the parsing subsystem to search through data that is originally organized in the form of a tree. Since real time operation is preferable, it is required to carry out such searches rapidly.

Data structures are known for efficiently storing information organized as trees. Such storage-efficient means typically require arithmetic computations to determine pointers to the data nodes. Searching using such storage-efficient data structures may therefore be too time consuming for the present application. It is therefore desirable to store the protocol data in some form that enables rapid searches.

In accordance with another aspect of the invention, the database **308** is stored in a memory and includes a data structure used to store the protocol specific operations that are to be performed on a packet. In particular, a compressed representation is used to store information in the pattern parse and extraction database **308** used by the pattern recognition process **304** and the extraction process **306** in the parser subsystem **301**. The data structure is organized for rapidly locating the child protocol related information by using a set of one or more indices to index the contents of the data structure. A data structure entry includes an indication of validity. Locating and identifying the child protocol includes indexing the data structure until a valid entry is found. Using the data structure to store the protocol information used by the pattern recognition engine (PRE) **1006** enables the parser subsystem **301** to perform rapid searches.

In one embodiment, the data structure is in the form of a three-dimensional structure. Note that this three dimensional structure in turn is typically stored in memory as a set of two-dimensional structures whereby one of the three dimensions of the 3-D structure is used as an index to a particular 2-D array. This forms a first index to the data structure.

FIG. **18A** shows such a 3-D representation **1800** (which may be considered as an indexed set of 2-D representations). The three dimensions of this data structure are:

1. Type identifier [1:M]. This is the identifier that identifies a type of protocol at a particular level. For example, 01 indicates an Ethernet frame. 64 indicates IP, 16 indicates an IEEE type Ethernet packet, etc. Depending on how many protocols the packet parser can handle, M may be a large number; M may grow over time as the capability of analyzing more protocols is added to monitor **300**. When the 3-D structure is considered a set of 2-D structures, the type ID is an index to a particular 2-D structure.
2. Size [1:64]. The size of the field of interest within the packet.
3. Location [1:512]. This is the offset location within the packet, expressed as a number of octets (bytes).

At any one of these locations there may or may not be valid data. Typically, there will not be valid data in most

locations. The size of the 3-D array is M by 64 by 512, which can be large; M for example may be 10,000. This is a sparse 3-D matrix with most entries empty (i.e., invalid).

Each array entry includes a "node code" that indicates the nature of the contents. This node code has one of four values: (1) a "protocol" node code indicating to the pattern recognition process **304** that a known protocol has been recognized as the next (i.e., child) protocol; (2) a "terminal" node code indicating that there are no children for the protocol presently being searched, i.e., the node is a final node in the protocol tree; (3) a "null" (also called "flush") node code indicating that there is no valid entry.

In the preferred embodiment, the possible children and other information are loaded into the data structure by an initialization that includes compilation process **310** based on the PDL files **336** and the layering selections **338**. The following information is included for any entry in the data structure that represents a protocol.

(a) A list of children (as type IDs) to search next. For example, for an Ethernet type 2, the children are Ethertype (IP, IPX, etc, as shown in **1712** of FIG. **17**). These children are compiled into the type codes. The code for IP is 64, that for IPX is 83, etc.

(b) For each of the IDs in the list, a list of the child recognition patterns that need to be compared. For example, $64{:}0800_{16}$ in the list indicates that the value to look for is 0800 (hex) for the child to be type ID 64 (which is the IP protocol). $83{:}8137_{16}$ in the list indicates that the value to look for is 8137 (hex) for the child to be type ID **83** (which is the IPX protocol), etc.

(c) The extraction operations to perform to build the identifying signature for the flow. The format used is (offset, length, flow_signature_value_identifier), the flow_signature_value_identifier indicating where the extracted entry goes in the signature, including what operations (AND, ORs, etc.) may need to be carried out. If there is also a hash key component, for instance, then information on that is included. For example, for an Ethertype packet, the 2-byte type (**1706** in FIG. **17**) is used in the signature. Furthermore, a 1-byte hash (**1708** in FIG. **17A**) of the type is included. . Note furthermore, the child protocol starts at offset **14**.

An additional item may be the "fold." Folding is used to reduce the storage requirements for the 3-D structure. Since each 2-D array for each protocol ID may be sparsely populated, multiple arrays may be combined into a single 2-D array as long as the individual entries do not conflict with each other. A fold number is then used to associate each element. For a given lookup, the fold number of the lookup must match the fold number entry. Folding is described in more detail below.

In the case of the Ethernet, the next protocol field may indicate a length, which tells the parser that this is a IEEE type packet, and that the next protocol is elsewhere. Normally, the next protocol field contains a value which identifies the next, i.e., child protocol.

The entry point for the parser subsystem is called the virtual base layer and contains the possible first children, i.e., the packet types. An example set of protocols written in a high level protocol description language (PDL) is included herein. The set includes PDL files, and the file describing all the possible entry points (i.e., the virtual base) is called virtual.pdl. There is only one child, 01, indicating the Ethernet, in this file. Thus, the particular example can only handle Ethernet packets. In practice, there can be multiple entry points.

In one embodiment, the packet acquisition device provides a header for every packet acquired and input into monitor **300** indicating the type of packet. This header is used to determine the virtual base layer entry point to the parser subsystem. Thus, even at the base layer, the parser subsystem can identify the type of packet.

Initially, the search starts at the child of the virtual base, as obtained in the header supplied by the acquisition device. In the case of the example, this has ID value 01, which is the 2-D array in the overall 3-D structure for Ethernet packets.

Thus hardware implementing pattern analysis process **304** (e.g., pattern recognition engine (PRE) **1006** of FIG. **10**) searches to determine the children (if any) for the 2-D array that has protocol ID 01. In the preferred embodiment that uses the 3-D data structure, the hardware PRE **1006** searches up to four lengths (i.e., sizes) simultaneously. Thus, the process **304** searches in groups of four lengths. Starting at protocol ID 01, the first two sets of 3-D locations searched are

(1, 1, 1)	(1, 1, 2)	. . .
(1, 2, 1)	(1, 2, 2)	
(1, 3, 1)	(1, 3, 2)	
(1, 4, 1)	(1, 4, 2)	

At each stage of a search, the analysis process **304** examines the packet and the 3-D data structure to see if there is a match (by looking at the node code). If no valid data is found, e.g., using the node code, the size is incremented (to maximum of 4) and the offset is then incremented as well.

Continuing with the example, suppose the pattern analysis process **304** finds something at 1, 2, 12. By this, we mean that the process **304** has found that for protocol ID value 01 (Ethernet) at packet offset **12**, there is information in the packet having a length of 2 bytes (octets) that may relate to the next (child) protocol. The information, for example, may be about a child for this protocol expressed as a child recognition pattern. The list of possible child recognition patterns that may be in that part of the packet is obtained from the data structure.

The Ethernet packet structure comes in two flavors, the Ethertype packet and newer IEEE types, and the packet location that indicates the child is different for both. The location that for the Ethertype packet indicates the child is a "length" for the IEEE type, so a determination is made for the Ethernet packet whether the "next protocol" location contains a value or a length (this is called a "LENGTH" operation). A successful LENGTH operation is indicated by contents less than or equal to $05DC_{16}$, then this is an IEEE type Ethernet frame. In such a case, the child recognition pattern is looked for elsewhere. Otherwise, the location contains a value that indicates the child.

Note that while this capability of the entry being a value (e.g., for a child protocol ID) or a length (indicating further analysis to determine the child protocol) is only used for Ethernet packets, in the future, other packets may end up being modified. Accordingly, this capability in the form of a macro in the PDL files still enables such future packets to be decoded.

Continuing with the example, suppose that the LENGTH operation fails. In that case, we have an Ethertype packet, and the next protocol field (containing the child recognition pattern) is 2 bytes long starting at offset **12** as shown as packet field **1702** in FIG. **17A**. This will be one of the children of the Ethertype shown in table **1712** in FIG. **17A**. The PRE uses the information in the data structure to check what the ID code is for the found 2-byte child recognition pattern. For example, if the child recognition pattern is 0800

(Hex), then the protocol is IP. If the child recognition pattern is 0BAD (Hex) the protocol is VIP (VINES).

Note that an alternate embodiment may keep a separate table that includes all the child recognition patterns and their corresponding protocol ID's

To follow the example, suppose the child recognition pattern at **1, 2, 12** is 0800_{16}, indicating IP. The ID code for the IP protocol is 64_{10}). To continue with the Ethertype example, once the parser matches one of the possible children for the protocl—in the example, the protocol type is IP with an ID of 64—then the parser continues the search for the next level. The ID is 64, the length is unknown, and offset is known to be equal or larger than 14 bytes (**12** offset for type, plus 2, the length of type), so the search of the 3-D structure commences from location **(64, 1)** at packet offset **14**. A populated node is found at **(64, 2)** at packet offset **14**. Heading details are shown as **1750** in FIG. **17B**. The possible children are shown in table **1752**.

Alternatively, suppose that at **(1, 2, 12)** there was a length 1211_{10}. This indicates that this is an IEEE type Ethernet frame, which stores its type elsewhere. The PRE now continues its search at the same level, but for a new ID, that of an IEEE type Ethernet frame. An IEEE Ethernet packet has protocol ID 16, so the PRE continues its search of the three-dimensional space with ID 16starting at packet offset **14**.

In our example, suppose there is a "protocol" node code found at **(16, 2)** at packet offset **14**, and the next protocol is specified by child recognition pattern 0800_{16}. This indicates that the child is the IP protocol, which has type ID 64. Thus the search continues, starting at **(64, 1)** at packet offset **16**. Compression.

As noted above, the 3-D data structure is very large, and sparsely populated. For example, if 32 bytes are stored at each location, then the length is M by 64 by 512 by 32 bytes, which is M megabytes. If M=10,000, then this is about 10 gigabytes. It is not practical to include 10 Gbyte of memory in the parser subsystem for storing the database **308**. Thus a compressed form of storing the data is used in the preferred embodiment. The compression is preferably carried out by an optimizer component of the compilation process **310**.

Recall that the data structure is sparse. Different embodiments may use different compression schemes that take advantage of the sparseness of the data structure. One embodiment uses a modification of multi-dimensional run length encoding.

Another embodiment uses a smaller number two-dimensional structures to store the information that otherwise would be in one large three-dimensional structure. The second scheme is used in the preferred embodiment.

FIG. **18A** illustrated how the 3-D array **1800** can be considered a set of 2-D arrays, one 2-D array for each protocol (i.e., each value of the protocol ID). The 2-D structures are shown as **1802-1, 1802-2, . . . , 1802-M** for up to M protocol ID's. One table entry is shown as **1804**. Note that the gaps in table are used to illustrate that each 2-D structure table is typically large.

Consider the set of trees that represent the possible protocols. Each node represents a protocol, and a protocol may have a child or be a terminal protocol. The base (root) of the tree has all packet types as children. The other nodes form the nodes in the tree at various levels from level 1 to the final terminal nodes of the tree. Thus, one element in the base node may reference node ID **1**, another element in the base node may reference node ID **2** and so on. As the tree is traversed from the root, there may be points in the tree where the same node is referenced next. This would occur, for example, when an application protocol like Telnet can run on several transport connections like TCP or UDP. Rather than repeating the Telnet node, only one node is represented in the patterns database **308** which can have several parents. This eliminates considerable space explosion.

Each 2-D structure in FIG. **18A** represents 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.

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.

Assume two 2-D arrays are being considered for folding. Call the first 2-D arrays A and the second 2-D array B. Since both 2-D arrays are partially populated, 2-D array B can be combined with 2-D arrays A if and only if none of the individual elements of these two 2-D arrays that have the same 2-D location conflict. If the result is foldable, then the valid entries of 2-D array B are combined with the valid entries of 2-D array A yielding one physical 2-D array. However, it is necessary to be able to distinguish the original 2-D array A entries from those of 2-D array B. For example, if a parent protocol of the protocol represented by 2-D array B wants to reference the protocol ID of 2-D array B, it must now reference 2-D array A instead. However, only the entries that were in the original 2-D array B are valid entries for that lookup. To accomplish this, each element in any given 2-D array is tagged with a fold number. When the original tree is created, all elements in all the 2-D arrays are initialized with a fold value of zero. Subsequently, if 2-D array B is folded into 2-D array A, all valid elements of 2-D array B are copied to the corresponding locations in 2-D array A and are given different fold numbers than any of the elements in 2-D array A. For example, if both 2-D array A and 2-D array B were original 2-D arrays in the tree (i.e., not previously folded) then, after folding, all the 2-D array A entries would still have fold **0** and the 2-D array B entries would now all have a fold value of 1. After 2-D array B is folded into 2-D array A, the parents of 2-D array B need to be notified of the change in the 2-D array physical location of their children and the associated change in the expected fold value.

This folding process can also occur between two 2-D arrays that have already been folded, as long as none of the individual elements of the two 2-D arrays conflict for the same 2-D array location. As before, each of the valid elements in 2-D array B must have fold numbers assigned to them that are unique from those of 2-D array A. This is accomplished by adding a fixed value to all the 2-D array B fold numbers as they are merged into 2-D array A. This fixed value is one larger than the largest fold value in the original 2-D array A. It is important to note that the fold number for any given 2-D array is relative to that 2-D array only and does not span across the entire tree of 2-D arrays.

This process of folding can now be attempted between all combinations of two 2-D arrays until there are no more candidates that qualify for folding. By doing this, the total number of 2-D arrays can be significantly reduced.

Whenever a fold occurs, the 3-D structure (i.e., all 2-D arrays) must be searched for the parents of the 2-D array being folded into another array. The matching pattern which previously was mapped to a protocol ID identifying a single 2-D array must now be replaced with the 2-D array ID and the next fold number (i.e., expected fold).

Thus, in the compressed data structure, each entry valid entry includes the fold number for that entry, and additionally, the expected fold for the child.

An alternate embodiment of the data structure used in database **308** is illustrated in FIG. **18**B. Thus, like the 3-D structure described above, it permits rapid searches to be performed by the pattern recognition process **304** by indexing locations in a memory rather than performing address link computations. The structure, like that of FIG. **18**A, is suitable for implementation in hardware, for example, for implementation to work with the pattern recognition engine (PRE) **1006** of FIG. **10**.

A table **1850**, called the protocol table (PT) has an entry for each protocol known by the monitor **300**, and includes some of the characteristics of each protocol, including a description of where the field that specifies next protocol (the child recognition pattern) can be found in the header, the length of the next protocol field, flags to indicate the header length and type, and one or more slicer commands, the slicer can build the key components and hash components for the packet at this protocol at this layer level.

For any protocol, there also are one or more lookup tables (LUTs). Thus database **308** for this embodiment also includes a set of LUTs **1870**. Each LUT has 256 entries indexed by one byte of the child recognition pattern that is extracted from the next protocol field in the packet. Such a protocol specification may be several bytes long, and so several of LUTs **1870** may need to be looked up for any protocol.

Each LUT's entry includes a 2-bit "node code" that indicates the nature of the contents, including its validity. This node code has one of four values: (1) a "protocol" node code indicating to the pattern recognition engine **1006** that a known protocol has been recognized; (2) an "intermediate" node code, indicating that a multi-byte protocol code has been partially recognized, thus permitting chaining a series of LUTs together before; (3) a "terminal" node code indicating that there are no children for the protocol presently being searched, i.e., the node is a final node in the protocol tree; (4) a "null" (also called "flush" and "invalid") node code indicating that there is no valid entry.

In addition to the node code, each LUT entry may include the next LUT number, the next protocol number (for looking up the protocol table **1850**), the fold of the LUT entry, and the next fold to expect. Like in the embodiment implementing a compressed form of the 3-D representation, folding is used to reduce the storage requirements for the set of LUTs. Since the LUTs **1870** may be sparsely populated, multiple LUTs may be combined into a single LUT 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 LUT.

For a given lookup, the fold number of the lookup must match the fold number in the lookup table. The expected fold is obtained from the previous table lookup (the "next fold to expect" field). The present implementation uses 5-bits to describe the fold and thus allows up to 32 tables to be folded into one table.

When using the data structure of FIG. **18**B, when a packet arrives at the parser, the virtual base has been pre-pended or is known. The virtual base entry tells the packet recognition engine where to find the first child recognition pattern in the packet. The pattern recognition engine then extracts the child recognition pattern bytes from the packet and uses them as an address into the virtual base table (the first LUT). If the entry looked up in the specified next LUT by this method matches the expected next fold value specified in the virtual base entry, the lookup is deemed valid. The node code is then examined. If it is an intermediate node then the next table field obtained from the LUT lookup is used as the most significant bits of the address. The next expected fold is also extracted from the entry. The pattern recognition engine **1006** then uses the next byte from the child recognition pattern as the for the next LUT lookup.

Thus, the operation of the PRE continues until a terminal code is found. The next (initially base layer) protocol is looked up in the protocol table **1850** to provide the PRE **1006** with information on what field in the packet (in input buffer memory **1008** of parser subsystem **1000**) to use for obtaining the child recognition pattern of the next protocol, including the size of the field. The child recognition pattern bytes are fetched from the input buffer memory **1008**. The number of bytes making up the child recognition pattern is also now known.

The first byte of the protocol code bytes is used as the lookup in the next LUT. If a LUT lookup results in a node code indicating a protocol node or a terminal node, the Next LUT and next expected fold is set, and the "next protocol" from LUT lookup is used as an index into the protocol table **1850**. This provides the instructions to the slicer **1007**, and where in the packet to obtain the field for the next protocol. Thus, the PRE **1006** continues until it is done processing all the fields (i.e., the protocols), as indicated by the terminal node code reached.

Note that when a child recognition pattern is checked against a table there is always an expected fold. If the expected fold matches the fold information in the table, it is used to decide what to do next. If the fold does not match, the optimizer is finished.

Note also that an alternate embodiment may use different size LUTs, and then index a LUT by a different amount of the child recognition pattern.

The present implementation of this embodiment allows for child recognition patterns of up to four bytes. Child recognition patterns of more than 4 bytes are regarded as special cases.

In the preferred embodiment, the database is generated by the compiler process **310**. The compiler process first builds a single protocol table of all the links between protocols. Links consist of the connection between parent and child protocols. Each protocol can have zero or more children. If a protocol has children, a link is created that consists of the parent protocol, the child protocol, the child recognition pattern, and the child recognition pattern size. The compiler first extracts child recognition patterns that are greater than two bytes long. Since there are only a few of these, they are handled separately. Next sub links are created for each link that has a child recognition pattern size of two.

All the links are then formed into the LUTs of 256 entries.

Optimization is then carried out. The first step in the optimization is checking all the tables against all the other tables to find out which ones can share a table. This process proceeds the same way as described above for two-dimensional arrays, but now for the sparse lookup tables.

Part of the initialization process (e.g., compiler process **310**) loads a slicer instruction database with data items

including of instruction, source address, destination address, and length. The PRE **1006** when it sends a slicer instruction sends this instruction as an offset into the slicer instruction database. The instruction or Op code tells the slicer what to extract from the incoming packet and where to put it in the flow signature. Writing into certain fields of the flow signature automatically generates a hash. The instruction can also tell the slicer how to determine the connection status of certain protocols.

Note that alternate embodiments may generate the pattern, parse and extraction database other than by compiling PDL files.

The Compilation Process

The compilation process **310** is now described in more detail. This process **310** includes creating the parsing patterns and extractions database **308** that provides the parsing subsystem **301** with the information needed to parse packets and extract identifying information, and the state processing instructions database **326** that provides the state processes that need to be performed in the state processing operation **328**.

Input to the compiler includes a set of files that describe each of the protocols that can occur. These files are in a convenient protocol description language (PDL) which is a high level language. PDL is used for specifying new protocols and new levels, including new applications. The PDL is independent of the different types of packets and protocols that may be used in the computer network. A set of PDL files is used to describe what information is relevant to packets and packets that need to be decoded. The PDL is further used to specify state analysis operations. Thus, the parser subsystem and the analyzer subsystems can adapt and be adapted to a variety of different kinds of headers, layers, and components and need to be extracted or evaluated, for example, in order to build up a unique signature.

There is one file for each packet type and each protocol. Thus there is a PDL file for Ethernet packets and there is a PDL file for frame relay packets. The PDL files are compiled to form one or more databases that enable monitor **300** to perform different protocol specific operations on a packet wherein the protocol headers of any protocol are located at different locations depending on the parent protocol or protocols used in the packet. Thus, the packet monitor adapts to different protocols according to the contents of the packet. In particular, the parser subsystem **301** is able to extract different types of data for different types of packets. For example, the monitor can know how to interpret a Ethernet packet, including decoding the header information, and also how to interpret an frame relay packet, including decoding the header information.

The set of PDL files, for example, may include a generic Ethernet packet file. There also is included a PDL file for each variation Ethernet file, for example, an EEE Ethernet file.

The PDL file for a protocol provides the information needed by compilation process **310** to generate the database **308**. That database in turn tells the parser subsystem how to parse and/or extract information, including one or more of what protocol-specific components of the packet to extract for the flow signature, how to use the components to build the flow signature, where in the packet to look for these components, where to look for any child protocols, and what child recognition patterns to look for. For some protocols, the extracted components may include source and destination addresses, and the PDL file may include the order to use these addresses to build the key. For example, Ethernet frames have end-point addresses that are useful in building a better flow signature. Thus the PDL file for an Ethernet packet includes information on how the parsing subsystem is to extract the source and destination addresses, including where the locations and sizes of those addresses are. In a frame-relay base layer, for example, there are no specific end point addresses that help to identify the flow better, so for those type of packets, the PDL file does not include information that will cause the parser subsystem to extract the end-point addresses.

Some protocols also include information on connections. TCP is an example of such a protocol. Such protocol use connection identifiers that exist in every packet. The PDL file for such a protocol includes information about what those connection identifiers are, where they are, and what their length is. In the example of TCP, for example running over IP, these are port numbers. The PDL file also includes information about whether or not there are states that apply to connections and disconnections and what the possible children are states. So, at each of these levels, the packet monitor **300** learns more about the packet. The packet monitor **300** can identify that a particular packet is part of a particular flow using the connection identifier. Once the flow is identified, the system can determine the current state and what states to apply that deal with connections or disconnections that exist in the next layer up to these particular packets.

For the particular PDL used in the preferred embodiment, a PDL file may include none or more FIELD statement each defining a specific string of bits or bytes (i.e., a field) in the packet. A PDL file may further include none or more GROUP statements each used to tie together several defined fields. A set of such tied together fields is called a group. A PDL file may further include none or more PROTOCOL statements each defining the order of the fields and groups within the header of the protocol. A PDL file may further include none or more FLOW statements each defining a flow by describing where the address, protocol type, and port numbers are in a packet. The FLOW statement includes a description of how children flows of this protocol are determined using state operations. States associated may have state operations that may be used for managing and maintaining new states learned as more packets of a flow are analyzed.

FIG. **19** shows a set of PDL files for a layering structure for an Ethernet packet that runs TCP on top of IP. The contents of these PDL files are attached as an APPENDIX hereto. Common.pdl **(1903)** is a file containing the common protocol definitions, i.e., some field definitions for commonly used fields in various network protocols. Flows.pdl **(1905)** is a file containing general flow definitions. Virtual.pdl **(1907)** is a PDL file containing the definition for the VirtualBase layer used. Ethernet.pdl **(1911)** is the PDL file containing the definition for the Ethernet packet. The decision on Ethertype vs. IEEE type Ethernet file is described herein. If this is Ethertype, the selection is made from the file Ethertype.pdl **(1913)**. In an alternate embodiment, the Ethertype selection definition may be in the same Ethernet file **1911**. In a typical implementation, PDL files for other Ethernet types would be included. IP.pdl **(1915)** is a PDL file containing the packet definitions for the Internet Protocol. TCP.pdl **(1917)** is the PDL file containing the packet definitions for the Transmission Control Protocol, which in this case is a transport service for the IP protocol. In addition to extracting the protocol information the TCP protocol definition file assists in the process of identification of connec-

tions for the processing of states. In a typical set of files, there also would be a file UDP.pdl for the User Datagram Protocol (UDP) definitions. RPC.pdl (**1919**) is a PDL file file containing the packet definitions for Remote Procedure Calls.

NFS.pdl (**1921**) is a PDL file containing the packet definitions for the Network File System. Other PDL files would typically be included for all the protocols that might be encountered by monitor **300**.

Input to the compilation process **310** is the set of PDL files (e.g., the files of FIG. **19**) for all protocols of interest. Input to process **310** may also include layering information shown in FIG. **3** as datagram layer selections **338**. The layer selections information describes the layering of the protocols—what protocol(s) may be on top of any particular protocols. For example, IP may run over Ethernet, and also over many other types of packets. TCP may run on top of IP. UDP also may run on top of IP. When no layering information is explicitly included, it is inherent; the PDL files include the children protocols, and this provides the layering information.

The compiling process **310** is illustrated in FIG. **20**. The compiler loads the PDL source files into a scratch pad memory (step **2003**) and reviews the files for the correct syntax (parse step **2005**). Once completed, the compiler creates an intermediate file containing all the parse elements (step **2007**). The intermediate file in a format called "Compiled Protocol Language" (CPL). CPL instructions have a fixed layer format, and include all of the patterns, extractions, and states required for each layer and for the entire tree for a layer. The CPL file includes the number of protocols and the protocol definitions. A protocol definition for each protocol can include one or more of the protocol name, the protocol ID, a header section, a group identification section, sections for any particular layers, announcement sections, a payload section, a children section, and a states section. The CPL file is then run by the optimizer to create the final databases that will be used by monitor **300**. It would be clear to those in the art that alternate implementations of the compilation process **310** may include a different form of intermediate output, or no intermediate output at all, directly generating the final database(s).

After the parse elements have been created, the compiler builds the flow signature elements (step **2009**). This creates the extraction operations in CPL that are required at each level for each PDL module for the building of the flow signature (and hash key) and for links between layers (**2009**).

With the flow signature operations complete, the PDL compiler creates (step **2011**) the operations required to extract the payload elements from each PDL module. These payload elements are used by states in other PDL modules at higher layers in the processing.

The last pass is to create the state operations required by each PDL module. The state operations are complied from the PDL files and created in CPL form for later use (**2013**).

The CPL file is now run through an optimizer that generates the final databases used by monitor **300**.

PROTOCOL DEFINITION LANGUAGE (PDL) REFERENCE GUIDE (VERSION A0.02)

Included herein is this reference guide (the "guide") for the page description language (PDL) which, in one aspect of the invention, permits the automatic generation of the databases used by the parser and analyzer sub-systems, and also allows for including new and modified protocols and applications to the capability of the monitor.

1. INTRODUCTION

The inventive protocol Definition Language (PDL) is a special purpose language used to describe network protocols and all the fields within the protocol headers. Within this guide, protocol descriptions (PDL files) are referred to as PDL or rules when there in no risk of confusion with other types of descriptions.

PDL uses both form and organization similar to the data structure definition part of the C programming language and the PERL scripting language. Since PDL was derived from a language used to decode network packet contact, the authors have mixed the language format with the requirements of packet decoding. This results in an expressive language that is very familiar and comfortable for describing packet content and the details required representing a flow.

1.1 Summary

The PDL is a non-procedural Forth Generation language (4GL). This means is describes what needs to be done without describing how to do it. The details of how are hidden in the compiler and the Compiled Protocol Layout (CPL) optimization utility.

In addition, it is used to describe network flows by defining which fields are the address fields, which are the protocol type fields, etc.

Once a PDL file is written, it is compiled using the Netscope compiler (nsc), which produces the MeterFlow database (MeterFlow.db) and the Netscope database (Netscope.db). The MeterFlow database contains the flow definitions and the Netscope database contains the protocol header definitions.

These databases are used by programs like: mfkeys, which produces flow keys (also called flow signatures); mfcpl, which produces flow definitions in CPL format; mfpkts which produces sample packets of all known protocols; and netscope, which decodes Sniffer™ and tcpdump files.

1.2 Guide Conventions

The following conventions will be used throughout this guide:

Small courier typeface indicates C code examples or function names. Functions are written with parentheses after them [function ()], variables are written just as their names [variables], and structure names are written prefixed with "struct" [struct packet].

Italics indicate a filename (for instance, mworks/base/h/base.h). Filenames will usually be written relative to the root directory of the distribution.

Constants are expressed in decimal, unless written "0x . . . ", the C language notation for hexadecimal numbers.

Note that any contents on any line in a PDL file following two hyphen (--) are ignored by the compiler. That is, they are comments.

2. PROGRAM STRUCTURE

A MeterFlow PDL decodes and flow set is a non-empty sequence of statements.

There are four basic types of statements or definitions available in MeterFlow PDL:

FIELD,

GROUP,

PROTOCOL and

FLOW.

2.1 Field Definitions

The FIELD definition is used to define a specific string of bits or bytes in the packet. The FIELD definition has the following format:

Name FIELD

SYNTAX Type [{Enums }]

DISPLAY-HINT "FormatString"

LENGTH "Expression"

FLAGS FieldFlags

ENCAP FieldName [, FieldName2]

LOOKUP LookupType [Filename]

ENCODING EncodingType

DEFAULT "value"

DESCRIPTION "Description"

Where only the FIELD and SYNTAX lines are required. All the other lines are attribute lines, which define special characteristics about the FIELD. Attribute lines are optional and may appear in any order. Each of the attribute lines are described in detail below:

2.1.1 SYNTAX Type [{Enums}]

This attribute defines the type and, if the type is an INT, BYTESTRING, BITSTRING, or SNMPSEQUENCE type, the enumerated values for the FIELD. The currently defined types are:

INT(numBits)	Integer that is numBits bits long.
UNSIGNED INT(numBits)	Unsigned integer that is numBits bits long.
BYTESTRING(numBytes)	String that is numBytes bytes long.
BYTESTRING(R1 . . . R2)	String that ranges in size from R1 to R2 bytes.
BITSTRING(numBits)	String that is numBits bits long.
LSTRING(lenBytes)	String with lenBytes header.
NSTRING	Null terminated string.
DNSSTRING	DNS encoded string.
SNMPOID	SNMP Object Identifier.
SNMPSEQUENCE	SNMP Sequence.
SNMPTIMETICKS	SNMP TimeTicks.
COMBO field1 field2	Combination pseudo field.

2.1.2 DISPLAY-HINT "FormatString"

This attribute is for specifying how the value of the FIELD is displayed. The currently supported formats are:

Numx	Print as a num byte hexidecimal number.
Numd	Print as a num byte decimal number.
Numo	Print as a num byte octal number.
Numb	Print as a num byte binary number.
Numa	Print num bytes in ASCII format.
Text	Print as ASCII text.
HexDump	Print in hexdump format.

2.1.3 LENGTH "Expression"

This attribute defines an expression for determining the FIELD's length. Expressions are arithmetic and can refer to the value of other FIELD's in the packet by adding a $ to the referenced field's name. For example, "($tcpHeaderLen*4)-20" is a valid expression if tcpHeaderLen is another field defined for the current packet.

2.1.4 FLAGS FieldFlags

The attribute defines some special flags for a FIELD. The currently supported FieldFlags are:

SAMELAYER	Display field on the same layer as the previous field.
NOLABEL	Don't display the field name with the value.
NOSHOW	Decode the field but don't display it.
SWAPPED	The integer value is swapped.

2.1.5 ENCAP FieldName [, FieldName2]

This attribute defines how one packet is encapsulated inside another. Which packet is determined by the value of the FieldName field. If no packet is found using FieldName then FieldName2 is tried.

2.1.6 LOOKUP LookupType [Filename]

This attribute defines how to lookup the name for a particular FIELD value. The currently supported LookupTypes are:

SERVICE	Use getservbyport( ).
HOSTNAME	Use gethostbyaddr( ).
MACADDRESS	Use $METERFLOW/conf/mac2ip.cf.
FILE file	Use file to lookup value.

2.1.7 ENCODING EncodingType

This attribute defines how a FIELD is encoded. Currently, the only supported EncodingType is BER (for Basic Encoding Rules defined by ASN.1).

2.1.8 DEFAULT "value"

This attribute defines the default value to be used for this field when generating sample packets of this protocol.

2.1.9 DESCRIPTION "Description"

This attribute defines the description of the FIELD. It is used for informational purposes only.

2.2 Group Definitions

The GROUP definition is used to tie several related FIELDs together. The GROUP definition has the following format:

Name GROUP

LENGTH "Expression"

OPTIONAL "Condition"

SUMMARIZE "Condition":"FormatString" ["Condition": "FormatString" . . .]

DESCRIPTION "Description"

::={Name=FieldOrGroup [, Name=FieldorGroup . . .]}

Where only the GROUP and ::=lines are required. All the other lines are attribute lines, which define special characteristics for the GROUP. Attribute lines are optional and may appear in any order. Each attribute line is described in detail below:

2.2.1 LENGTH "Expression"

This attribute defines an expression for determining the GROUP's length. Expressions are arithmetic and can refer to the value of other FIELD's in the packet by adding a $ to the referenced field's name. For example,

"($tcpHeaderLen*4)-20" is a valid expression if tcpHeaderLen is another field defined for the current packet.

2.2.2 OPTIONAL "Condition"

This attribute defines a condition for determining whether a GROUP is present or not. Valid conditions are defined in the Conditions section below.

2.2.3 SUMMARIZE "Condition":"FormatString" ["Condition":"FormatString" . . .]

This attribute defines how a GROUP will be displayed in Detail mode. A different format (FormatString) can be specified for each condition (Condition). Valid conditions are defined in the Conditions section below. Any FIELD's value can be referenced within the FormatString by proceeding the FIELD's name with a $. In addition to FIELD names there are several other special $ keywords:

$LAYER	Displays the current protocol layer.
$GROUP	Displays the entire GROUP as a table.
$LABEL	Displays the GROUP label.
$field	Displays the field value (use enumerated name if available).
$:field	Displays the field value (in raw format).

2.2.4 DESCRIPITION "Description"

This attribute defines the description of the GROUP. It is used for informational purposes only.

2.2.5 ::={Name=FieldOrGroup [, Name=FieldOrGroup . . .]}

This defines the order of the fields and subgroups within the GROUP.

2.3 PROTOCOL Definitions

The PROTOCOL definition is used to define the order of the FIELDs and GROUPs within the protocol header. The PROTOCOL definition has the following format:

Name PROTOCOL
SUMMARIZE "Condition":"FormatString"] "Condition":"FormatString" . . .]
DESCRIPTION "Description"
REFERENCE "Reference"
::={Name=FieldOrGroup [, Name=FieldOrGroup . . .]}

Where only the PROTOCOL and ::=lines are required. All the other lines are attribute lines, which define special characteristics for the PROTOCOL. Attribute lines are optional and may appear in any order. Each attribute line is described in detail below:

2.3.1 SUMMARIZE "Condition":"FormatString" ["Condition":"FormatString". . .]

This attribute defines how a PROTOCOL will be displayed in Summary mode. A different format (FormatString) can be specified for each condition (Condition). Valid conditions are defined in the Conditions section below. Any FIELD's value can be referenced within the FormatString by proceeding the FIELD's name with a $. In addition to FIELD names there are several other special $ keywords:

$LAYER	Displays the current protocol layer.
$VARBIND	Displays the entire SNMP VarBind list.
$field	Displays the field value (use enumerated name if available).
$:field	Displays the field value (in raw format).
$#field	Counts all occurrences of field.
$*field	Lists all occurrences of field.

2.3.2 DESCRIPTION "Description"

This attribute defines the description of the PROTOCOL. It is used for informational purposes only.

2.3.3 REFERENCE "Reference"

This attribute defines the reference material used to determine the protocol format. It is used for informational purposes only.

2.3.4 ::={Name=FieldOrGroup [, Name=FieldOrGroup . . .]}

This defines the order of the FIELDs and GROUPs within the PROTOCOL.

2.4 FLOW Definitions

The FLOW definition is used to define a network flow by describing where the address, protocol type, and port numbers are in a packet. The FLOW definition has the following format:

Name FLOW
HEADER {Option [, Option . . .]}
DLC-LAYER {Option [, Option . . .]}
NET-LAYER {Option [, Option . . .]}
CONNECTION {Option [, Option . . .]}
PAYLOAD {Option [, Option . . .]}
CHILDREN {Option [, Option . . .]}
STATE-BASED
STATES "Definitions"

Where only the FLOW line is required. All the other lines are attribute lines, which define special characteristics for the FLOW. Attribute lines are optional and may appear in any order. However, at least one attribute line must be present. Each attribute line is described in detail below:

2.4.1 HEADER {Option [, Option . . .]}

This attribute is used to describe the length of the protocol header. The currently supported Options are:

LENGTH = number	Header is a fixed length of size number.
LENGTH = field	Header is variable length determined by value of field.
IN-WORDS	The units of the header length are in 32-bit words rather than bytes.

2.4.2 DLC-LAYER {Option [, Option . . .]}

If the protocol is a data link layer protocol, this attribute describes it. The currently supported Options are:

DESTINATION = field	Indicates which field is the DLC destination address.
SOURCE = field	Indicates which field is the DLC source address.
PROTOCOL	Indicates this is a data link layer protocol.
TUNNELING	Indicates this is a tunneling protocol.

2.4.3 NET-LAYER {Option [, Option . . .]}

If the protocol is a network layer protocol, then this attribute describes it. The currently supported Options are:

DESTINATION = field	Indicates which field is the network destination address.
SOURCE = field	Indicates which field is the network source address.
TUNNELING	Indicates this is a tunneling protocol.
FRAGMENTATION = type	Indicates this protocol supports fragmentation. There are currently two fragmentation types: IPV4 and IPV6.

2.4.4 CONNECTION {Option [, Option . . .]}

If the protocol is a connection-oriented protocol, then this attribute describes how connections are established and torn down. The currently supported Options are:

IDENTIFIER = field	Indicates the connection identifier field.
CONNECT-START = "flag"	Indicates when a connection is being initiated.
CONNECT-COMPLETE = "flag"	Indicates when a connection has been established.
DISCONNECT-START = "flag"	Indicates when a connection is being torn down.
DISCONNECT-COMPLETE = "flag"	Indicates when a connection has been torn down.
INHERITED	Indicates this is a connection-oriented protocol but the parent protocol is where the connection is established.

2.4.5 PAYLOAD {Option [, Option . . .]}

This attribute describes how much of the payload from a packet of this type should be stored for later use during analysis. The currently supported Options are:

INCLUDE-HEADER	Indicates that the protocol header should be included.
LENGTH = number	Indicates how many bytes of the payload should be stored.
DATA = field	Indicates which field contains the payload.

2.4.6 CHILDREN {Option [, Option . . .]}

This attribute describes how children protocols are determined. The currently supported Options are:

DESTINATION = field	Indicates which field is the destination port.
SOURCE = field	Indicates which field is the source port.
LLCCHECK = flow	Indicates that if the DESTINATION field is less than 0 x 05DC then use flow instead of the current flow definition.

2.4.7 STATE-BASED

This attribute indicates that the flow is a state-based flow.

2.4.8 STATES "Definitions"

This attribute describes how children flows of this protocol are determined using states. See the State Definitions section below for how these states are defined.

2.5 CONDITIONS

Conditions are used with the OPTIONAL and SUMMARIZE attributes and may consist of the following:

Value1 == Value2	Value1 equals Value2. Works with string values.
Value1 != Value2	Value1 does not equal Value2. Works with string values.
Value1 <= Value2	Value1 is less than or equal to Value2.
Value1 >= Value2	Value1 is greater than or equal to Value2.
Value1 < Value2	Value1 is less than Value2.
Value1 > Value2	Value1 is greater than Value2.
Field m/regex/	Field matches the regular expression regex.

Where Value1 and Value2 can be either FIELD references (field names preceded by a $) or constant values. Note that compound conditional statements (using AND and OR) are not currently supported.

2.6 STATE DEFINITIONS

Many applications running over data networks utilize complex methods of classifying traffic through the use of multiple states. State definitions are used for managing and maintaining learned states from traffic derived from the network.

The basic format of a state definition is:

StateName: Operand Parameters [Operand Parameters . . .]

The various states of a particular flow are described using the following operands:

2.6.1 CHECKCONNECT, Operand

Checks for connection. Once connected executes operand.

2.6.2 GOTO State

Goes to state, using the current packet.

2.6.3 NEXT State

Goes to state, using the next packet.

2.6.4 DEFAULT Operand

Executes operand when all other operands fail.

2.6.5 CHILD Protocol

Jump to child protocol and perform state-based processing (if any) in the child.

2.6.6 WAIT Numpackets, Operand1, Operand2

Waits the specified number of packets. Executes operand1 when the specified number of packets have been received. Executes operand2 when a packet is received but it is less than the number of specified packets.

2.6.7 MATCH 'String' Weight Offset LF-offset Range LF-range, Operand

Searches for a string in the packet, executes operand if found.

2.6.8 CONSTANT Number Offset Range, Operand

Checks for a constant in a packet, executes operand if found.

2.6.9 EXTRACTIP Offset Destination, Operand

Extracts an IP address from the packet and then executes operand.

2.6.10 EXTRACTPORT Offset Destination, Operand

Extracts a port number from the packet and then executes operand.

2.6.11 CREATEREDIRECTEDFLOW, Operand

Creates a redirected flow and then executes operand.

3. EXAMPLE PDL RULES

The following section contains several examples of PDL Rule files.

3.1 Ethernet

The following is an example of the PDL for Ethernet:

```
MacAddress      FIELD
                SYNTAX          BYTESTRING (6)
                DISPLAY-HINT    "1x:"
                LOOKUP          MACADDRESS
                DESCRIPTION
                        "MAC layer physical address"
etherType       FIELD
                SYNTAX          INT(16)
                DISPLAY-HINT    "1x:"
                LOOKUP          FILE "EtherType.cf"
                DESCRIPTION
                        "Ethernet type field"
etherData       FIELD
                SYNTAX          BYTESTRING(46..1500)
                ENCAP             etherType
                DISPLAY-HINT    "HexDump"
                DESCRIPTION
                        "Ethernet data"
ethernet        PROTOCOL
                DESCRIPTION
                  "Protocol format for an Ethernet frame"
                REFERENCE       "RFC 894"
::= { MacDest=macAddress, MacSrc=macAddress, EtherType=etherType,
   Data=etherData }
ethernet        FLOW
                HEADER { LENGTH=14 }
                DLC-LAYER {
                  SOURCE=MacSrc,
                  DESTINATION=MacDest,
                  TUNNELING,
                  PROTOCOL
                }
                CHILDREN { DESTINATION=EtherType,
                LLC-CHECK=llc }
```

3.2 IP Version 4

Here is an example of the PDL for the IP protocol:

```
ipAddress       FIELD
                SYNTAX          BYTESTRING(4)
                DISPLAY-HINT    "1d."
                LOOKUP          HOSTNAME
                DESCRIPTION
  "IP address"
ipversion       FIELD
                SYNTAX          INT(4)
                DEFAULT         "4"
ipHeaderLength      FIELD
                    SYNTAX INT(4)
ipTypeOfService     FIELD
                    SYNTAX          BITSTRING(8) { minCost(1),
                                    maxReliability(2),
                                    maxThruput(3),
                                    minDelay(4) }
ipLength        FIELD
                SYNTAX UNSIGNED INT(16)
ipFlags         FIELD
                SYNTAX          BITSTRING(3) { moreFrags(0),
                                dontFrag(1) }
IpFragmentOffset    FIELD
                    SYNTAX          INT(13)
ipProtocol      FIELD
                SYNTAX INT(8)
                LOOKUP FILE "IpProtocol.cf"
```

-continued

```
ipData FIELD
                SYNTAX          BYTESTRING(0..1500)
                ENCAP           ipProtocol
                DISPLAY-HINT    "HexDump"
ip          PROTOCOL
            SUMMARIZE
            "$FragmentOffset != 0"
              "IpFragment ID=$Identification Offset=$Fragmentoffset"
            ""Default" :
              "IP Protocol=$Protocol"
            DESCRIPTION
              "Protocol format for the Internet Protocol"
            REFERENCE   "RFC 791"
::= { Version=ipVersion, HeaderLength=ipHeaderLength,
            TypeOfService=ipTypeOfService, Length=ipLength,
            Identification=UInt16, IpFlags=ipFlags,
            FragmentOffset=ipFragmentOffset, TimeToLive=Int8,
            Protocol=ipProtocol, Checksum=ByteStr2,
            IpSrc=ipAddress, IpDest=ipAddress, Options=ipOptions,
            Fragment=ipFragment, Data=ipData }
ip          FLOW
            HEADER { LENGTH=HeaderLength, IN-WORDS }
            NET-LAYER {
                SOURCE=IpSrc,
                DESTINATION=IpDest,
                FRAGMENTATION=IPV4,
                TUNNELING
            }
            CHILDREN { DESTINATION=Protocol }
ipFragData      FIELD
                SYNTAX          BYTESTRING(1..1500)
                LENGTH          "ipLength - ipHeaderLength * 4"
                DISPLAY-HINT    "HexDump"
ipFragment      GROUP
                OPTIONAL        "$FragmentOffset != 0"
::= { Data=ipFragData }
ipOptionCode    FIELD
                SYNTAX INT(8) { ipRR(0x07), ipTimestamp(0x44),
                                        ipLSRR(0x83),
                                        ipSSRR(0x89) }
                DESCRIPTION
                  "IP option code"
ipOptionLength      FIELD
                    SYNTAX UNSIGNED INT(8)
                    DESCRIPTION
                      "Length of IP option"
ipOptionData    FIELD
                    SYNTAX          BYTESTRING(0..1500)
                    ENCAP             ipOptionCode
                    DISPLAY-HINT    "HexDump"
ipOptions       GROUP
                LENGTH          "(ipHeaderLength * 4) - 20"
::= { Code=ipOptionCode, Length=ipOptionLength, Pointer=UInt8,
            Data=ipOptionData }
```

3.3 TCP

Here is an example of the PDL for the TCP protocol:

```
tcpPort FIELD
        SYNTAX UNSIGNED INT(16)
        LOOKUP FILE "TcpPort.cf"
tcpHeaderLen FIELD
        SYNTAX INT(4)
tcpFlags FIELD
        SYNTAX BITSTRING(12) { fin(0), syn(1), rst(2), psh(3),
                                        ack(4), urg(5) }
tcpData FIELD
        SYNTAX BYTESTRING(0..1564)
        LENGTH " ($ipLength- ($jpHeaderLength*4)) -
        ($tcpHeaderLen*4) "
        ENCAP           tcpport
        DISPLAY-HINT    "HexDump"
tcp     PROTOCOL
```

-continued

```
        SUMMARIZE
          "Default"
            "TCP ACK=$Ack WIN=$WindowSize"
        DESCRIPTION
          "Protocol format for the Transmission Control Protocol"
        REFERENCE       "RFC 793"
::= { SrcPort=tcpPort, DestPort=tcpPort, SequenceNum=UInt32,
        Ack=UInt32, HeaderLength=tcpHeaderLen, TcpFlags=tcpFlags,
        WindowSize=UInt16, Checksum=ByteStr2,
        UrgentPointer=UInt16, Options=tcpOptions, Data=tcpData }
tcp     FLOW
        HEADER { LENGTH=HeaderLength, IN-WORDS }
        CONNECTION {
              IDENTIFIER=SequenceNum,
              CONNECT-START="TcpFlags:1",
              CONNECT-COMPLETE="TcpFlags:4",
              DISCONNECT-START="TcpFlags:0",
              DISCONNECT-COMPLETE="TcpFlags:4"
         }
        PAYLOAD { INCLUDE-HEADER }
        CHILDREN { DESTINATION=DestPort, SOURCE=SrcPort }
tcpOptionKind FIELD
                SYNTAX UNSIGNED INT(8) { tcpOptEnd(0),
                tcpNop(1),
tcpMSS(2), tcpWscale(3), tcpTimestamp(4) }
                DESCRIPTION
                        "Type of TCP option"
tcpOptionDataFIELD
                SYNTAX          BYTESTRING(0..1500)
                ENCAP           tcpOptionKind
                FLAGS           SAMELAYER
                DISPLAY-HINT    "HexDump"
tcpOptions      GROUP
                LENGTH          "($tcpHeaderLen * 4) - 20"
::= { Option=tcpOptionKind, OptionLength=UInt8,
        OptionData=tcpOptionData }
tcpMSS PROTOCOL
::= { MaxSegmentSize=UInt16 }
```

3.4 HTTP (With State)

Here is an example of the PDL for the HTTP protocol:

```
httpData FIELD
    SYNTAX   BYTESTRING(1..1500)
    LENGTH     "($ipLength - ($ipHeaderLength * 4)) -
               ($tcpHeaderLen * 4) "
DISPLAY-HINT     "Text"
FLAGS              NOLABEL
http            PROTOCOL
                SUMMARIZE
                    "$httpData m/^GET|^HTTP|^HEAD|^POST/" :
                        "HTTP $httpData"
                    "$httpData m/^[Dd]ate|^[Ss]erver|^[Ll]ast-
                    [Mm]odified/" :
                        "HTTP $httpData"
                    "$httpData m/^[Cc]ontent-/" :
                        "HTTP $httpData"
                    $httpData m/^<HTML>/" :
                        "HTTP [HTML document]"
                    $httpData m/^GIF/" :
                        "HTTP [GIF image]"
                    "Default" :
                        "HTTP [Data]"
                DESCRIPTION
                    "Protocol format for HTTP."
::= { Data=httpData }
http   FLOW
HEADER { LENGTH=0 }
CONNECTION { INHERITED }
PAYLOAD { INCLUDE-HEADER, DATA=Data, LENGTH=256 }
STATES
                    "S0: CHECKCONNECT, GOTO S1
                        DEFAULT NEXT S0
```

-continued

```
                S1: WAIT 2, GOTO S2, NEXT S1
                    DEFAULT NEXT S0
                S2: MATCH
                    '\n\r\n'            900 0 0 255 0, NEXT S3
                    '\n\n'              900 0 0 255 0, NEXT S3
                    'POST /tds?'         50 0 0 127 1,
                                        CHILD sybaseWebsql
                    '.hts HTTP/1.0'      50 4 0 127 1,
                                        CHILD sybaseJdbc
                    'jdbc:sybase:Tds'    50 4 0 127 1,
                                        CHILD sybaseTds
                    'PCN-The Poin'      500 4 1 255 0,
                                        CHILD pointcast
                    't: BW-C-'          100 4 1 255 0,
                                        CHILD backweb
                    DEFAULT NEXT S3
                s3: MATCH
                    '\n\r\n'             50 0 0    0 0, NEXT S3
                    '\n\n'               50 0 0    0 0, NEXT S3
                    'Content-Type:'     800 0 0 255 0,
                                        CHILD mime
                    'PCN-The Poin'      500 4 1 255 0,
                                        CHILD pointcast
                    't: BW-C-'          100 4 1 255 0,
                                        CHILD backweb
                    DEFAULT NEXT S0"
sybaseWebsql        FLOW
                    STATE-BASED
sybaseJdbc          FLOW
                    STATE-BASED
sybaseTds           FLOW
                    STATE-BASED
pointcast           FLOW
                    STATE-BASED
backweb             FLOW
                    STATE-BASED
mime                FLOW
                    STATE-BASED
                    STATES
"  S0:   MATCH
'application'       900 0 0    1 0,
                    CHILD mimeApplication
    'audio'         900 0 0    1 0,
                    CHILD mimeAudio
    'image'          50 0 0    1 0,
                    CHILD mimeImage
    'text'           50 0 0    1 0,
                    CHILD mimeText
    'video'          50 0 0    1 0,
                    CHILD mimeVideo
    'x-world'       500 4 1 255 0,
                    CHILD mimeXworld
DEFAULT GOTO S0"
mimApplication      FLOW
                    STATE-BASED
mimeAudio       FLOW
                STATE-BASED
                STATES
                 "S0: MATCH
                     'basic'            100 0 0 1 0,
                                        CHILD pdBasicAudio
                     'midi'             100 0 0 1 0,
                                        CHILD pdMidi
                     'mpeg'             100 0 0 1 0,
                                        CHILD pdMpeg2Audio
                     'vnd.rn-realaudio' 100 0 0 1 0,
                                        CHILD pdRealAudio
                     'wav'              100 0 0 1 0,
                                        CHILD pdWav
                     'x-aiff'           100 0 0 1 0,
                                        CHILD pdAiff
                     'x-midi'           100 0 0 1 0,
                                        CHILD pdMidi
                     'x-mpeg'           100 0 0 1 0,
                                        CHILD pdMpeg2Audio
                     'x-mpgurl'         100 0 0 1 0,
                                        CHILD pdMpeg3Audio
```

-continued

```
                'x-pn-realaudio'    100 0 0 1 0,
                                    CHILD pdRealAudio
                'x-wav'             100 0 0 1 0,
                                    CHILD pdWav
            DEFAULT GOTO S0"
mimeImage   FLOW
            STATE-BASED
mimeText    FLOW
            STATE-BASED
mimeVideo   FLOW
            STATE-BASED
mimeXworld  FLOW
            STATE-BASED
pdBasicAudio    FLOW
            STATE-BASED
pdMidi      FLOW
            STATE-BASED
pdMpeg2Audio    FLOW
            STATE-BASED
pdMpeg3Audio    FLOW
            STATE-BASED
pdRealAudio     FLOW
            STATE-BASED
pdWav       FLOW
            STATE-BASED
pdAiff      FLOW
            STATE-BASED
```

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 protoçols, 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.

APPENDIX: SOME PDL FILES

The following pages include some PDL files as examples. Included herein are the PDL contents of the following files. A reference to PDL is also included herein. Note that any contents on any line following two hyphen (--) are ignored by the compiler. That is, they are comments.

common.pdl;
flows.pdl;
virtual.pdl;
ethernet.pdl;
IEEE8032.pdl and IEEE8033.pdl (ethertype files);
IP.pdl;
TCP.pdl and UDP.pdl;
RPC.pdl;
NFS.pdl; and
HTTP.pdl.

```
--
--  Common.pdl - Common protocol definitions
--
--  Description:
--      This file contains some field definitions for commonly used fields
--      in various network protocols.
--
--  Copyright:
--      Copyright (c) 1996-1999 Apptitude, Inc.
--       (formerly Technically Elite, Inc.)
--      All rights reserved.
--
--  RCS:
--      $Id: Common.pdl,v 1.7 1999/04/13 15:47:56 skip Exp $
--

Int4    FIELD
        SYNTAX INT(4)
Int8    FIELD
        SYNTAX INT(8)
Int16   FIELD
        SYNTAX INT(16)
Int24   FIELD
        SYNTAX INT(24)
Int32   FIELD
        SYNTAX INT(32)
Int64   FIELD
        SYNTAX INT(64)
UInt8   FIELD
        SYNTAX UNSIGNED INT(8)
UInt16  FIELD
        SYNTAX UNSIGNED INT(16)
UInt24  FIELD
        SYNTAX UNSIGNED INT(24)
UInt32  FIELD
```

-continued

```
        SYNTAX UNSIGNED INT(32)
UInt64  FIELD
        SYNTAX UNSIGNED INT(64)
SInt16  FIELD
        SYNTAX INT(16)
        FLAGS SWAPPED
SUInt16     FIELD
        SYNTAX UNSIGNED INT(16)
        FLAGS SWAPPED
SInt32  FIELD
        SYNTAX INT(32)
        FLAGS SWAPPED
ByteStr1        FIELD
                SYNTAX BYTESTRING(1)
ByteStr2        FIELD
                SYNTAX BYTESTRING(2)
ByteStr4        FIELD
                SYNTAX BYTESTRING(4)
Pad1            FIELD
                SYNTAX BYTESTRING(1)
                FLAGS NOSHOW
Pad2            FIELD
                SYNTAX BYTESTRING(2)
                FLAGS NOSHOW
Pad3            FIELD
                SYNTAX BYTESTRING(3)
                FLAGS NOSHOW
Pad4            FIELD
                SYNTAX BYTESTRING(4)
                FLAGS NOSHOW
Pad5            FIELD
                SYNTAX BYTESTRING(5)
                FLAGS NOSHOW
macAddress      FIELD
    SYNTAX          BYTESTRING(6)
    DISPLAY-HINT    "1x:"
    LOOKUP          MACADDRESS
    DESCRIPTION
        "MAC layer physical address"
ipAddress       FIELD
    SYNTAX          BYTESTRING(4)
    DISPLAY-HINT    "1d."
    LOOKUP          HOSTNAME
    DESCRIPTION
        "IP address"
ipv6Address     FIELD
    SYNTAX          BYTESTRING(16)
    DISPLAY-HINT    "1d."
    DESCRIPTION
        "IPV6 address"
-----------------------------------------------------------------
--
--  Flows.pdl - General FLOW definitions
--
--  Description:
--      This file contains general flow definitions.
--
--  Copyright:
--      Copyright (c) 1998-1999 Apptitude, Inc.
--       (fomerly Technically Elite, Inc.)
--      All rights reserved.
--
--  RCS:
--      $Id: Flows.pdl,v 1.12 1999/04/13 15:47:57 skip Exp $
--
-----------------------------------------------------------------
chaosnet    FLOW
spanningTree FLOW
sna         FLOW
oracleTNS FLOW
        PAYLOAD { INCLUDE-HEADER, LENGTH=256 }
ciscoOUI FLOW
-----------------------------------------------------------------
--  IP Protocols
-----------------------------------------------------------------
igmp        FLOW
GGP         FLOW
ST          FLOW
UCL         FLOW
```

-continued

egp	FLOW
igp	FLOW
BBN-RCC-MON	FLOW
NVP2	FLOW
PUP	FLOW
ARGUS	FLOW
EMCON	FLOW
XNET	FLOW
MUX	FLOW
DCN-MEAS	FLOW
HMP	FLOW
PRM	FLOW
TRUNK1	FLOW
TRUNK2	FLOW
LEAF1	FLOW
LEAF2	FLOW
RDP	FLOW
IRTP	FLOW
ISO-TP4	FLOW
NETBLT	FLOW
MFE-NSP	FLOW
MERIT-INP	FLOW
SEP	FLOW
PC3	FLOW
IDPR	FLOW
XTP	FLOW
DDP	FLOW
IDPR-CMTP	FLOW
TPPlus	FLOW
IL	FLOW
SIP	FLOW
SDRP	FLOW
SIP-SR	FLOW
SIP-FRAG	FLOW
IDRP	FLOW
RSVP	FLOW
MHRP	FLOW
BNA	FLOW
SIPP-ESP	FLOW
SIPP-AH	FLOW
INLSP	FLOW
SWIPE	FLOW
NHRP	FLOW
CFTP	FLOW
SAT-EXPAK	FLOW
KRYPTOLAN	FLOW
RVD	FLOW
IPPC	FLOW
SAT-MON	FLOW
VISA	FLOW
IPCV	FLOW
CPNX	FLOW
CPHB	FLOW
WSN	FLOW
PVP	FLOW
BR-SAT-MON	FLOW
SUN-ND	FLOW
WB-MON	FLOW
WB-EXPAK	FLOW
ISO-IP	FLOW
VMTP	FLOW
SECURE-VMTP	FLOW
TTP	FLOW
NSFNET-IGP	FLOW
DGP	FLOW
TCF	FLOW
IGRP	FLOW
OSPFIGP	FLOW
Sprite-RPC	FLOW
LARP	FLOW
MTP	FLOW
AX25	FLOW
IPIP	FLOW
MICP	FLOW
SCC-SP	FLOW
ETHERIP	FLOW
encap	FLOW
GMTP	FLOW

-continued

```
--  UDP Protocols
compressnet   FLOW
rje         FLOW
echo        FLOW
discard         FLOW
systat      FLOW
daytime         FLOW
qotd        FLOW
msp         FLOW
chargen         FLOW
biff        FLOW
who         FLOW
syslog      FLOW
loadav      FLOW
notify      FLOW
acmaint_dbd     FLOW
acmaint_transd        FLOW
puparp      FLOW
applix      FLOW
ock         FLOW
-------------------------------------------------
--  TCP Protocols
-------------------------------------------------
tcpmux      FLOW
telnet      FLOW
            CONNECTION { INHERITED }
privMail        FLOW
nsw-fe      FLOW
msg-icp         FLOW
msg-auth        FLOW
dsp         FLOW
privPrint       FLOW
time        FLOW
rap         FLOW
rip         FLOW
graphics        FLOW
nameserver      FLOW
nicname         FLOW
mpm-flags       FLOW
mpm         FLOW
mpm-snd         FLOW
ni-ftp      FLOW
auditd      FLOW
finger      FLOW
re-mail-ck      FLOW
la-maint        FLOW
xns-time        FLOW
xns-ch      FLOW
isi-gl      FLOW
xns-auth        FLOW
privTerm        FLOW
xns-mail        FLOW
privFile        FLOW
ni-mail         FLOW
acas        FLOW
covia       FLOW
tacacs-ds       FLOW
sqlnet      FLOW
gopher      FLOW
netrjs-1        FLOW
netrjs-2        FLOW
netrjs-3        FLOW
netrjs-4        FLOW
privDial        FLOW
deos        FLOW
privRJE         FLOW
vettcp      FLOW
hosts2-ns       FLOW
xfer        FLOW
ctf             FLOW
mit-ml-dev      FLOW
mfcobol         FLOW
kerberos        FLOW
su-mit-tg       FLOW
dnsix       FLOW
mit-dov         FLOW
npp         FLOW
dcp         FLOW
objcall         FLOW
```

-continued

supdup	FLOW
dixie	FLOW
swift-rvf	FLOW
tacnews	FLOW
metagram	FLOW
newacct	FLOW
hostname	FLOW
iso-tsap	FLOW
gppitnp	FLOW
csnet-ns	FLOW
threeCom-tsmux	FLOW
rtelnet	FLOW
snagas	FLOW
mcidas	FLOW
auth	FLOW
audionews	FLOW
sftp	FLOW
ansanotify	FLOW
uucp-path	FLOW
sqlserv	FLOW
cfdptkt	FLOW
erpc	FLOW
smakynet	FLOW
ntp	FLOW
ansatrader	FLOW
locus-map	FLOW
unitary	FLOW
locus-con	FLOW
gss-xlicen	FLOW
pwdgen	FLOW
cisco-fna	FLOW
cisco-tna	FLOW
cisco-sys	FLOW
statsrv	FLOW
ingres-net	FLOW
loc-srv	FLOW
profile	FLOW
emfis-data	FLOW
emfis-cntl	FLOW
bl-idm	FLOW
imap2	FLOW
news	FLOW
uaac	FLOW
iso-tp0	FLOW
iso-ip	FLOW
cronus	FLOW
aed-512	FLOW
sql-net	FLOW
hems	FLOW
bftp	FLOW
sgmp	FLOW
netsc-prod	FLOW
netsc-dev	FLOW
sqlsrv	FLOW
knet-cmp	FLOW
pcmail-srv	FLOW
nss-routing	FLOW
sgmp-traps	FLOW
cmip-man	FLOW
cmip-agent	FLOW
xns-courier	FLOW
s-net	FLOW
namp	FLOW
rsvd	FLOW
send	FLOW
print-srv	FLOW
multiplex	FLOW
cl-1	FLOW
xyplex-mux	FLOW
mailq	FLOW
vmnet	FLOW
genrad-mux	FLOW
xdmcp	FLOW
nextstep	FLOW
bgp	FLOW
ris	FLOW
unify	FLOW
audit	FLOW
ocbinder	FLOW

-continued

ocserver	FLOW
remote-kis	FLOW
kis	FLOW
aci	FLOW
mumps	FLOW
qft	FLOW
gacp	FLOW
prospero	FLOW
osu-nms	FLOW
srmp	FLOW
irc	FLOW
dn6-nlm-aud	FLOW
dn6-smm-red	FLOW
dls	FLOW
dls-mon	FLOW
smux	FLOW
src	FLOW
at-rtmp	FLOW
at-nbp	FLOW
at-3	FLOW
at-echo	FLOW
at-5	FLOW
at-zis	FLOW
at-7	FLOW
at-8	FLOW
tam	FLOW
z39-50	FLOW
anet	FLOW
vmpwscs	FLOW
softpc	FLOW
atls	FLOW
dbase	FLOW
mpp	FLOW
uarps	FLOW
imap3	FLOW
fln-spx	FLOW
rsh-spx	FLOW
cdc	FLOW
sur-meas	FLOW
link	FLOW
dsp3270	FLOW
pdap	FLOW
pawserv	FLOW
zserv	FLOW
fatserv	FLOW
csi-sgwp	FLOW
clearcase	FLOW
ulistserv	FLOW
legent-1	FLOW
legent-2	FLOW
hassle	FLOW
nip	FLOW
tnETOS	FLOW
dsETOS	FLOW
is99c	FLOW
is99s	FLOW
hp-collector	FLOW
hp-managed-node	FLOW
hp-alarm-mgr	FLOW
arns	FLOW
ibm-app	FLOW
asa	FLOW
aurp	FLOW
unidata-ldm	FLOW
ldap	FLOW
uis	FLOW
synotics-relay	FLOW
synotics-broker	FLOW
dis	FLOW
embl-ndt	FLOW
netcp	FLOW
netware-ip	FLOW
mptn	FLOW
kryptolan	FLOW
work-sol	FLOW
ups	FLOW
genie	FLOW
decap	FLOW
nced	FLOW

-continued

```
ncld        FLOW
imsp        FLOW
timbuktu        FLOW
prm-sm      FLOW
prm-nm      FLOW
decladebug      FLOW
rmt         FLOW
synoptics-trap      FLOW
smsp        FLOW
infoseek        FLOW
bnet        FLOW
silverplatter       FLOW
onmux       FLOW
hyper-g         FLOW
ariell      FLOW
smpte       FLOW
ariel2      FLOW
ariel3      FLOW
opc-job-start       FLOW
opc-job-track       FLOW
icad-el         FLOW
smartsdp        FLOW
svrloc      FLOW
ocs_cmu         FLOW
ocs_amu         FLOW
utmpsd      FLOW
utmpcd      FLOW
iasd        FLOW
nnsp        FLOW
mobileip-agent      FLOW
mobilip-mn      FLOW
dna-cml         FLOW
comscm      FLOW
dsfgw       FLOW
dasp        FLOW
sgcp        FLOW
decvms-sysmgt       FLOW
cvc_hostd       FLOW
https       FLOW
            CONNECTION { INHERITED }
snpp        FLOW
microsoft-ds        FLOW
ddm-rdb         FLOW
ddm-dfm         FLOW
ddm-byte        FLOW
as-servermap        FLOW
tserver         FLOW
exec        FLOW
            CONNECTION { INHERITED }
login       FLOW
            CONNECTION { INHERITED }
cmd         FLOW
            CONNECTION { INHERITED }
printer         FLOW
            CONNECTION { INHERITED }
talk        FLOW
            CONNECTION { INHERITED }
ntalk       FLOW
            CONNECTION { INHERITED }
utime       FLOW
efs         FLOW
timed       FLOW
tempo       FLOW
courier         FLOW
conference      FLOW
netnews         FLOW
netwall         FLOW
apertus-ldp         FLOW
uucp        FLOW
uucp-rlogin         FLOW
klogin      FLOW
kshell      FLOW
new-rwho        FLOW
dsf         FLOW
remotefs        FLOW
rmonitor        FLOW
monitor         FLOW
chshell         FLOW
p9fs        FLOW
```

-continued

```
whoami      FLOW
meter       FLOW
ipcserver       FLOW
urm         FLOW
nqs         FLOW
sift-uft        FLOW
npmp-trap       FLOW
npmp-local      FLOW
npmp-gui        FLOW
ginad       FLOW
doom        FLOW
mdqs        FLOW
elcsd       FLOW
entrustmanager      FLOW
netviewdm1      FLOW
netviewdm2      FLOW
netviewdm3      FLOW
netgw       FLOW
netrcs      FLOW
flexlm      FLOW
fujitsu-dev     FLOW
ris-cm      FLOW
kerberos-adm    FLOW
rfile       FLOW
pump        FLOW
qrh         FLOW
rrh         FLOW
tell        FLOW
nlogin      FLOW
con         FLOW
ns          FLOW
rxe         FLOW
quotad      FLOW
cycleserv       FLOW
omserv      FLOW
webster         FLOW
phonebook       FLOW
vid         FLOW
cadlock         FLOW
rtip        FLOW
cycleserv2      FLOW
submit      FLOW
rpasswd         FLOW
entomb      FLOW
wpages      FLOW
wpgs        FLOW
concert         FLOW
mdbs_daemon FLOW
device      FLOW
xtreelic        FLOW
maitrd      FLOW
busboy      FLOW
garcon      FLOW
puprouter       FLOW
socks       FLOW
```

```
--
--  Virtual.pdl - Virtual Layer definition
--
--  Description:
--    This file contains the definition for the VirtualBase layer used
--    by the embodiment.
--  Copyright:
--    Copyright (c) 1998–1999 Apptitude,
--     (formerly Technically Elite, Inc.)
--    All rights reserved.
--
--  RCS:
--    $Id: Virtual.pdl,v 1.13 1999/04/13 15:48:03 skip Exp $
```

```
--  This includes two things: the flow signature (called FLOWKEY) that the
--  system that is going to use.
--
--  note that not all elements are in the HASH. Reason is that these non-HASHED
--  elements may be varied without the HASH changing, which allows the system
--  to look up multiple buckets with a single HASH. That is, the MeyMatchFlag,
--  StateStatus Flag and MulipacketID may be varied.
--
FLOWKEY {
```

-continued

```
    KeyMatchFlags, -- to tell the system which of the in-HASH elements have to
-- match for the this particular flow record.
                -- Flows for which complete signatures may not yet have
                -- been generated may then be stored in the system
--
StateStatusFlags,
    GroupId1            IN-HASH, -- user defined
    GroupId2            IN-HASH, -- user defined
    DLCProtocol         IN-HASH, , -- data link protocol - lowest level we
                                   -- evaluate. It is the type for the
-- Ethernet V 2
    NetworkProtocol     IN-HASH,   -- IP, etc.
    TunnelProtocol      IN-HASH,   -- IP over IPx, etc.
    TunnelTransport     IN-HASH,
    TransportProtocol   IN-HASH,
    ApplicationProtocol IN-HASH,
    DLCAddresses(8)     IN-HASH, -- lowest level address
    NetworkAddresses(16) IN-HASH,
    TunnelAddresses(16) IN-HASH,
    ConnectionIds       IN-HASH,
    MultiPacketId                -- used for fragmentaion purposes
}
--  now define all of the children. In this example, only one virtual
--  child - Ethernet.
virtualChildren     FIELD
            SYNTAX INT(*) { ethernet(1) }
--  now define the base for the children. In this case, it is the same as
--  for the overall system. There may be multiples.
VirtualBase    PROTOCOL
::= { VirtualChildren=virtualChildren }
--
--  The following is the header that every packet has to have and
--  that is placed into the system by the packet acquisition system.
--
VirtualBase    FLOW
            HEADER { LENGTH=8 }
            CHILDREN { DESTINATION=VirtualChildren } -- this will be
--  Ethernet for this example.
--
--  the VirtualBAse will be 01 for these packets.
```

```
--
--  Ethernet.pdl - Ethernet frame definition
--
--  Description:
--    This file contains the definition for the Ethernet frame. In this
--  PDL file, the decision on EtherType vs. IEEE is made. If this is
--  EtherType, the selection is made from this file. It would be possible
--  to move the EtherType selection to another file, if that would assist
--  in the modularity.
--
--  Copyright:
--    Copyright (c) 1994–1998 Applitude, Inc.
--     (formerly Technically Elite, Inc.)
--    All rights reserved.
--
--  RCS:
--    $Id: Ethernet.pdl,v 1.13 1999/01/26 15:15:57 skip Exp $
```

```
--
--  Enumerated type of a 16 bit integer that contains all of the
--  possible values of interest in the etherType field of an
--  Ethernet V2 packet.
--
etherType    FIELD
            SYNTAX          INT(16) { xns(0x0600), ip(0x0800),
                            chaosnet(0x0804), arp(0x0806),
                            vines(0xbad),
                            vinesLoop(0x0bae), vinesLoop(0x80c4),
                            vinesEcho(0xbaf), vinesEcho(0x80c5),
                            netbios(0x3c00, netbios(0x3c01),
                            netbios(0x3c02), netbios(0x3c03),
                            netbios(0x3c04), netbios(0x3c05),
                            netbios(0x3c06), netbios(0x3c07)
                            netbios(0x3c08), netbios(0x3c09)
                            netbios(0x3c0a), netbios(0x3c0b),
                            netbios(0x3c0c), netbios(0x3c0d)
                            dec(0x6000), mop(0x6001), mop2(0x6002)
                            drp(0x6003), lat(0x6004), decDiag(0x6005),
```

-continued

```
                        lavc(0x6007), rarp(0x8035), appleTalk(0x809b),
                        sna(0x80d5), aarp(0x80f3), ipx(0x8137)
                        snmp(0x814c), ipv6(0x86dd), loopback(0x9000) }
            DISPLAY-HINT    "1x:"
            LOOKUP          FILE "EtherType.cf"
            DESCRIPTION
                "Ethernet type field"
--
--  The unformatted data field in and Ethernet V2 type frame
--
etherData   FIELD
            SYNTAX          BYTESTRING(46..1500)
            ENCAP           etherType
            DISPLAY-HINT    "HexDump"
            DESCRIPTION
                "Ethernet data"
--
--  The layout and structure of an Ethernet V2 type frame with
--  the address and protocol fields in the correct offset position
ethernet    PROTOCOL
            DESCRIPTION
                "Protocol format for an Ethernet frame"
            REFERENCE       "RFC 894"
::= { MacDest=macAddress, MacSrc=macAddress, EtherType=etherType,
    Data=etherData )
--
--  The elements from this Ethernet frame used to build a flow key
--  to classify and track the traffic. Notice that the total length
--  of the header for this tyoe of packet is fixed and at 14 bytes or
--  octets in length. The special field, LLC-CHECK, is specific to
--  Ethernet frames for the decoding of the base Ethernet type value.
--  If it is NOT LLC, the protocol field in the flow is set to the
--  EtherType value decoded from the packet.
--
ethernet    FLOW
            HEADER { LENGTH=14 }
            DLC-LAYER {
                SOURCE=MacSrc,
                DESTINATION=MacDest,
                TUNNELING,
                PROTOCOL
            }
            CHILDREN { DESTINATION=EtherType, LLC-CHECK=11c }
-------------------------------------------------------------------
--
--  IEEE8022.pdl - IEEE 802.2 frame definitions
--
--  Description:
--    This file contains the definition for the IEEE 802.2 Link Layer
--    protocols including the SNAP (Sub-network Access Protocol).
--
--  Copyright:
--    Copyright (c) 1994-1998 Apptitude, Inc.
--     (formerly Technically Elite, Inc.)
--    All rights reserved.
--
--  RCS:
--    $Id: IEEE8022.pdl,v 1.18 1999/01/26 15:15:58 skip Exp $
-------------------------------------------------------------------
--
--  IEEE 802.2 LLC
--
11cSap      FIELD
            SYNTAX          INT(16) { ipx(0xFFFF), ipx(0xE0E0), isoNet(0xFEFE),
                            netbios(0xF0F0), vsnap(0XAAAA), ip(0x0606),
                            vines(0xBCBC), xns(0x8080), spanningTree(0x4242),
                            sna(0x0c0c), sna(0x0808), sna(0x0404) }
            DISPLAY-HINT    "ix:"
            DESCRIPTION
                "Service Access Point"
11cControl      FIELD
            -- This is a special field. When the decoder encounters this field, it
            -- invokes the hard-coded LLC decoder to decode the rest of the packet.
            -- This is necessary because LLC decoding requires the ability to
            -- handle forward references which the current PDL format does not
            -- support at this time.
            SYNTAX          UNSIGNED INT(8)
            DESCRIPTION
                "Control field"
```

-continued

```
11cPduType      FIELD
        SYNTAX BITSTRING(2) { 11cInformation(0), 11cSupervisory(1),
            11cInformation(2), 11cUnnumbererd(3) }
11cData         FIELD
        SYNTAX          BYTESTRING(38..1492)
        ENCAP           11cPduType
        FLAGS           SAMELAYER
        DISPLAY-HINT    "HexDump"
11c     PROTOCOL
        SUMMARIZE
            "$11cPduType == 11cUnnumbered" :
              "LLC ($SAP) $Modifier"
            "$11cPduType == 11cSupervisory" :
              "LLC ($SAP) $Function N(R)=$NR"
            "$11cPduType == 0|2" :
              "LLC ($SAP) N(R)=$NR N(S)=$NS"
            "Default"
              "LLC ($SAP) $11cPduType"
        DESCRIPTION
            "IEEE 802.2 LLC frame format"
::= { SAP=11cSap, Control=11cControl, Data=11cData }
11c      FLOW
        HEADER { LENGTH=3 }
        DLC-LAYER { PROTOCOL }
        CHILDREN { DESTINATION=SAP }
11cUnnumberedData FIELD
        SYNTAX          BYTESTRING(0..1500)
        ENCAP           11cSap
        DISPLAY-HINT    "HexDump"
11cUnnumbered PROTOCOL
        SUMMARIZE
            "Default" :
              "LLC ($SAP) $Modifier"
::= { Data=11cUnnumberedData }
11cSupervisoryData      FIELD
        SYNTAX          BYTESTRING(0..1500)
        DISPLAY-HINT    "HexDump"
11cSupervisory          PROTOCOL
        SUMMARIZE
            "Default" :
              "LLC ($SAP) $Function N(R)=$NR"
::= { Data=11cSupervisoryData }
11cInformationData      FIELD
        SYNTAX          BYTESTRING(0..1500)
        ENCAP           11cSap
        DISPLAY-HINT    "HexDump"
11cInformation          PROTOCOL
        SUMMARIZE
            "Default" :
              "LLC ($SAP) N(R)=$NR N(S)=$NS"
::= { Data=11cInformationData }
--
--  SNAP
--
snapOrgCode     FIELD
        SYNTAX          BYTESTRING(3) { snap("00:00:00"}, ciscoOUI("00:00:0C"),
                        appleOUI("08:00:07") }
        DESCRIPTION
            "Protocol ID or Organizational Code"
vsnapData       FIELD
        SYNTAX          BYTESTRING(46..1500)
        ENCAP           snapOrgCode
        FLAGS           SAMELAYER
        DISPLAY-HINT    "HexDump"
        DESCRIPTION
            "SNAP LLC data"
vsnap   PROTOCOL
        DESCRIPTION
            "SNAP LLC Frame"
::= { OrgCode=snapOrgCode, Data=vsnapData }
vsnap           FLOW
                HEADER { LENGTH=3 }
                DLC-LAYER { PROTOCOL }
                CHILDREN { DESTINATION=OrgCode }
snapType         FIELD
                SYNTAX      INT(16) { xns(0x0600), ip(0x0800), arp(0x0806)
                            vines (0xbad),
                            mop(0x6001), mop2(0x6002), drp(0x6003),
                            lat(0x6004), decDiag(0x6005), lavc(0x6007)
```

-continued

```
                    rarp(0x8035), appleTalk(0x809B), sna(0x80d5),
                    aarp(0x80F3), ipx(0x8137), snmp(0x814c), ipv6(0x86dd) }
            DISPLAY-HINT    "1x:"
            LOOKUP          FILE "EtherType.cf"
            DESCRIPTION
              "SNAP type field"
snapData    FIELD
        SYNTAX          BYTESTRING(46..1500)
        ENCAP           snapType
        DISPLAY-HINT    "HexDump"
        DESCRIPTION
            "SNAP data"
snap    PROTOCOL
        SUMMARIZE
            "$OrgCode == 00:00:00"
              "SNAP Type=$SnapType"
            "Default"
              "VSNAP Org=$OrgCode Type=$SnapType"
        DESCRIPTION
            "SNAP Frame"
::={ SnapType=snapType, Data=snapData }
snap    FLOW
        HEADER { LENGTH=2 }
        DLC-LAYER { PROTOCOL }
        CHILDREN { DESTINATION=SnapType }
-------------------------------------------------------------------
--
--  IEEE8023.pdl - IEEE 802.3 frame definitions
--  Description:
--    This file contains the definition for the IEEE 802.3 (Ethernet)
--    protocols.
--
--  Copyright:
--    Copyright (c) 1994-1998 Apptitude, Inc.
--     (formerly Technically Elite, Inc.)
--    All rights reserved.
--
--  RCS:
--    $Id: IEEE8023.pdl,v 1.7 1999/01/26 15:15:58 skip Exp $
--
-------------------------------------------------------------------
--
--  IEEE 802.3
--
ieee8023Length          FIELD
        SYNTAX UNSIGNED INT(16)
ieee8023Data    FIELD
        SYNTAX          BYTESTRING(38..1492)
        ENCAP           =11c
        LENGTH          "$ieee8023Length"
        DISPLAY-HINT    "HexDump"
ieee8023        PROTOCOL
        DESCRIPTION
            "IEEE 802.3 (Ethernet) frame"
        REFERENCE       "RFC 1042"
::= { MacDest=macAddress, Mac:Src=macAddress, Length=ieee8023Length,
        Data=ieee8023Data }
-------------------------------------------------------------------
--
--  IP.pdl - Internet Protocol (IP) definitions
--
--  Description:
--    This file contains the packet definitions for the Internet
--    Protocol. These elements are all of the fields, templates and
--  processes required to recognize, decode and classify IP datagrams
--  found within packets.
--
--  Copyright:
--    Copyright (c) 1994-1998 Apptitude, Inc.
--     (formerly Technically Elite, Inc.)
--    All rights reserved.
--
--  RCS:
--    $Id: IP.pdl,v 1.14 1999/01/26 15:15:58 skip Exp $
--
-------------------------------------------------------------------
--
--  The following are the fields that make up an IP datagram.
--  Some of these fields are used to recognize datagram elements, build
```

-continued

```
--  flow signatures and determine the next layer in the decode process.
--
ipVersion       FIELD
                SYNTAX INT(4)
                DEFAULT     "4"
ipHeaderLength      FIELD
                SYNTAX INT(4)
ipTypeOfService     FIELD
                SYNTAXBITSTRING(8) { minCost(1), maxReliability(2),
                    maxThruput(3), minDelay(4) }
ipLength        FIELD
                SYNTAX UNSIGNED INT(16)
--
--  This field will tell us if we need to do special processing to support
--  the payload of the datagram existing in multiple packets.
--
ipFlags             FIELD
                SYNTAX BITSTRING(3) { moreFrags(0), dontFrag(1) }
ipFragmentOffset FIELD
                SYNTAX INT(13)
--
--  This field is used to determine the children or next layer of the
--  datagram.
--
ipProtocol      FIELD
                SYNTAX INT(8)
                LOOKUP FILE "IpProtocol.cf"
ipData          FIELD
                SYNTAX          BYTESTRING(0..1500)
                ENCAP           ipProtocol
                DISPLAY-HINT    "HexDump"
--
--  Detailed packet layout for the IP datagram. This includes all fields
--  and format. All offsets are relative to the beginning of the header.
ip  PROTOCOL
        SUMMARIZE
            "$FragmentOffset != 0":
                "IPFragment ID=$Identification Offset=$FragmentOffset"
            "Default" :
                "IP Protocol=$Protocol"
        DESCRIPTION
          "Protocol format for the Internet Protocol"
        REFERENCE       "RFC 791"
::= {   Version=ipVersion, HeaderLength=ipHeaderLength,
        TypeOfService=ipTypeOfService, Length=ipLength,
        Identification=UInt16, IpFlags=ipFlags,
        FragmentOffset=ipFragmentOffset, TimeToLive=Int8,
        Protocol=ipProtocol, Checksum=ByteStr2,
        IpSrc=ipAddress, IpDest=ipAddress, Options=ipOptions,
        Fragment=ipFragment, Data=ipData }
--
--  This is the description of the signature elements required to build a flow
--  that includes the IP network layer protocol. Notice that the flow builds on
--  the lower layers. Only the fields required to complete IP are included.
--  This flow requires the support of the fragmentation engine as well as the
--  potential of having a tunnel. The child field is found from the IP
--  protocol field
--
ip      FLOW
        HEADER { LENGTH=HeaderLength, IN-WORDS }
        NET-LAYER {
          SOURCE=IpSrc,
          DESTINATION=IpDest,
          FRAGMENTATION=IPV4,
          TUNNELING
        }
        CHILDREN { DESTINATION=Protocol }
ipFragData      FIELD
                SYNTAX          BYTESTRING(1..1500)
                LENGTH          "$ipLength - $ipHeaderLength * 4"
                DISPLAY-HINT    "HexDump"
ipFragment      Group
                OPTIONAL        "$FragmentOffset != 0"
::= { Data=ipFragData }
ipOptionCode    FIELD
                SYNTAXINT(8) { ipRR(0x07), ipTimestamp(0x44),
                    ipLSRR(0x83), ipSSRR(0x89) }
                DESCRIPTION
                  "IP option code"
```

-continued

```
ipOptionLength      FIELD
                    SYNTAX UNSIGNED INT(8)
                    DESCRIPTION
                      "Length of IP option"
ipOptionData        FIELD
                    SYNTAX          BYTESTRING(0..1500)
                    ENCAP           ipOptionCode
                    DISPLAY-HINT    "HexDump"
ipOptions           GROUP
                    LENGTH          "($ipHeaderLength * 4) - 20"
::= {     Code=ipOptionCode, Length=ipOptionLength, Pointer=UInt8,
          Data=ipOptionData }
-------------------------------------------------------------------
--
--  TCP.pdl - Transmission Control Protocol (TCP) definitions
--  Description:
--    This file contains the packet definitions for the Transmission
--    Control Protocol. This protocol is a transport service for
--  the IP protocol. In addition to extracting the protocol information
--  the TCP protocol assists in the process of identification of connections
--  for the processing of states.
--
--  Copyright:
--    Copyright (c) 1994-1998 Apptitude, Inc.
--     (formerly Technically Elite, Inc.)
--    All rights reserved.
--  RCS:
--    $Id: TCP.pdl,v 1.9 1999/01/26 15:16:02 akip Exp $
--
-------------------------------------------------------------------
--
--  This is the 16 bit field where the child protocol is located for
--  the next layer beyond TCP.
--
tcpPort FIELD
    SYNTAX UNSIGNED INT(16)
    LOOKUP FILE "TcpPort.cf"
tcpHeaderLen FIELD
    SYNTAX INT(4)
tcpFlags FIELD
    SYNTAXBITSTRING(12) { fin(0), syn(1), rst(2), psh(3), ack(4), urg(5) }
tcpData FIELD
        SYNTAX          BYTESTRING(0..1564)
        LENGTH          "($ipLength - ($ipHeaderLength * 4)) - ($tcpHeaderLen * 4)"
        ENCAP           tcpPort
        DISPLAY-HINT    "HexDump"
--
--  The layout of the TCP datagram found in a packet. Offset based on the
--  beginning of the header for TCP.
tcp PROTOCOL
        SUMMARIZE
            "Default" :
                "TCP ACK=$Ack WIN=$WindowSize"
        DESCRIPTION
            "Protocol format for the Transmission Control Protocol"
        REFERENCE       "RFC 793"
::= {   Srcport=tcpPort, DestPort=tcpPort, SequenceNum=UInt32,
        Ack=UInt32, HeaderLength=tcpHeaderLen, TcpFlags=tcpFlags,
        WindowSize=UInt16, Checksum=ByteStr2,
        UrgentPointer=UInt16, Options=tcpOptions, Data=tcpData }
--
--  The flow elements required to build a key for a TCP datagram.
--  Noticed that this FLOW description has a CONNECTION section. This is
--  used to describe what connection state is reached for each setting
--  of the TcpFlags field.
--
tcp         FLOW
        HEADER { LENGTH=HeaderLength, IN-WORDS }
        CONNECTION {
            IDENTIFIER=SequenceNum,
            CONNECT-START="TcpFlags:1",
            CONNECT-COMPLETE="TcpFlags:4",
            DISCONNECT-START="TcpFlags:0",
            DISCONNECT-COMPLETE="TcpFlags:4"
        }
        PAYLOAD { INCLUDE-HEADER }
        CHILDREN { DESTINATION=DestPort, SOURCE=SrcPort }
tcpOptionKind   FIELD
                SYNTAX UNSIGNED INT(8) { tcpOptEnd(0), tcpNop(1), tcpMSS(2),
```

-continued

```
                                tcpWscale(3), tcpTimestamp(4) }
                DESCRIPTION
                    "Type of TCP option"
tcpOptionData   FIELD
                SYNTAX          BYTESTRING(0..1500)
                ENCAP           tcpOptionKind
                FLAGS           SAMELAYER
                DISPLAY-HINT    "HexDump"
tcpOptions      GROUP
                LENGTH          "($tcpHeaderLen * 4) - 20"
--              SUMMARIZE
--                  "Default" :
--                    "Option=$Option, Len=$OptionLength, $OptionData"
::= { Option=tcpOptionKind, optionLength=UInt8, OptionData=tcpOptionData }
tcpMSS          PROTCCOL
::= { MaxSegmentSize=UInt16 }
-------------------------------------------------------------------
--
--  UDP.pdl - User Datagram Protocol (UDP) definitions
--
--  Description:
--    This file contains the packet definitions for the User Datagram
--    Protocol.
--
--  Copyright:
--    Copyright (c) 1994-1998 Apptitude, Inc.
--     (formerly Technically Elite, Inc.)
--    All rights reserved.
--
--  RCS:
--    $Id: UDP.pdl,v 1.9 1999/01/26 15:16:02 skip Exp $
--
-------------------------------------------------------------------
udpPort         FIELD
        SYNTAX UNSIGNED INT(16)
        LOOKUPFILE "Udpport.cf"
udpLength FIELD
        SYNTAX          UNSIGNED INT(16)
udpData FIELD
        SYNTAX          BYTESTRING(0..1500)
        ENCAP           udpPort
        DISPLAY-HINT    "HexDump"
udp     PROTOCOL
        SUMARIZE
            "Default" :
                "UDP Dest=$DestPort Src=$SrcPort"
        DESCRIPTION
            "Protocol format for the User Datagram Protocol."
        REFERENCE       "RFC 768"
::= {   SrcPort=udpPort, DestPort=udpPort, Length=udpLength,
        Checksum=ByteStr2, Data=udpData }
udp     FLOW
        HEADER { LENGTH=8 }
        CHILDREN { DESTINATION=DestPort, SOURCE=Srcport }
-------------------------------------------------------------------
--
--  RPC.pdl - Remote Procedure Calls (RPC) definitions
--
--  Description:
--    This file contains the packet definitions for Remote Procedure
--    Calls.
--
--  Copyright:
--    Copyright (c) 1994-1999 Apptitude,
--     (formerly Technically Elite, Inc.)
--    All rights reserved.
--  RCS:
--    $Id: RPC.pdl,v 1.7 1999/01/26 15:16:01 skip Exp $
-------------------------------------------------------------------
rpcType     FIELD
        SYNTAX UNSIGNED INT(32) { rpcCall(0), rpcReply(1) }
rpcData     FIELD
        SYNTAX          BYTESTRING(0..100)
        ENCAP           rpcType
        FLAGS           SAMELAYER
        DISPLAY-HINT    "HexDump"
rpc     PROTOCOL
        SUMMARIZE
            "$Type == rpcCall"
```

-continued

```
            "RPC $Program"
          "$ReplyStatus == rpcAcceptedReply" :
            "RPC Reply Status=$Status"
          "$ReplyStatus == rpcDeniedReply"
            "RPC Reply Status=$:Status, AuthStatus=$AuthStatus"
          "Default"
            "RPC $Program"
      DESCRIPTION
          "Protocol format for RPC"
      REFERENCE
          "RFC 1057"
::= {  XID=UInt32, Type=rpcType, Data=rpcData }
rpc   FLOW
      HEADER { LENGTH=0 }
      PAYLOAD { DATA=XID, LENGTH=256 }

--  RPC Call

rpcProgram FIELD
      SYNTAX UNSIGNED INT(32) { portMapper(100000), nfs(100003),
          mount(100005), lockManager(100021), statusMonitor(100024) }
rpcProcedure GROUP
      SUMMARIZE
          "Default" :
            "Program=$Program, Version=$Version, Procedure=$Procedure"
::= {  Program=rpcProgram, Version=UInt32, Procedure=UInt32 }
rpcAuthFlavor FIELD
      SYNTAX UNSIGNED INT(32) { null(0), unix(1), short(2) }
rpcMachine   FIELD
      SYNTAX LSTRING(4)
rpcGroup     GROUP
      LENGTH "$NumGroups * 4"
::= {  Gid=Int32 }
rpcCredentials      GROUP
      LENGTH "$CredentialLength"
::= {  Stamp=UInt32, Machine=rpcMachine, Uid=Int32, Gid=Int32,
      NumGroups=UInt32, Groups=rpcGroup }
rpcVerifierData      FIELD
      SYNTAX          BYTESTRING(0..400)
      LENGTH          "$VerifierLength"
rpcEncap      FIELD
      SYNTAX COMBO Program Procedure
      LOOKUP FILE "RPC.cf"
rpcCallData   FIELD
      SYNTAX          BYTESTRING(0..100)
      ENCAP           rpcEncap
      DISPLAY-HINT    "HexDump"
rpcCall      PROTOCOL
      DESCRIPTION
          "Protocol format for RPC call"
::= {  RPCVersion=UInt32, Procedure=rpcProcedure,
      CredentialAuthFlavor=rpcAuthFlavor, CredentialLength=UInt32,
      Credentials=rpcCredentials,
      VerifierAuthFlavor=rpcAuthFlavor, VerifierLength=UInt32,
      Verifier=rpcVerifierData, Encap=rpcEncap, Data=rpcCallData }

--  RPC Reply

rpcReplyStatus      FIELD
      SYNTAX INT(32) { rpcAcceptedReply(0), rpcDeniedReply(1) }
rpcReplyData  FIELD
      SYNTAX          BYTESTRING(0..40000)
      ENCAP           rpcReplyStatus
      FLAGS           SAMELAYER
      DISPLAY-HINT    "HexDump"
rpcReply      PROTOCOL
      DESCRIPTION
          "Protocol format for RPC reply"
::= {  ReplyStatus=rpcReplyStatus, Data=rpcReplyData }
rpcAcceptStatus      FIELD
      SYNTAX INT(32) { Success(0), ProgUnavail(1), ProgMismatch(2),
          ProcUnavail(3), GarbageArgs(4), SystemError(5)
rpcAcceptEncap      FIELD
      SYNTAX BYTESTRING(0)
      FLAGS  NOSHOW
rpcAcceptData FIELD
      SYNTAX          BYTESTRING(0..40000)
      ENCAP           rpcAcceptEncap
      DISPLAY-HINT    "HexDump"
```

-continued

```
rpcAcceptedReply PROTOCOL
::= {   VerifierAuthFlavor=rpcAuthFlavor, VerifierLength=UInt32,
        Verifier=rpcVerifierData, Status=rpcAcceptStatus,
        Encap=rpcAcceptEncap, Data=rpcAcceptData }
rpcDeniedstatus     FIELD
        SYNTAX INT(32) { rpcVersionMismatch(0), rpcAuthError(1) }
rpcAuthStatus FIELD
        SYNTAX INT(32) { Okay(0), BadCredential(1), RejectedCredential(2),
            BadVerifier(3), ReDectedVerifier(4), TooWeak(5),
            InvalidResponse(6), Failed(7) }
rpcDeniedReply      PROTOCOL
::= {   Status=rpcDeniedStatus, AuthStatus=rpcAuthStatus }
--------------
--  RPC Transactions
--------------
rpcBindLookup PROTOCOL
        SUMMARIZE
            "Default" :
              "RPC GetPort Prog=$Prog, Ver=$Ver, Proto=$Protocol"
::= {   Prog=rpcProgram, Ver=UInt32, Protocol=UInt32 }
rpcBindLookupReply PROTOCOL
        SUMMARIZE
            "Default"
              "RPC GetPortReply Port=$Port"
::= { Port=UInt32 }
-------------------------------------------------------------
--
--  NFS.pdl - Network File System (NFS) definitions
--
--  Description:
--    This file contains the packet definitions for the Network File
--    System.
--  Copyright:
--    Copyright (c) 1994-1998 Apptitude, Inc.
--     (formerly Techhically Elite, Inc.)
--    All rights reserved.
--  RCS:
--    $Id: NFS.pdl,v 1.3 1999/01/26 15:15:59 akip Exp $
-------------------------------------------------------------
nfsString       FIELD
        SYNTAX LSTRING(4)
nfsHandle       FIELD
        SYNTAX          BYTESTRING(32)
        DISPLAY-HINT    "16x\n          "
nfsData         FIELD
        SYNTAX          BYTESTRING(0..100)
        DISPLAY-HINT    "HexDump"
nfsAccess       PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS Access $Filename"
::= { Handle=nfsHandle, Filename=nfsString }
nfsStatus       FIELD
        SYNTAX INT(32) { OK(0), NoSuchFile(2) }
nfsAccessReply      PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS AccessReply $Status"
::= { Status=nfsStatus }
nfsMode             FIELD
        SYNTAX UNSIGNED INT(32)
        DISPLAY-HINT "4o"
nfsCreate       PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS Create $Filename"
::= { Handle=nfsHandle, Filename=nfsString, Filler=Int8, Mode=nfsMode,
        Uid=Int32, Gid=Int32, Size=Int32, AccessTime=Int64, ModTime=Int64 }
nfsFileType     FIELD
        SYNTAX INT(32) { Regular(1), Directory(2) }
nfsCreateReply      PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS CreateReply $Status"
::= { Status=nfsStatus, Handle=nfsHandle, FileType=nfsFileType,
        Mode=nfsMode, Links=UInt32, Uid=Int32, Gid=Int32, Size=Int32,
        BlockSize=Int32, NumBlocks=Int64, FileSysId=UInt32, FileId=UInt32,
        AccessTime=Int64, ModTime=Int64, InodeChangeTime=Int64 }
nfsRead     PROTOCOL
```

-continued

```
        SUMMARIZE
            "Default" :
              "NFS Read Offset=$Offset Length=$Length"
::= { Length=Int32, Handle=nfsHandle, Offset=UInt64, Count=Int32 }
nfsReadReply  PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS ReadReply $Status"
::= { Status=nfsStatus, FileType=nfsFileType,
        Mode=nfsMode, Links=UInt32, Uid=Int32, Gid=Int32, Size=Int32,
        BlockSize=Int32, NumBlocks=Int64, FileSysId=UInt32, FileId=UInt32,
        AccessTime=Int64, ModTime=Int64, InodeChangeTime=Int64 }
nfsWrite PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS Write Offset=$Offset"
::= { Handle=nfsHandle, Offset=Int32, Data=nfsData }
nfsWriteReply PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS WriteReply $Status"
::= { Status=nfsStatus, FileType=nfsFileType,
        Mode=nfsMode, Links=UInt32, Uid=Int32, Gid=Int32, Size=Int32,
        BlockSize=Int32, NumBlocks=Int64, FileSysId=UInt32, FileId=UInt32,
        AccessTime=Int64, ModTime=Int64, InodeChangeTime=Int64 }
nfsReadDir    PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS ReadDir"
::= { Handle=nfsHandle, Cookie=Int32, Count=Int32 }
nfsReadDirReply       PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS ReadDirReply $Status"
::= { Status=nfsStatus, Data=nfsData }
nfsGetFileAttr        PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS GetAttr"
::= { Handle=nfsHandle }
nfsGetFileAttrReply PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS GetAttrReply $Status $FileType"
::= { Status=nfsStatus, FileType=nfsFileType,
        Mode=nfsMode, Links=UInt32, Uid=Int32, Gid=Int32, Size=Int32,
        BlockSize=Int32, NumBlocks=Int64, FileSysId=UInt32, FileId=UInt32,
        AccessTime=Int64, ModTime=Int64, InodeChangeTime=Int64 }
nfsReadLink   PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS ReadLink"
::= { Handle=nfsHandle }
nfsReadLinkReply  PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS ReadLinkReply Path=$Path"
::= { Status=nfsStatus, Path=nfsString }
nfsMount          PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS Mount $Path"
::= { Path=nfsstring
nfsMountReply PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS MountReply $MountStatus"
::= { MountStatus=nfsStatus, Handle=nfsHandle }
nfsStatFs       PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS StatFs"
::= { Handle=nfsHandle }
nfsStatFsReply       PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS StatFsReply $Status"
::= { Status=nfsStatus, TransferSize=UInt32, BlockSize=UInt32,
        TotalBlocks=UInt32, FreeBlocks=UInt32, AvailBlocks=UInt32 }
```

-continued

```
nfsRemoveDir   PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS RmDir $Name"
::= { Handle=nfsHandle, Name=nfsString }
nfsRemoveDirReply   PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS RmDirReply $Status"
::= { Status=nfsStatus }
nfsMakeDir    PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS MkDir $Name"
::= { Handle=nfsHandle, Name=nfsString }
nfsMakeDirReply   PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS MkDirReply $Status"
::= { Status=nfsStatus }
nfsRemove       PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS Remove $Name"
::= { Handle=nfsHandle, Name=nfsString }
nfsRemoveReply   PROTOCOL
        SUMMARIZE
            "Default" :
              "NFS RemoveReply $Status"
::= { Status=nfsStatus
-----------------------------------------------------------
--
--   HTTP.pdl - Hypertext Transfer Protocol (HTTP) definitions
--
--   Description:
--      This file contains the packet definitions for the Hypertext Transfer
--      Protocol.
--
--   Copyright:
--      Copyright (c) 1994-1999 Apptitude, Inc.
--      (formerly Technically Elite, Inc.)
--      All rights reserved.
--   RCS:
--      $Id: HTTP.pdl,v 1.13 1999/04/13 15:47:57 akip Exp $
--
-----------------------------------------------------------
httpData    FIELD
            SYNTAX          BYTESTRING(1..1500)
            LENGTH            "($ipLength - ($ipHeaderLength * 4)) - ($tcpHeaderLen
* 4)"
            DISPLAY-HINT    "Text"
            FLAGS           NOLABEL
http        PROTOCOL
            SUMMARIZE
                "$httpData m/^GET|^HTTP|^HEAD|^POST/" :
                  "HTTP $httpData"
                "$httpData m/^[Dd]ate|^[Ss]erver|^[L1]ast-[Mm]odified/" :
                  "HTTP $httpData"
                "$httpData m/^[Cc]ontent-/" :
                  "HTTP $httpData"
                "$httpData m/^<HTML>/" :
                  "HTTP [HTML document]"
                "$httpData m/^GIF/" :
                  "HTTP [GIF image]"
                "Default" :
                  "HTTP [Data]"
            DESCRIPTION
                "Protocol format for HTTP."
::= { Data=httpData }
http        FLOW
            CONNECTION { INHERITED }
            PAYLOAD { INCLUDE-HEADER, DATA=Data, LENGTH=256 }
            STATES
                "S0:   CHECKCONNECT, GOTO S1
                       DEFAULT NEXT S0
                S1:    WAIT 2, GOTO S2, NEXT S1
                       DEFAULT NEXT S0
                S2:    MATCH
                         '\n\r\n'              900 0 0 255 0, NEXT S3
```

-continued

```
                    '\n\n'                  900 0 0 255 0, NEXT 53
                    'POST /tds?'             50 0 0 127 1, CHILD sybaseWebsq1
                    '.hts HTTP/1.0'          50 4 0 127 1, CHILD sybaseJdbc
                    'jdbc:sybase:Tds'        50 4 0 127 1, CHILD sybaseTds
                    'PCN-The Poin'          500 4 1 255 0, CHILD pointcast
                    't: BW-C-'              100 4 1 255 0, CHILD backweb
                    DEFAULT NEXT S3
               S3:  MATCH
                    '\n\r\n'                50 0 0   0 0, NEXT S3
                    '\n\n'                   50 0 0   0 0, NEXT S3
                    'Content-Type:'         800 0 0 255 0, CHILD mime
                    'PCN-The Poin'          500 4 1 255 0, CHILD pointcast
                    't: BW-C-'              100 4 1 255 0, CHILD backweb
                    DEFAULT NEXT S0"
sybaseWebsql   FLOW
               STATE-BASED
sybaseJdbc     FLOW
               STATE-BASED
sybaseTds      FLOW
               STATE-BASED
pointcast      FLOW
               STATE-BASED
backweb        FLOW
               STATE-BASED
mime           FLOW
               STATE-BASED
               STATES
                 "S0:      MATCH
                           'application'  900 0 0   1 0, CHILD mimeApplication
                           'audio'        900 0 0   1 0, CHILD mimeAudio
                           'image'         50 0 0   1 0, CHILD mimeImage
                           'text'          50 0 0   1 0, CHILD mimeText
                           'video'         50 0 0   1 0, CHILD mimeVideo
                           'x-world'      500 4 1 255 0, CHILD mimeXworld
                         DEFAULT GOTO S0"
mimeApplication  FLOW
                 STATE-BASED
mimeAudio      FLOW
               STATE-BASED
               STATES
                 "S0:  MATCH
                       'basic'           100 0 0 1 0, CHILD pdBasicAudio
                       'midi'            100 0 0 1 0, CHILD pdMidi
                       'mpeg'            100 0 0 1 0, CHILD pdMpeg2Audio
                       'vnd.m-realaudio' 100 0 0 1 0, CHILD pdRealAudio
                       'wav'             100 0 0 1 0, CHILD pdWav
                       'x-aiff'          100 0 0 1 0, CHILD pdAiff
                       'x-midi'          100 0 0 1 0, CHILD pdMidi
                       'x-mpeg'          100 0 0 1 0, CHILD pdMpeg2Audio
                       'x-mpgurl'        100 0 0 1 0, CHILD pdMpeg3Audio
                       'x-pn-realaudio'  100 0 0 1 0, CHILD pdRealAudio
                       'x-wav'           100 0 0 1 0, CHILD pdWav
                     DEFAULT GOTO S0"
mimeImage      FLOW
               STATE-BASED
mimeText       FLOW
               STATE-BASED
mimeVideo      FLOW
               STATE-BASED
mimeXworld     FLOW
               STATE-BASED
pdBasicAudio   FLOW
               STATE-BASED
pdMidi         FLOW
               STATE-BASED
pdMpeg2Audio   FLOW
               STATE-BASED
pdMpeg3Audio   FLOW
               STATE-BASED
pdRealAudio    FLOW
               STATE-BASED
pdWav          FLOW
               STATE-BASED
pdAiff         FLOW
               STATE-BASED
```

What is claimed is:

1. A method of performing protocol specific operations on a packet passing through a connection point on a computer network, the method comprising:

(a) receiving the packet:

(b) receiving a set of protocol descriptions for a plurality of protocols that conform to a layered model, a protocol description for a particular protocol at a particular layer level including:

(i) if there is at least one child protocol of the protocol at the particular layer level, the-one or more child protocols of the particular protocol at the particular layer level, the packet including for any particular child protocol of the particular protocol at the particular layer level information at one or more locations in the packet related to the particular child protocol,

(ii) the one or more locations in the packet where information is stored related to any child protocol of the particular protocol, and

(iii) if there is at least one protocol specific operation to be performed on the packet for the particular protocol at the particular layer level, the one or more protocol specific operations to be performed on the packet for the particular protocol at the particular layer level; and

(c) performing the protocol specific operations on the packet specified by the set of protocol descriptions based on the base protocol of the packet and the children of the protocols used in the packet,

the method further comprising:

storing a database in a memory, the database generated from the set of protocol descriptions and including a data structure containing information on the possible protocols and organized for locating the child protocol related information for any protocol, the data structure contents indexed by a set of one or more indices, the database entry indexed by a particular set of index values including an indication of validity,

wherein the child protocol related information includes a child recognition pattern,

wherein step (c) of performing the protocol specific operations includes, at any particular protocol layer level starting from the base level, searching the packet at the particular protocol for the child field, the searching including indexing the data structure until a valid entry is found, and

whereby the data structure is configured for rapid searches using the index set.

2. A method according to claim 1, wherein the protocol descriptions are provided in a protocol description language, the method further comprising:

compiling the PDL descriptions to produce the database.

3. A method according to claim 1, wherein the data structure comprises a set of arrays, each array identified by a first index, at least one array for each protocol, each array further indexed by a second index being the location in the packet where the child protocol related information is stored, such that finding a valid entry in the data structure provides the location in the packet for finding the child recognition pattern for an identified protocol.

4. A method according to claim 3, wherein each array is further indexed by a third index being the size of the region in the packet where the child protocol related information is stored, such that finding a valid entry in the data structure provides the location and the size of the region in the packet for finding the child recognition pattern.

5. A method according to claim 4, wherein the data structure is compressed according to a compression scheme that takes advantage of the sparseness of valid entries in the data structure.

6. A method according to claim 5, wherein the compression scheme combines two or more arrays that have no conflicting common entries.

7. A method according to claim 1, wherein the data structure includes a set of tables, each table identified by a first index, at least one table for each protocol, each table further indexed by a second index being the child recognition pattern, the data structure further including a table that for each protocol provides the location in the packet where the child protocol related information is stored, such that finding a valid entry in the data structure provides the location in the packet for finding the child recognition pattern for an identified protocol.

8. A method according to claim 7, wherein the data structure is compressed according to a compression scheme that takes advantage of the sparseness of valid entries in the set of tables.

9. A method according to claim 8, wherein the compression scheme combines two or more tables that have no conflicting common entries.

10. A method of performing protocol specific operations on a packet passing through a connection point on a computer network, the method comprising:

(a) receiving the packet;

(b) receiving a set of protocol descriptions for a plurality of protocols that conform to a layered model, a protocol description for a particular protocol at a particular layer level including:

(i) if there is at least one child protocol of the protocol at the particular layer level, the-one or more child protocols of the particular protocol at the particular layer level, the packet including for any particular child protocol of the particular protocol at the particular layer level information at one or more locations In the packet related to the particular child protocol,

(ii) the one or more locations in the packet where information is stored related to any child protocol of the particular protocol, and

(iii) if there is at least one protocol specific operation to be performed on the packet for the particular protocol at the particular layer level, the one or more protocol specific operations to be performed on the packet for the particular protocol at the particular layer level: and

(c) performing the protocol specific operations on the packet specified by the set of protocol descriptions based on the base protocol of the packet and the children of the protocols used in the packet,

wherein the protocol specific operations include one or more parsing and extraction operations on the packet to extract selected portions of the packet to form a function of the selected portions for identifying the packet as belonging to a conversational flow.

11. A method according to claim 10, wherein step (c) of performing protocol specific operations is performed recursively for any children of the children.

12. A method according to claim 10, wherein which protocol specific operations are performed is step (c) depends on the contents of the packet such that the method adapts to different protocols according to the contents of the packet.

13. A method according to claim 10, wherein the protocol descriptions are provided in a protocol description language.

14. A method according to claim 13, further comprising:

compiling the PDL descriptions to produce a database and store the database in a memory, the database generated from the set of protocol descriptions and including a data structure containing information on the possible protocols and organized for locating the child protocol related information for any protocol, the data structure contents indexed by a set of one or more indices, the database entry indexed by a particular set of index values including an indication of validity,

wherein the child protocol related information includes a child recognition pattern, and

wherein the step of performing the protocol specific operations includes, at any particular protocol layer level starting from the base level, searching the packet at the particular protocol for the child field, the searching including indexing the data structure until a valid entry is found,

whereby the data structure is configured for rapid searches using the index set.

15. A method according to claim 10, further comprising:

looking up a flow-entry database comprising at least one flow-entry for each previously encountered conversational flow, the looking up using at least some of the selected packet portions and determining if the packet matches an flow-entry in the flow-entry database

if the packet is of an existing flow, classifying the packet as belonging to the found existing flow; and

if the packet is of a new flow, storing 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;

wherein for at least one protocol, the parsing and extraction operations depend on the contents of one or more packet headers.

16. A method according to claim 10, wherein the protocol specific operations further include one or more state processing operations that are a function of the state of the flow of the packet.

17. A method of performing protocol specific operations on a packet passing through a connection point on a computer network, the method comprising:

(a) receiving the packet;

(b) receiving a set of protocol descriptions for a plurality of protocols that conform to a layered model, a protocol description for a particular protocol at a particular layer level including:

(i) if there is at least one child protocol of the protocol at the particular layer level, the one or more child protocols of the particular protocol at the particular layer level, the packet including for any particular child protocol of the particular protocol at the particular layer level information at one or more locations in the packet related to the particular child protocol,

(ii) the one or more locations in the packet where information is stored related to any child protocol of the particular protocol, and

(iii) if there is at least one protocol specific operation to be performed on the packet for the particular protocol at the particular layer level, the one or more protocol specific operations to be performed on the packet for the particular protocol at the particular layer level; and

(c) performing the protocol specific operations on the packet specified by the set of protocol descriptions based on the base protocol of the packet and the children of the protocols used in the packet,

wherein the packet belongs to a conversational flow of packets having a set of one or more states, and wherein the protocol specific operations include one or more state processing operations that are a function of the state of the conversational flow of the packet, the state of the conversational flow of the packet being indicative of the sequence of any previously encountered packets of the same conversational flow as the packet.

* * * * *

tw
11-16-07

ISSUE SLIP STAPLE AREA (for additional cross references)

POSITION	INITIALS	ID NO.	DATE
EXAMINER			7/12/02
ASSISTANT EXAMINER		43	7/17/02.
FORMAL REVIEW			
RESPONSE FORMALITY REVIEW		64674	8-22

INDEX OF CLAIMS

✓	Rejected	N	Non-elected
=	Allowed	I	Interference
(Through numeral)	Canceled	A	Appeal
>	Restricted	O	Objected

Claim Final	Claim Original	Date
	1	✓ 5/29/03 6/25/03
11	2	✓ =
12	3	✓ =
1	4	O =
2	5	O =
3	6	O =
4	7	O =
5	8	O =
6	9	O =
7	10	O =
8	11	O =
9	12	O =
10	13	✓ =
13	14	✓ =
14	15	O =
15	16	O =
16	17	✓ =
17	18	✓ =
	19	
	20	
	21	
	22	
	23	
	24	
	25	
	26	
	27	
	28	
	29	
	30	
	31	
	32	
	33	
	34	
	35	
	36	
	37	
	38	
	39	
	40	
	41	
	42	
	43	
	44	
	45	
	46	
	47	
	48	
	49	
	50	

Claim Final	Claim Original	Date
	51	
	52	
	53	
	54	
	55	
	56	
	57	
	58	
	59	
	60	
	61	
	62	
	63	
	64	
	65	
	66	
	67	
	68	
	69	
	70	
	71	
	72	
	73	
	74	
	75	
	76	
	77	
	78	
	79	
	80	
	81	
	82	
	83	
	84	
	85	
	86	
	87	
	88	
	89	
	90	
	91	
	92	
	93	
	94	
	95	
	96	
	97	
	98	
	99	
	100	

Claim Final	Claim Original	Date
	101	
	102	
	103	
	104	
	105	
	106	
	107	
	108	
	109	
	110	
	111	
	112	
	113	
	114	
	115	
	116	
	117	
	118	
	119	
	120	
	121	
	122	
	123	
	124	
	125	
	126	
	127	
	128	
	129	
	130	
	131	
	132	
	133	
	134	
	135	
	136	
	137	
	138	
	139	
	140	
	141	
	142	
	143	
	144	
	145	
	146	
	147	
	148	
	149	
	150	

If more than 150 claims or 10 actions staple additional sheet here

(LEFT INSIDE)

SEARCHED

Class	Sub.	Date	Exmr.
709	203	5/28/03	LD
	206		
	216		
	217		
	222		
	225		
	228		
	238		
	252		
703	26		
370	489		
	13		
	17		
updated	class search	6/18/03	KD
updated	search	6/19/03	LD

INTERFERENCE SEARCHED

Class	Sub.	Date	Exmr.
709	230	6/18/03	LD
	246		
	228		
370	489		

SEARCH NOTES
(INCLUDING SEARCH STRATEGY)

	Date	Exmr.
WEST	5/29/03	LD

(RIGHT OUTSIDE)

PATENT APPLICATION

09609179

JC858 U.S. PTO
09/609179
06/30/00

INITIALS 7/12/00 12

CONTENTS

		Date Received (Incl. C. of M.) or Date Mailed			Date Received (Incl. C. of M.) or Date Mailed
1.	Application 1 papers.		42.		
2.	Ltr. Re/Decl, Fee	8-23	43.		
3.	Decl, Fee	8-24-00	44.		
4.	IDS	4-12-01	45.		
5.	IDS.	4-11-02	46.		
6.	Restriction 3 months	6/4/3	47.		
7.	Amdt A	6-13-03	48.		
8.	Suppl. Amdt B	6-27-03	49.		
9.	notice of allow.	7/01/03	50.		
10.	Amdt C (NE.) PTO	7-14-03	51.		
11.	Ltr. reentry	10-27-03	52.		
12.	Reg Cofc	4/8/04	53.		
13.	Reg. for Cyc	9-4-13	54.		
14.			55.		
15.			56.		
16.			57.		
17.			58.		
18.			59.		
19.			60.		
20.			61.		
21.			62.		
22.			63.		
23.			64.		
24.			65.		
25.			66.		
26.			67.		
27.			68.		
28.			69.		
29.			70.		
30.			71.		
31.			72.		
32.			73.		
33.			74.		
34.			75.		
35.			76.		
36.			77.		
37.			78.		
38.			79.		
39.			80.		
40.			81.		
41.			82.		

(LEFT OUTSIDE)