ISSUE CLASSIFICATION	Class	Subclass
	709	230

PATENT NUMBER

6665725

U.S. **UTILITY** Patent Application

O.I.P.E.	PATENT DATE
SCANNED ZK@ Q.A. CK	DEC 16 2003

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

APPLICANTS

TITLE

Certificate

JUN 29 2004

# of Correction

TO-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 CV; 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)

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 ✓ papers.		42.		
2.	Ltr. Re/Daily Fee	8-23	43.		
3.	Daily Fee	8-24-00	44.		
4.	FWC	4-12-01	45.		
5.	I.D.S.	4-11-02	46.		
6.	Restriction Requirement	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 (NOE) R312	7-14-03	51.		
11.	Ltr. re entry	10-27-03	52.		
12.	Reg Corr	4/8/04	53.		
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)

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

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

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

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

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 A

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

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

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. 38,687

Petitioners EX1039 Page 6

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. 09/608,266 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. 09/608,261 for STATE PROCESSOR FOR PATTERN MATCHING IN A NETWORK MONITOR DEVICE, to inventors Sarkissian, et al., filed June 30, 2000, Attorney/Agent Reference Number APPT-001-5, and incorporated herein by reference.

FIELD OF INVENTION

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

COPYRIGHT NOTICE

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

BACKGROUND

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

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 09/608,157 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

A portion of this of this document included with the patent contains material which is subject to copyright protection. The copyright owner (Apptitude, Inc., of San Jose, California, formerly Technically Elite, Inc.) has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure or this document, as it appears

in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

Copyright © 1997-1999 by Apptitude, Inc. (formerly Technically Elite, Inc.). All Rights Reserved.

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=11c }
```

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)

ipProtocol1     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)
5                 LENGTH        "ipLength - ipHeaderLength * 4"
                  DISPLAY-HINT  "HexDump"

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

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

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

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

    ipOptions     GROUP
                  LENGTH        "(ipHeaderLength * 4) - 20"
30  ::= { 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"

sybaseWebsql1    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-mpgur1'          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
ariel1 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=UIntl6, Checksum=ByteStr2,
       UrgentPointer=UIntl6, 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) }
```

```
rpcAuthStatus FIELD
       SYNTAX INT(32) { Okay(0), BadCredential(1), RejectedCredential(2),
              BadVerifier(3), RejectedVerifier(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 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 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"

sybaseWebsql1   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.

Petitioners' EX1039 Page 134

PRINT OF DRAWINGS
AS ORIGINALLY FILED

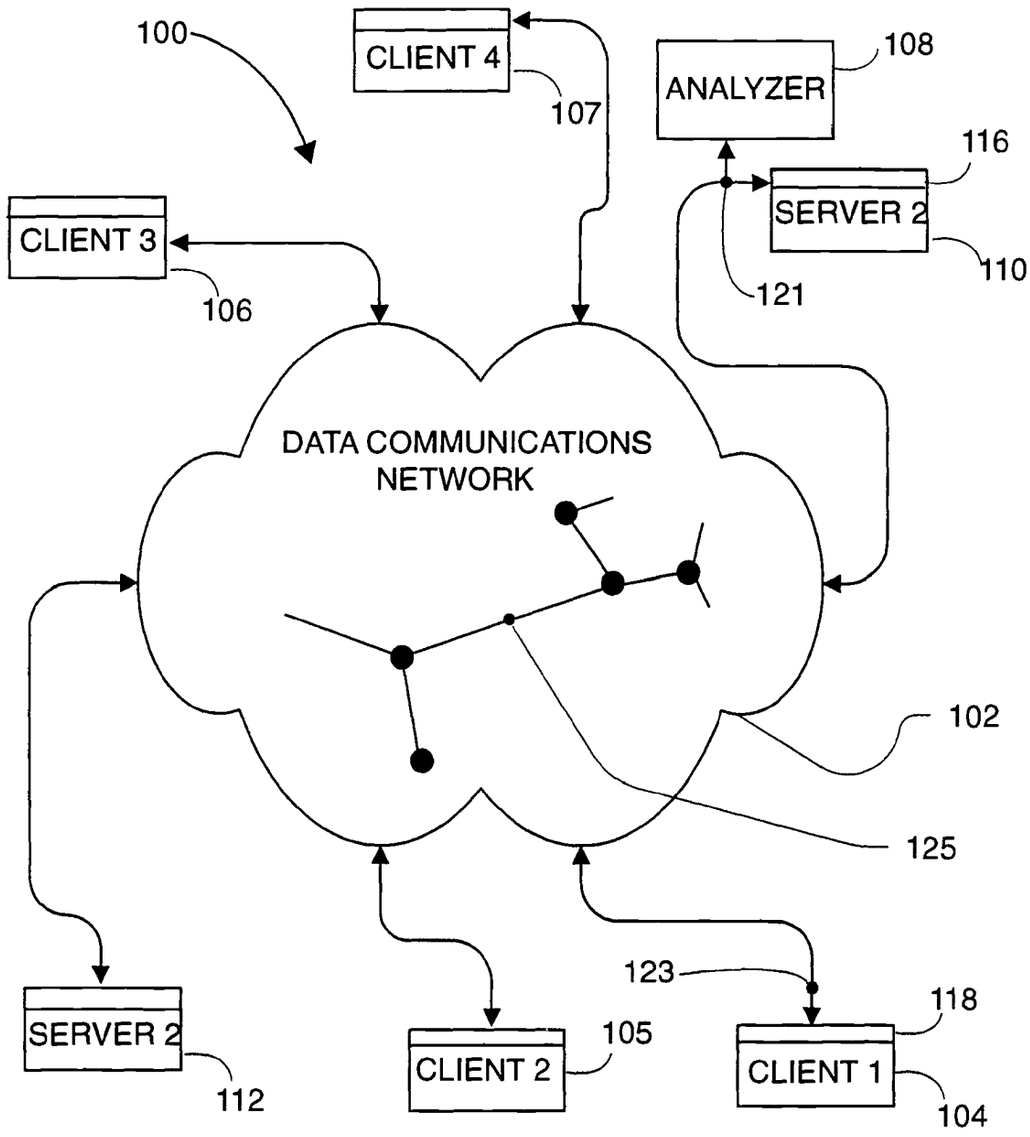


FIG. 1

PRINT OF DRAWINGS
AS ORIGINALLY FILED

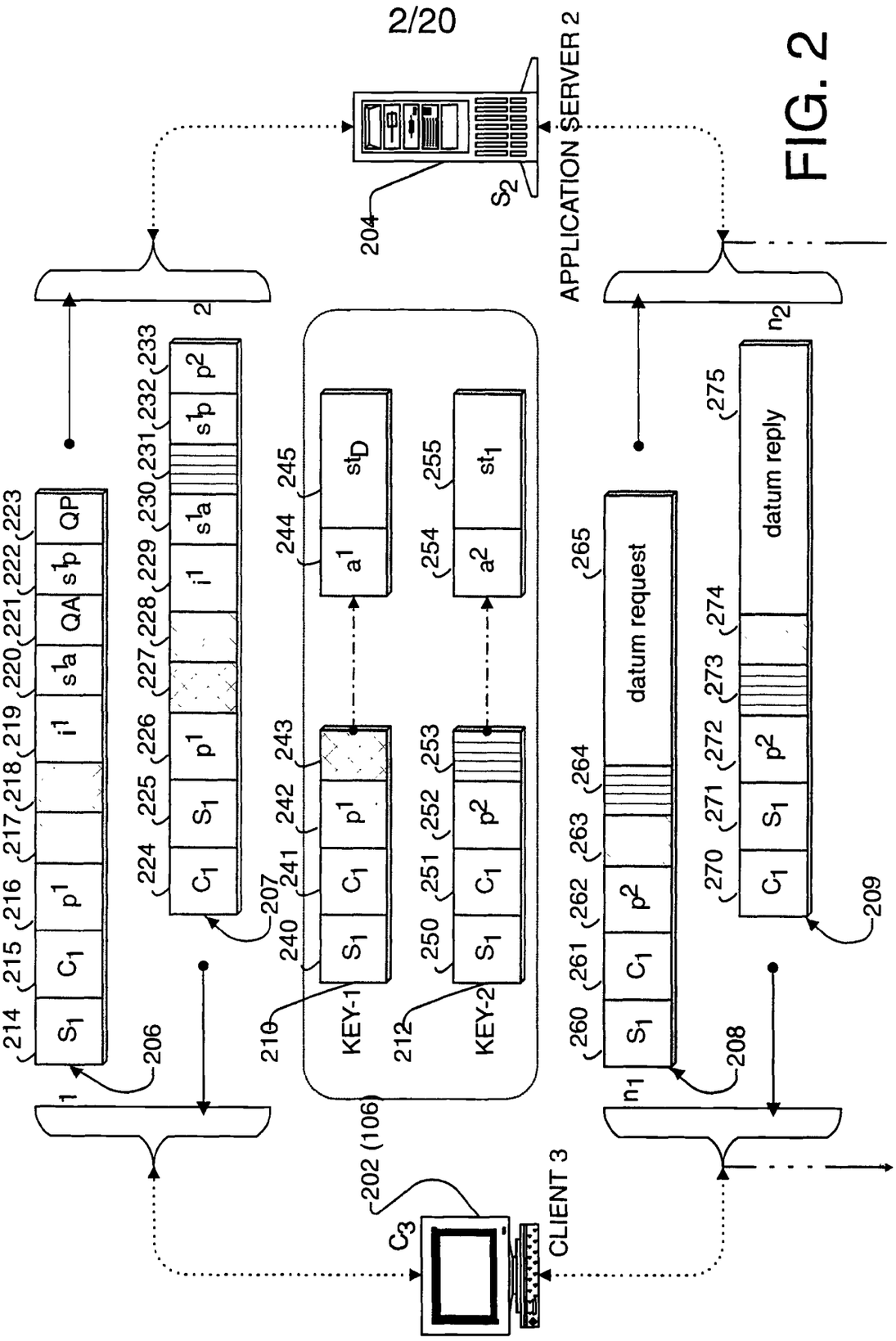


FIG. 2

PRINT OF DRAWINGS
AS ORIGINALLY FILED

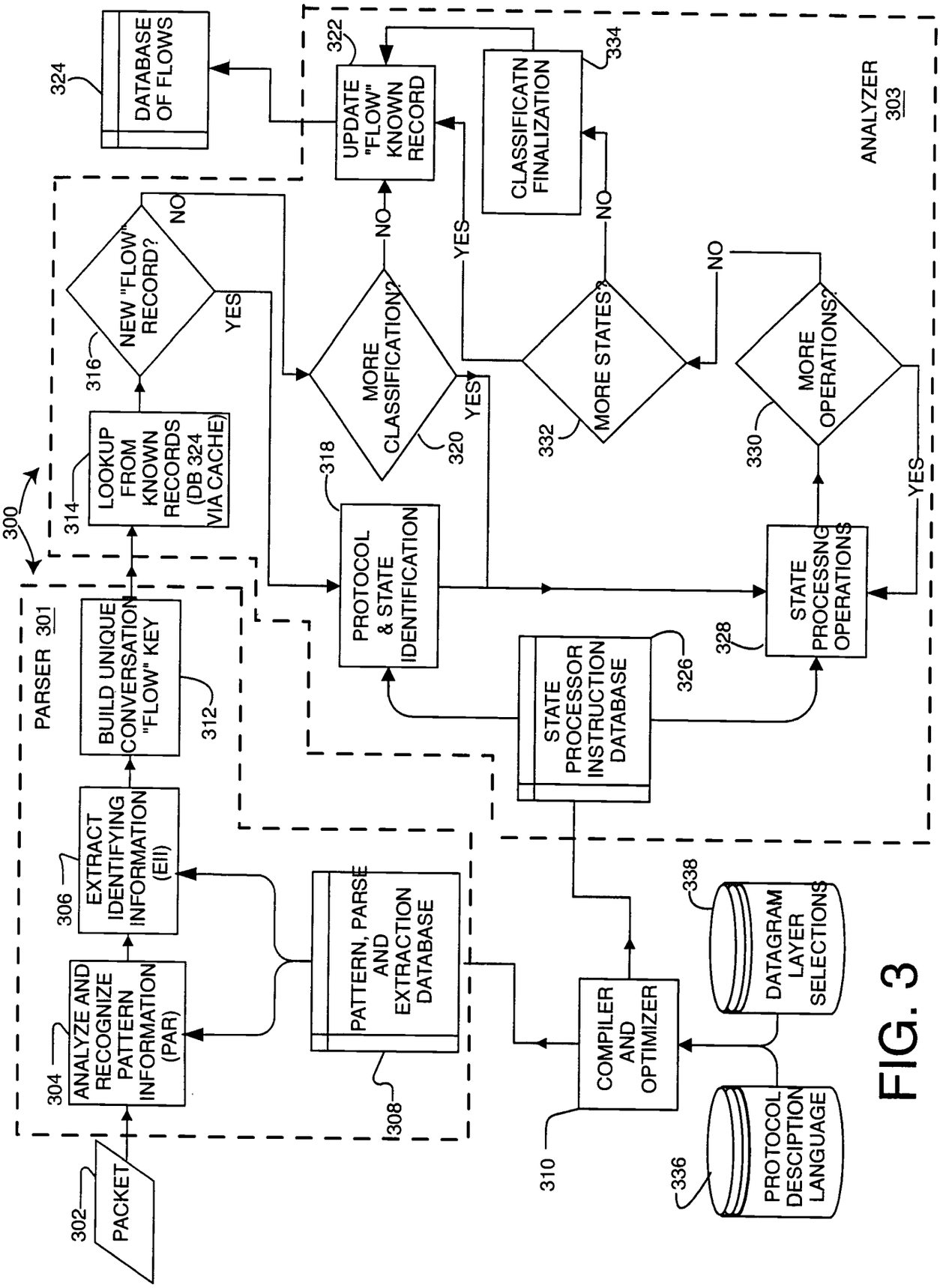


FIG. 3

PRINT OF DRAWINGS
AS ORIGINALLY FILED

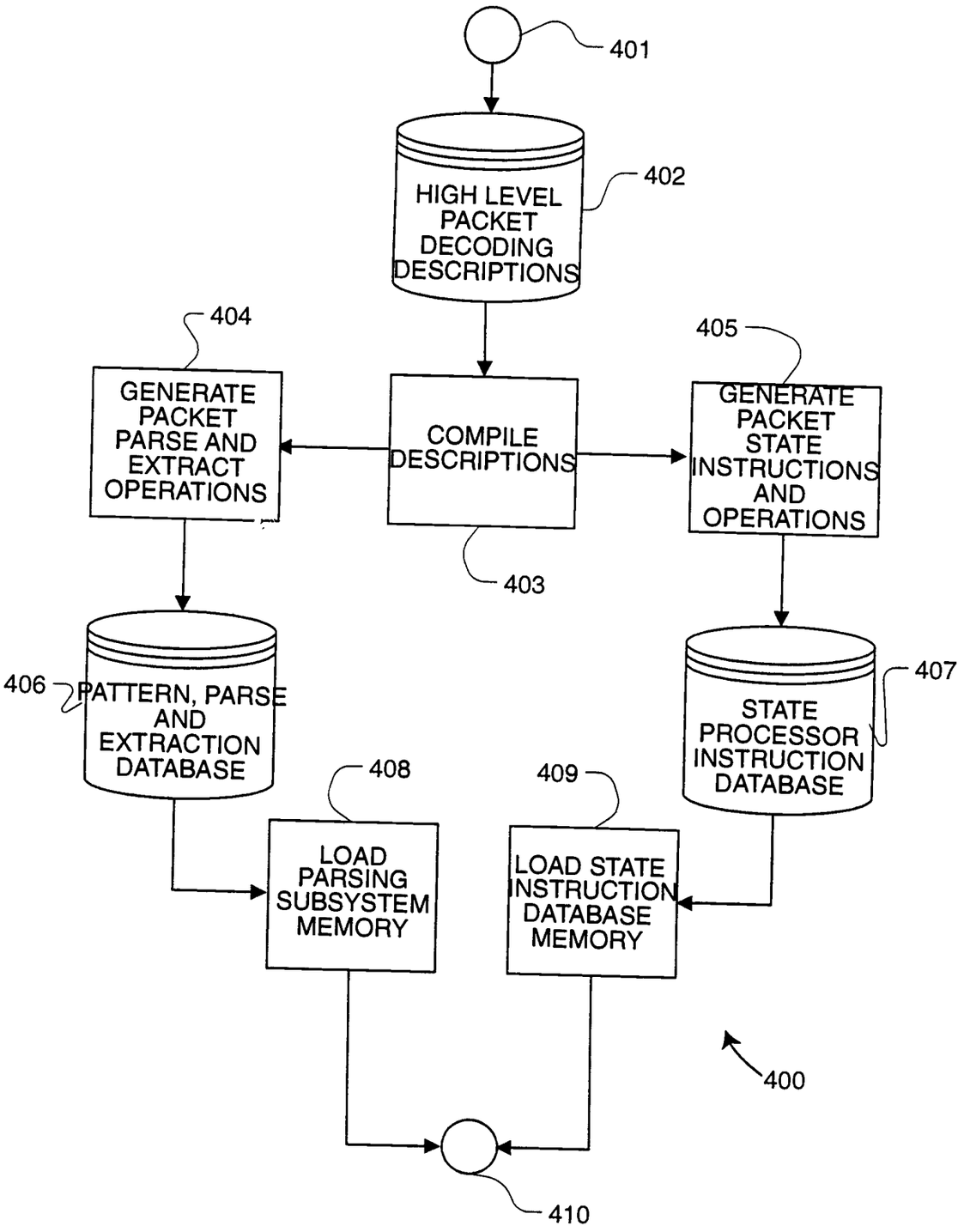


FIG. 4

Dietz
PRINT OF DRAWINGS
AS ORIGINALLY FILED

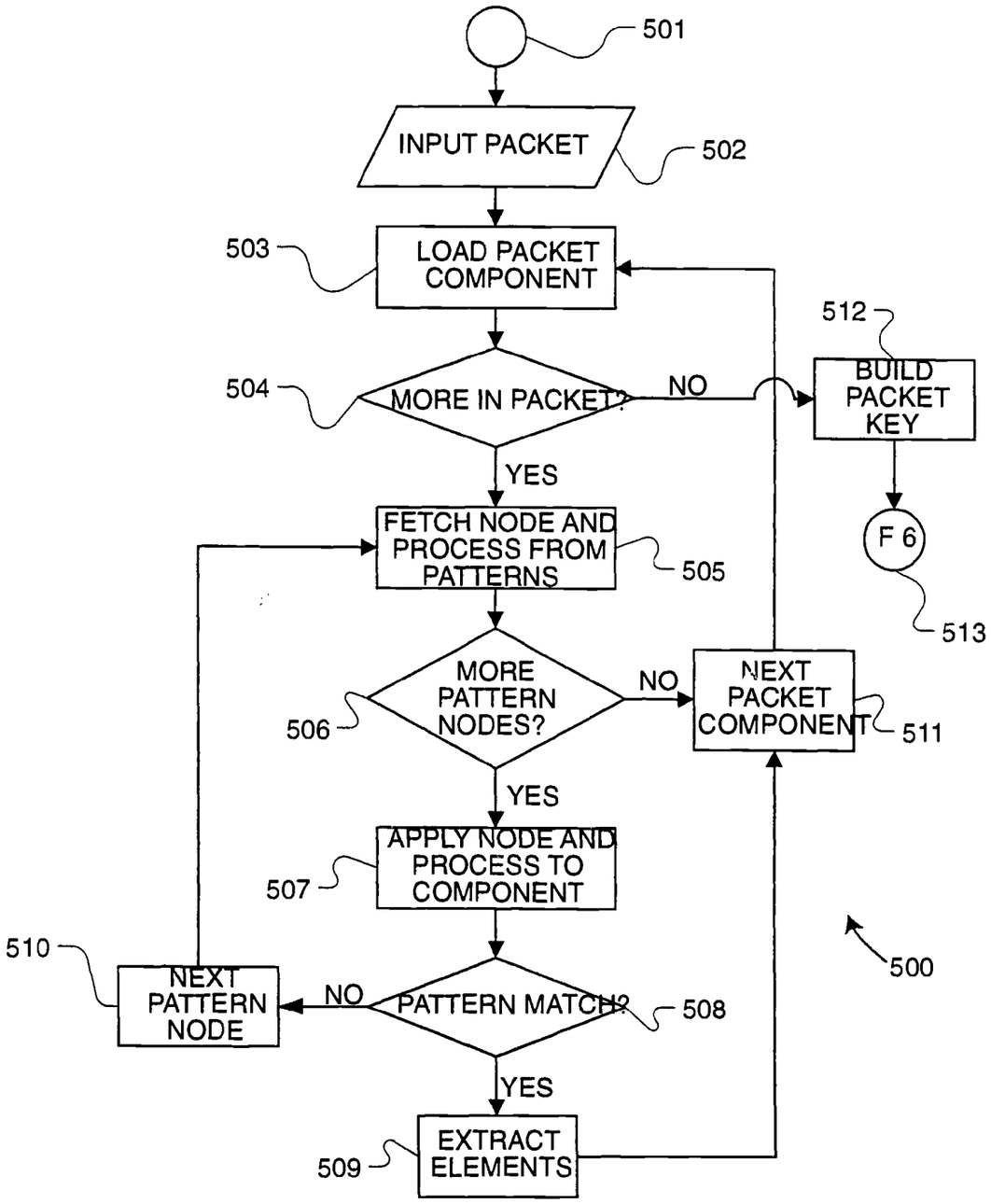


FIG. 5

PRINT OF DRAWINGS
AS ORIGINALLY FILED

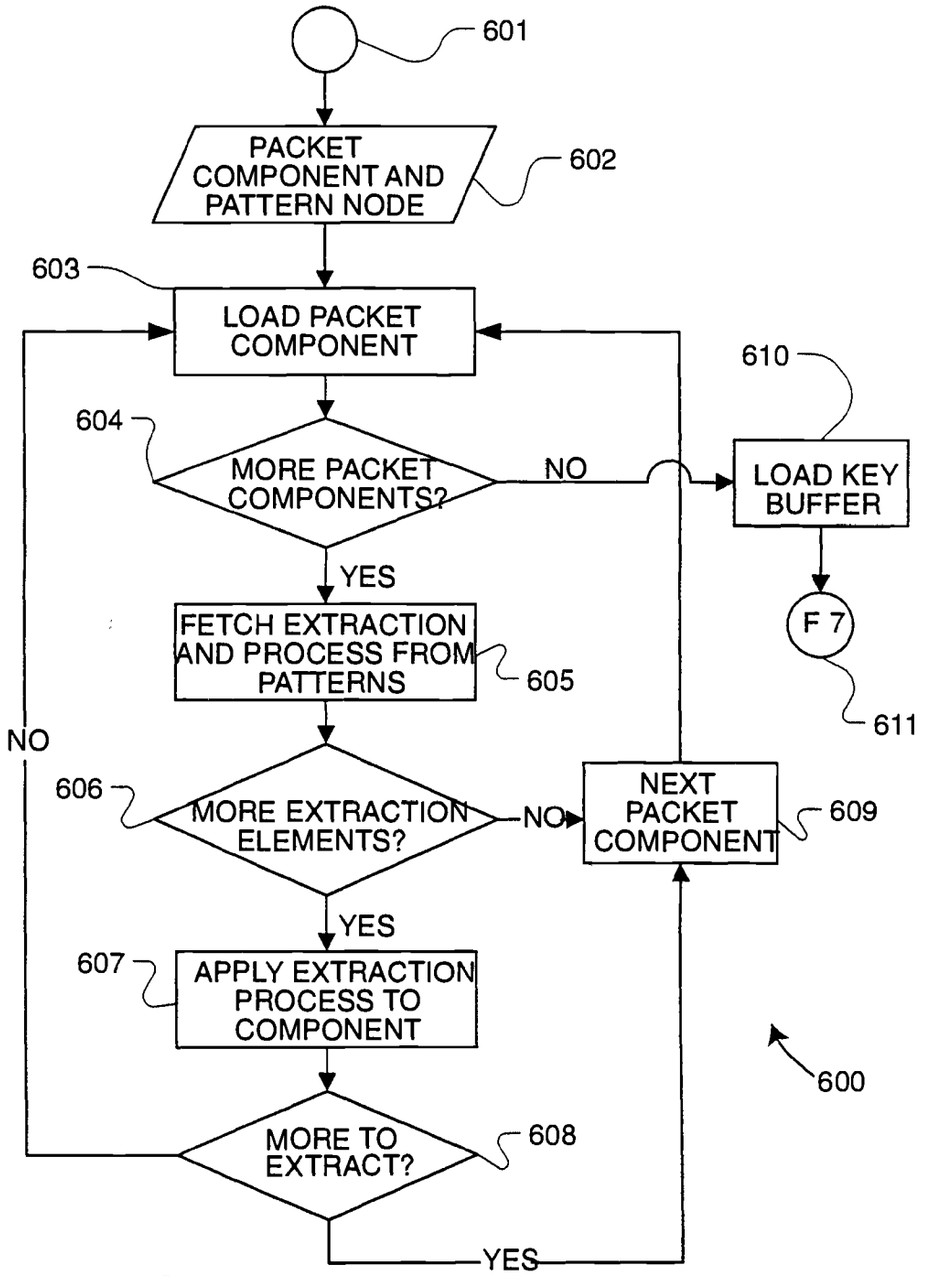


FIG. 6

PRINT OF DRAWINGS
AS ORIGINALLY FILED

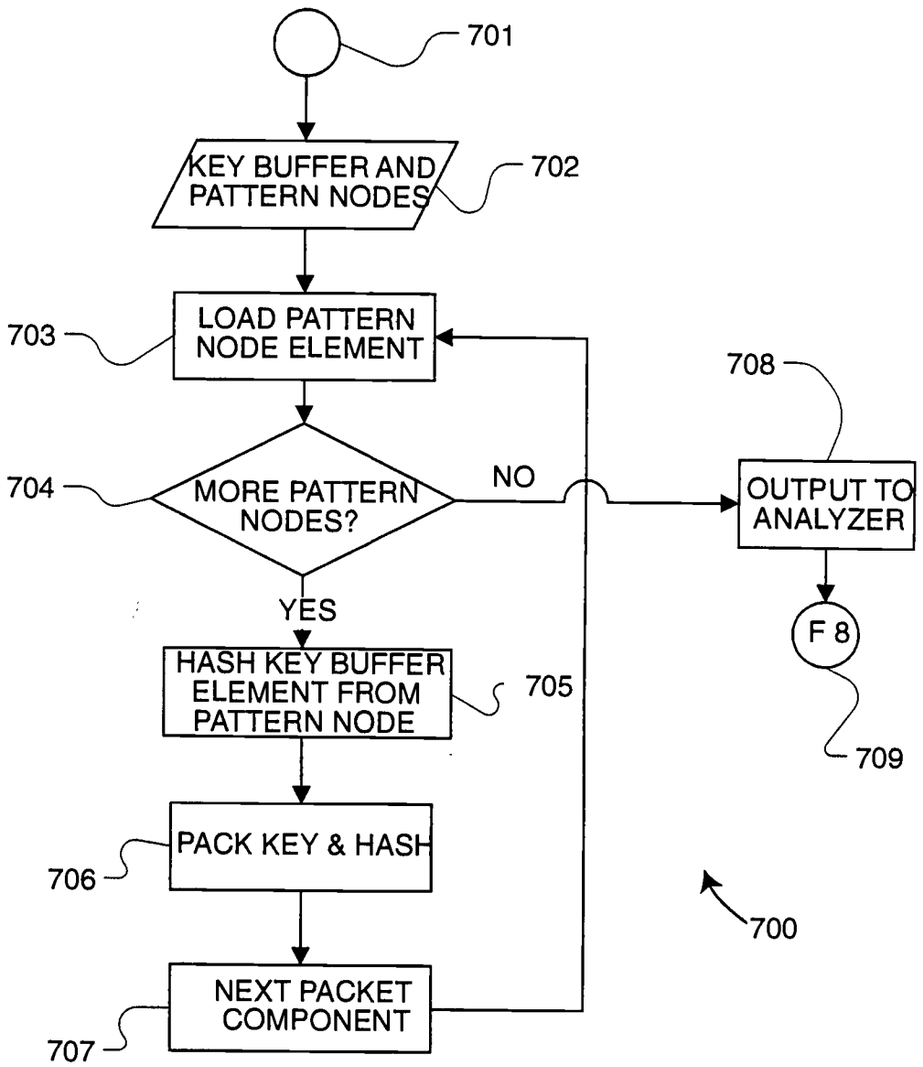


FIG. 7

PRINT OF DRAWINGS
AS ORIGINALLY FILED

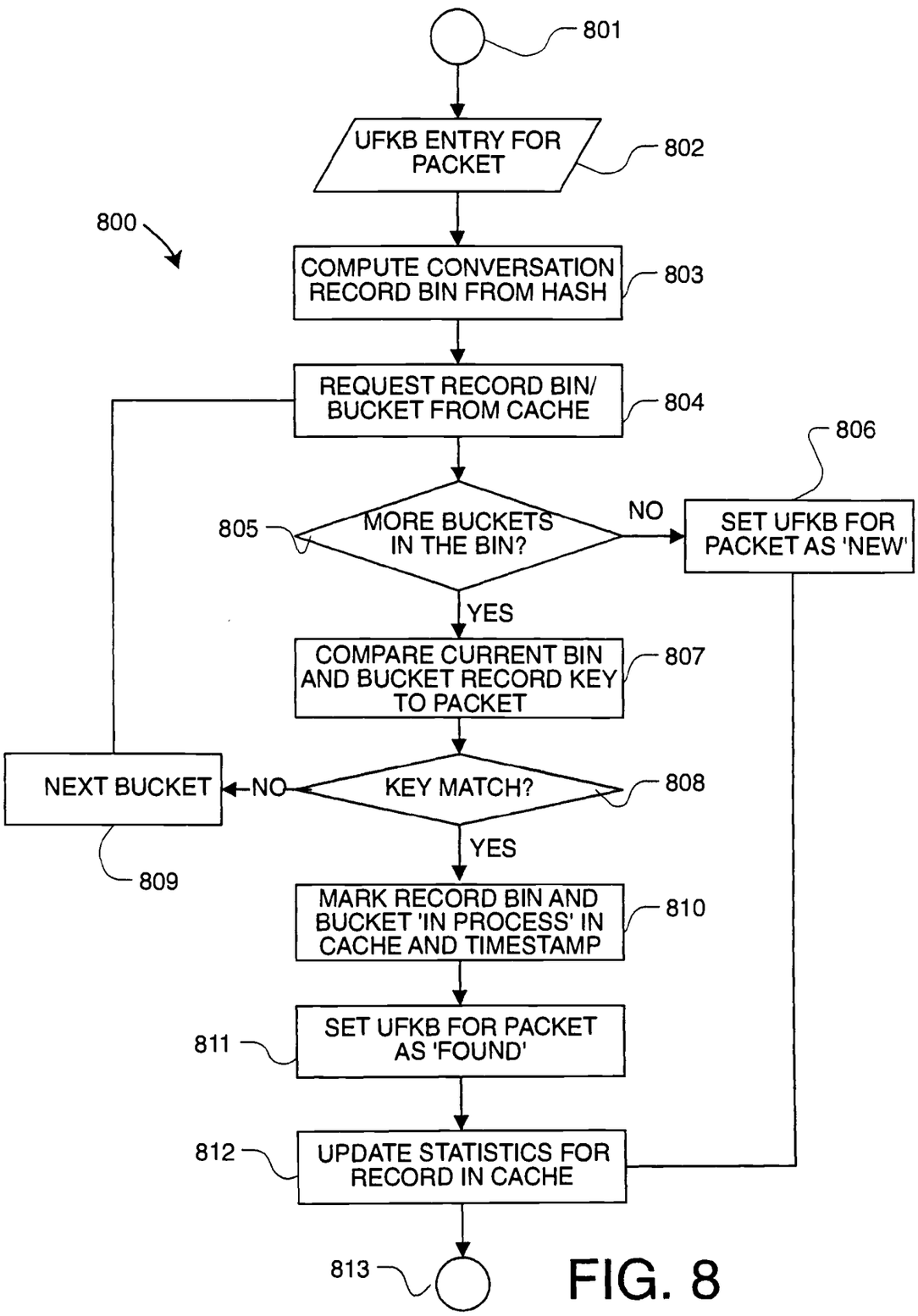


FIG. 8

PRINT OF DRAWINGS
AS ORIGINALLY FILED

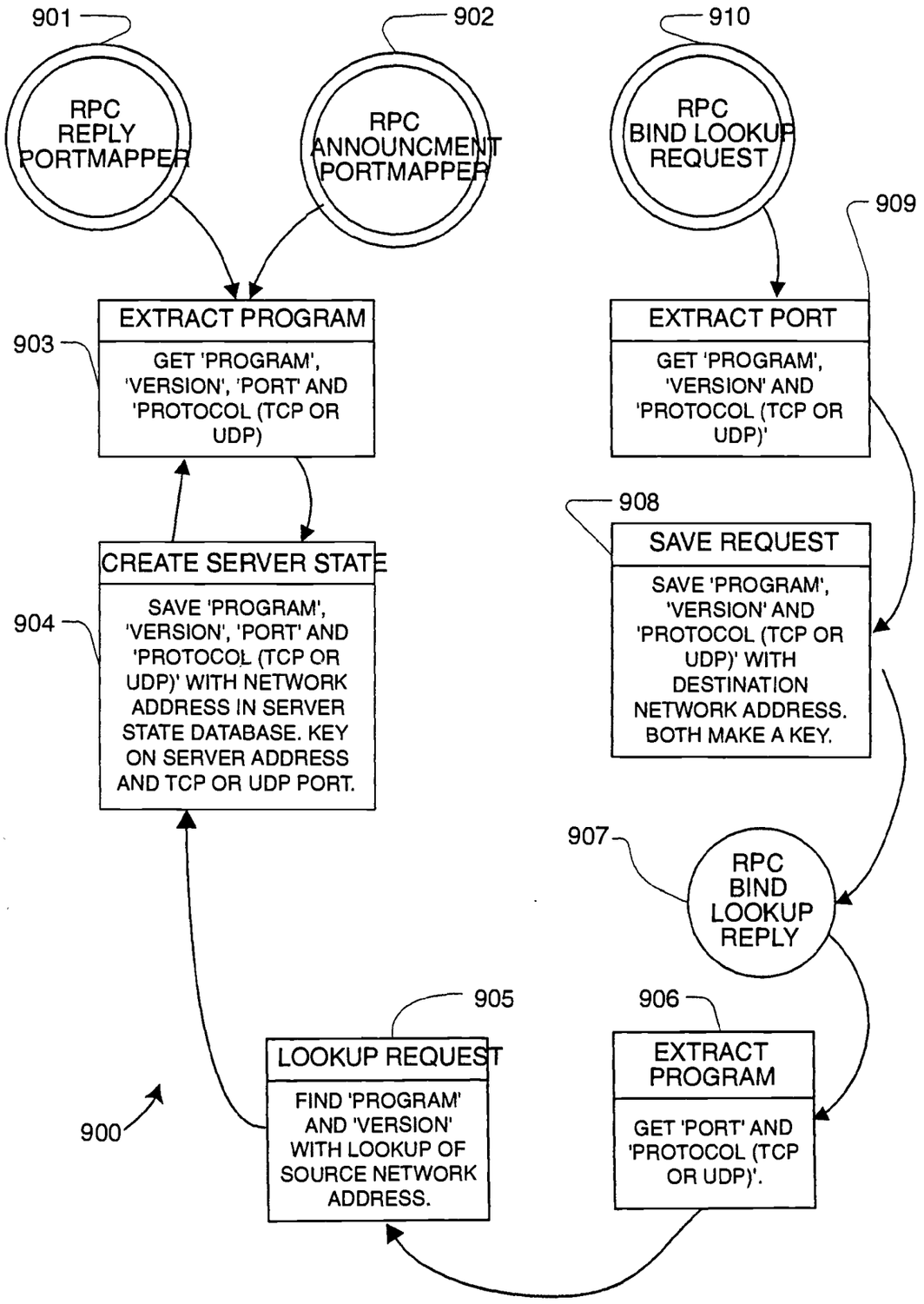


FIG. 9

PRINT OF DRAWINGS
AS ORIGINALLY FILED

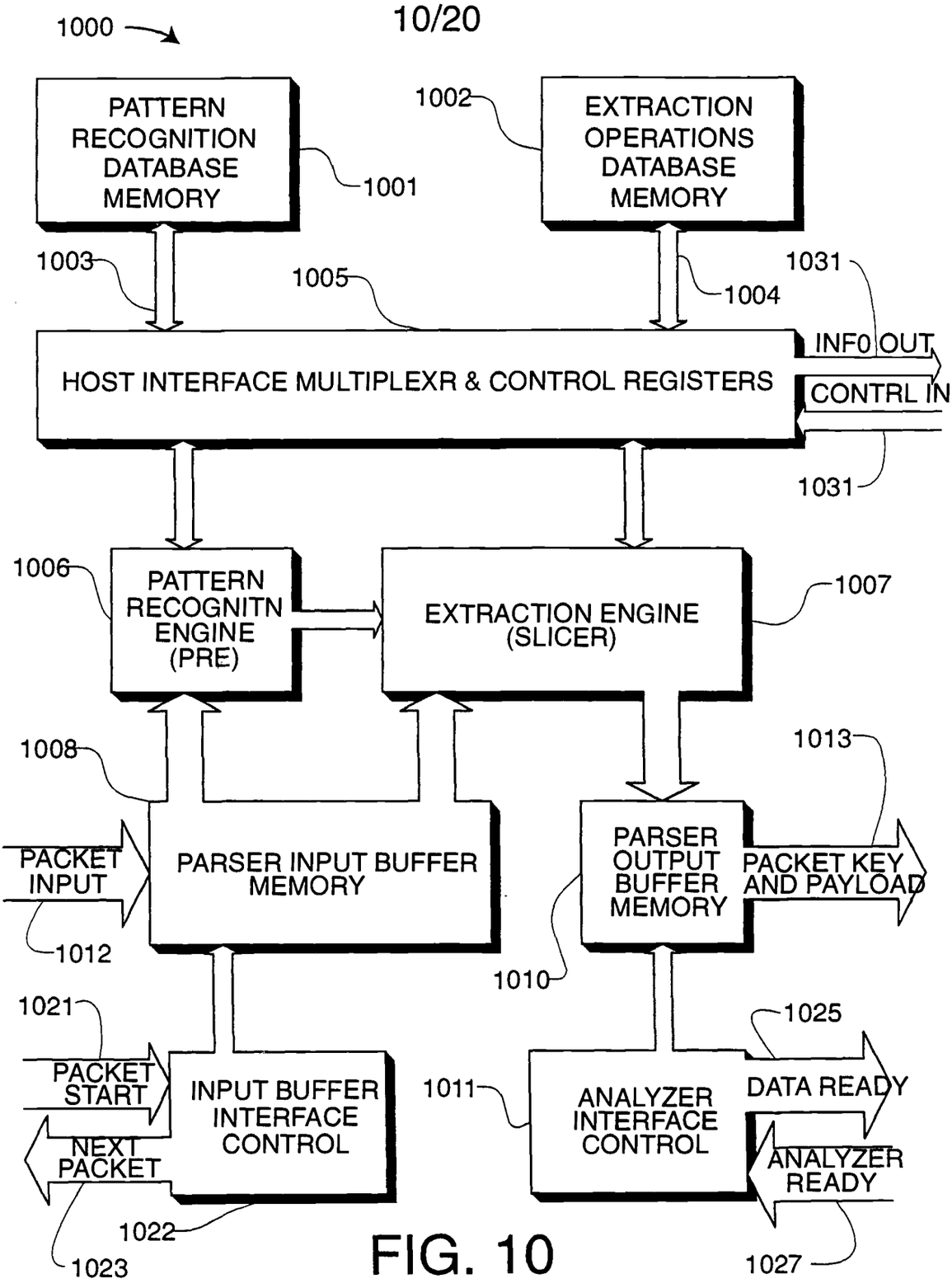


FIG. 10

PRINT OF DRAWINGS
AS ORIGINALLY FILED

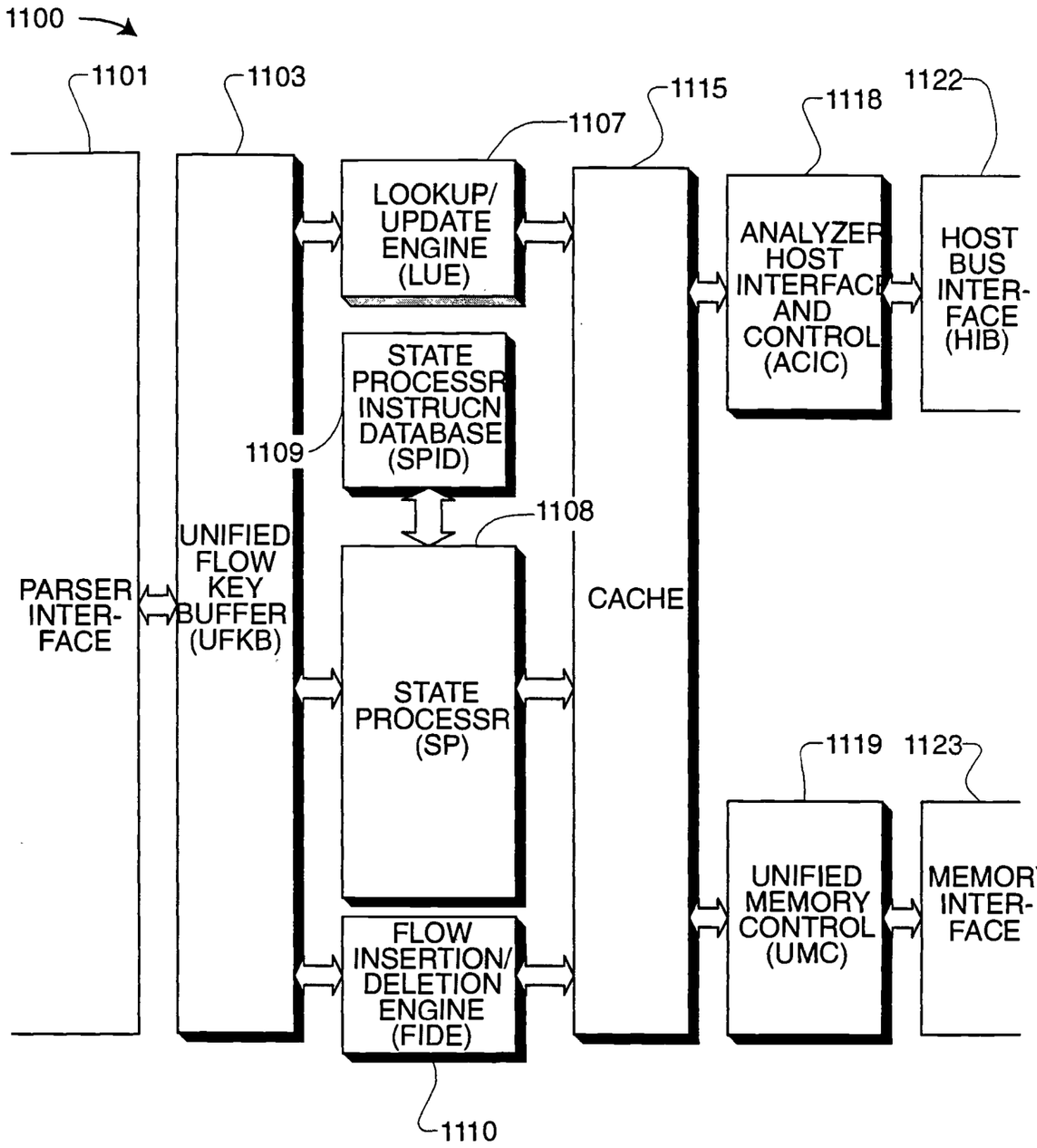


FIG. 11

PRINT OF DRAWINGS
AS ORIGINALLY FILED

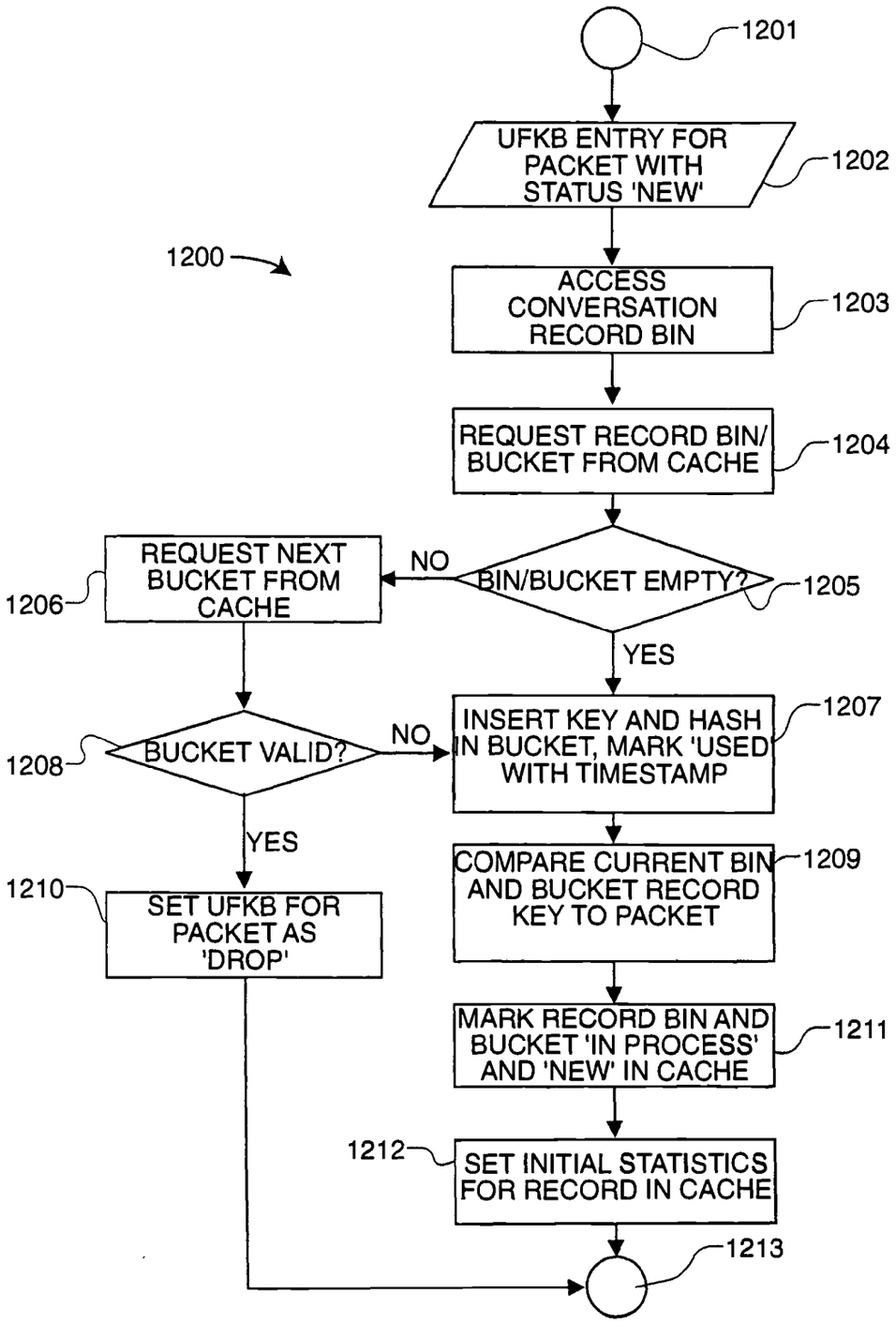


FIG. 12

PRINT OF DRAWINGS
AS ORIGINALLY FILED

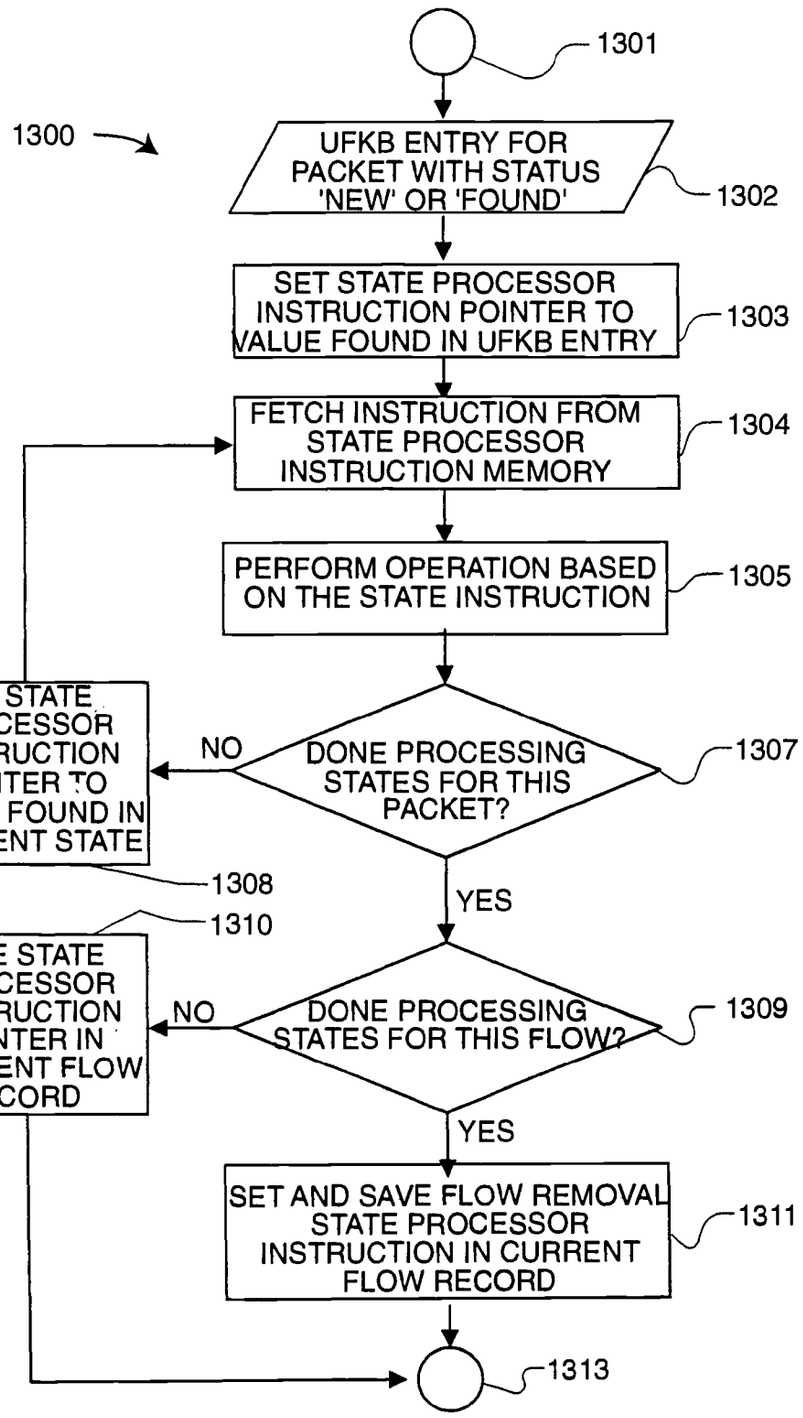


FIG. 13

PRINT OF DRAWINGS
AS ORIGINALLY FILED

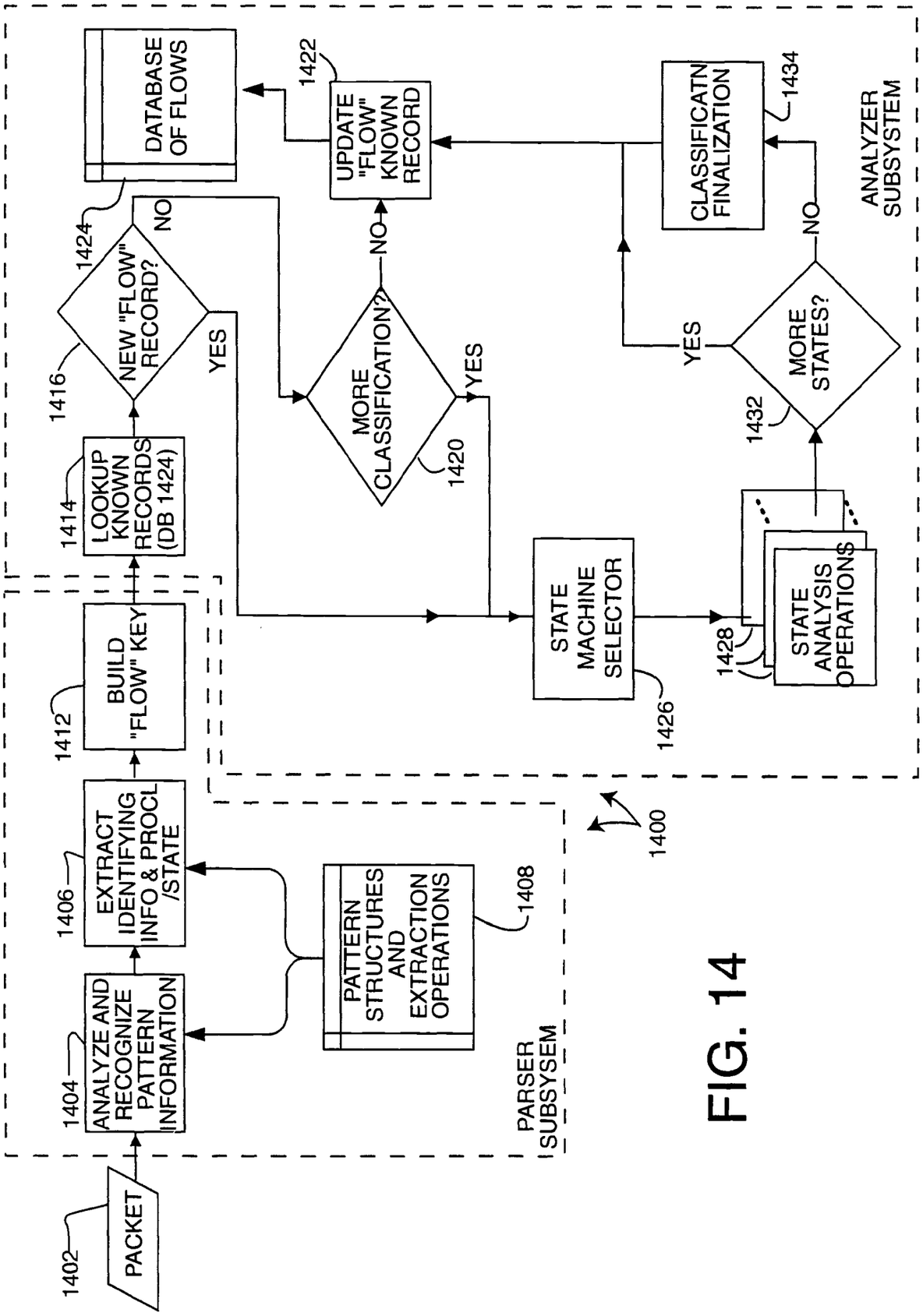


FIG. 14

PRINT OF DRAWINGS
AS ORIGINALLY FILED

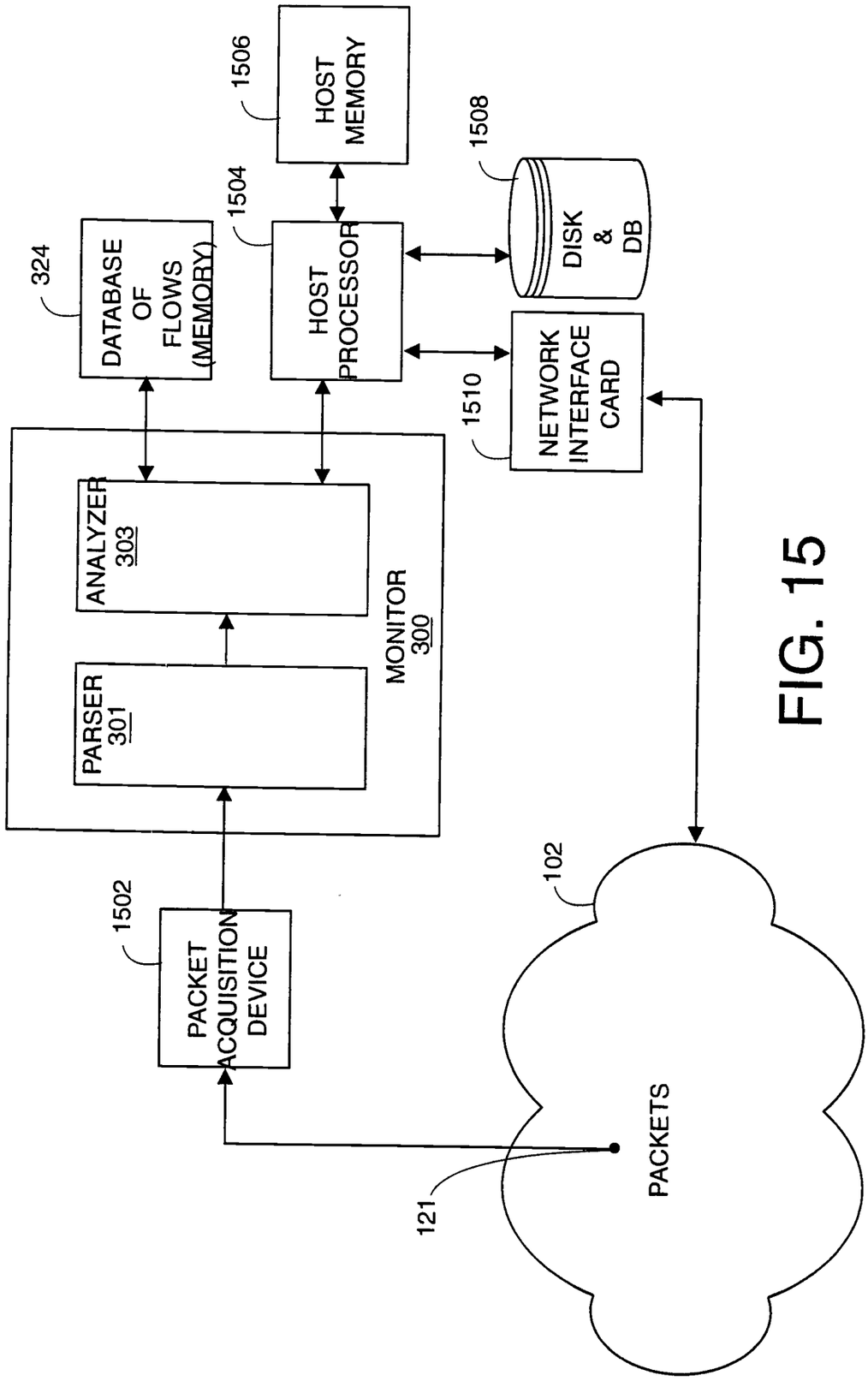


FIG. 15

PRINT OF DRAWINGS
AS ORIGINALLY FILED

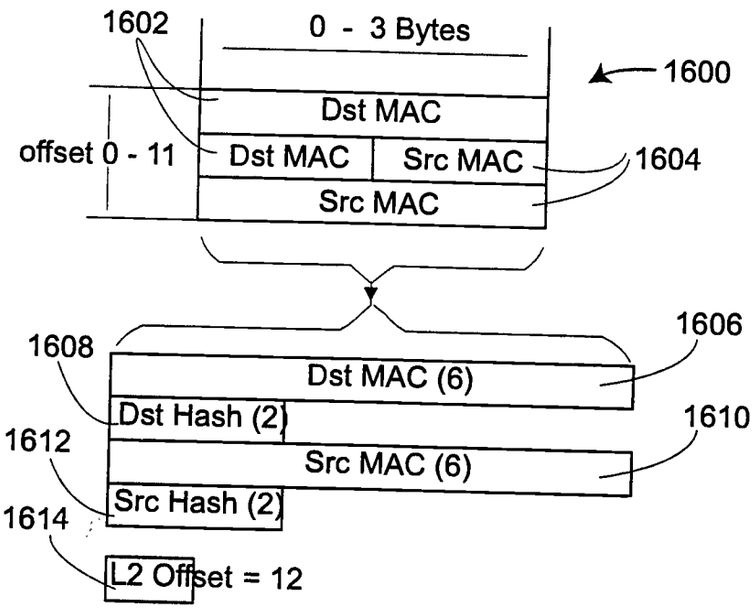


FIG. 16

Dietz
PRINT OF DRAWINGS
AS ORIGINALLY FILED

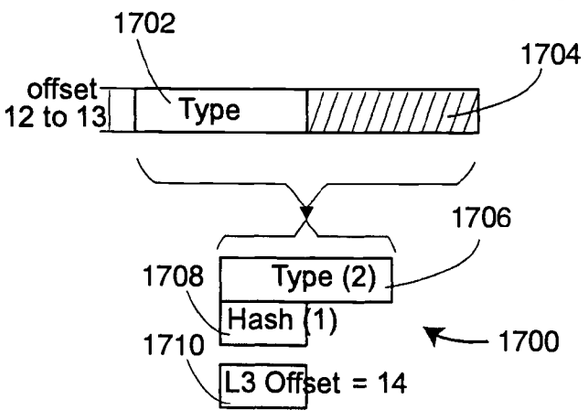


FIG. 17A

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

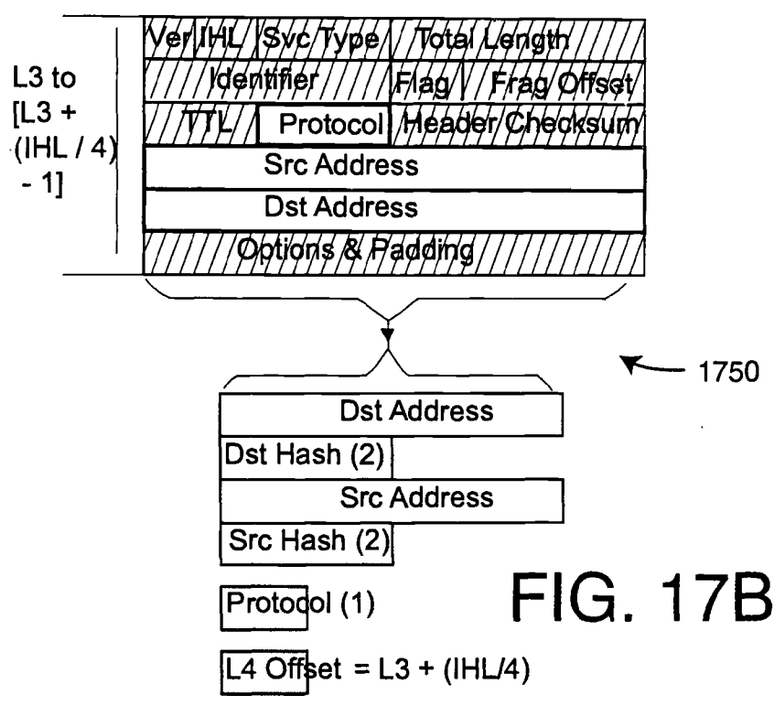


FIG. 17B

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

PRINT OF DRAWINGS
AS ORIGINALLY FILED

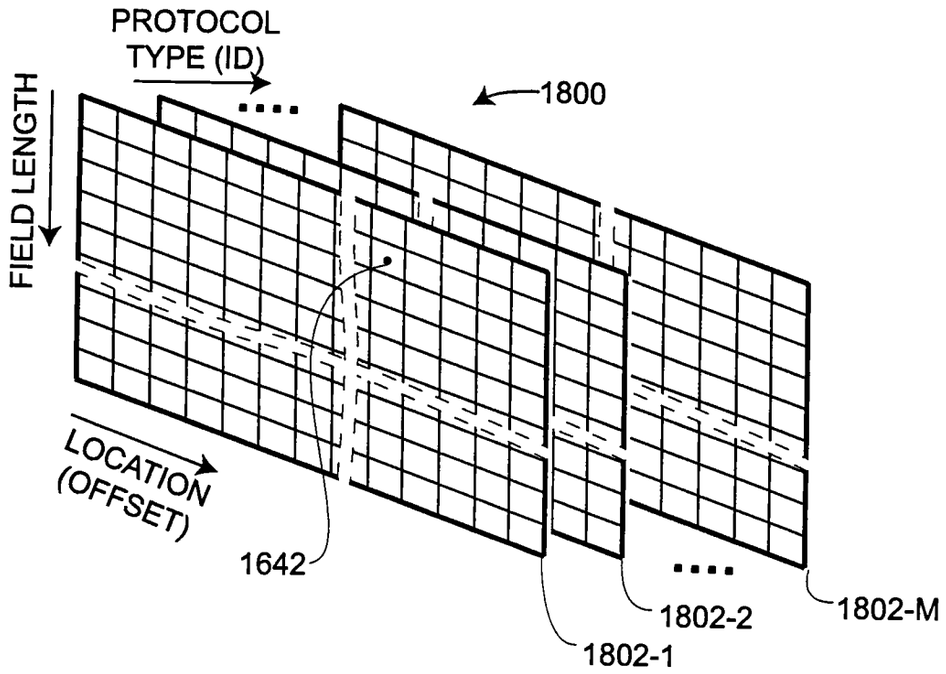


FIG. 18A

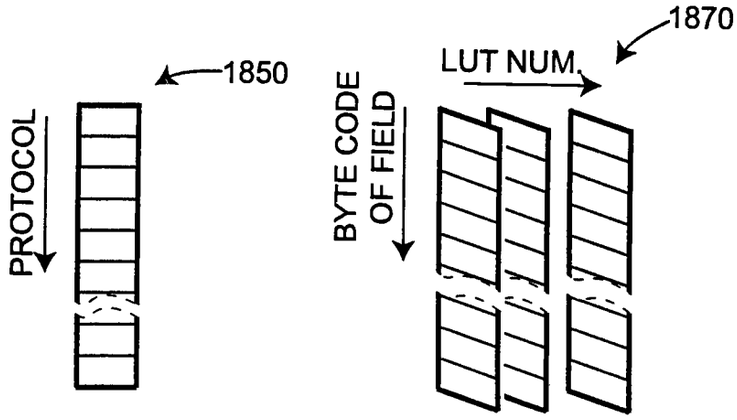


FIG. 18B

PRINT OF DRAWINGS
AS ORIGINALLY FILED

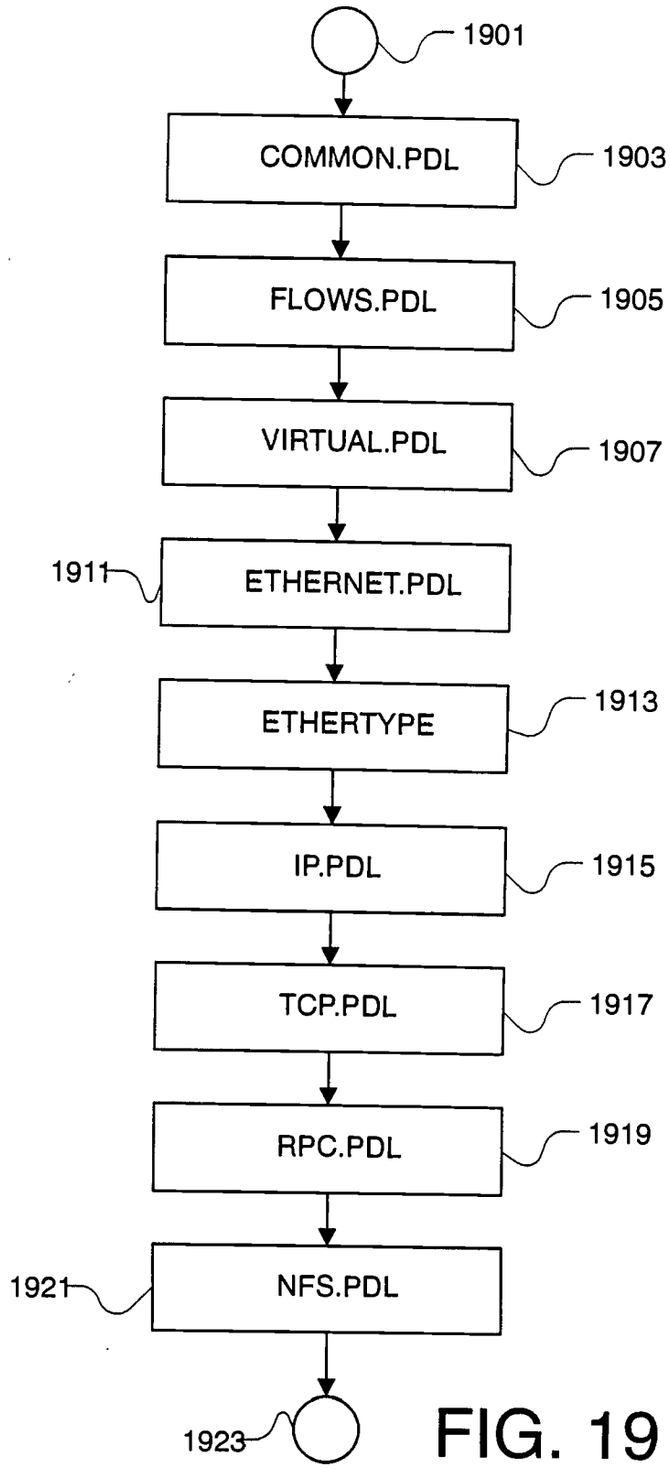


FIG. 19

PRINT OF DRAWINGS
AS ORIGINALLY FILED

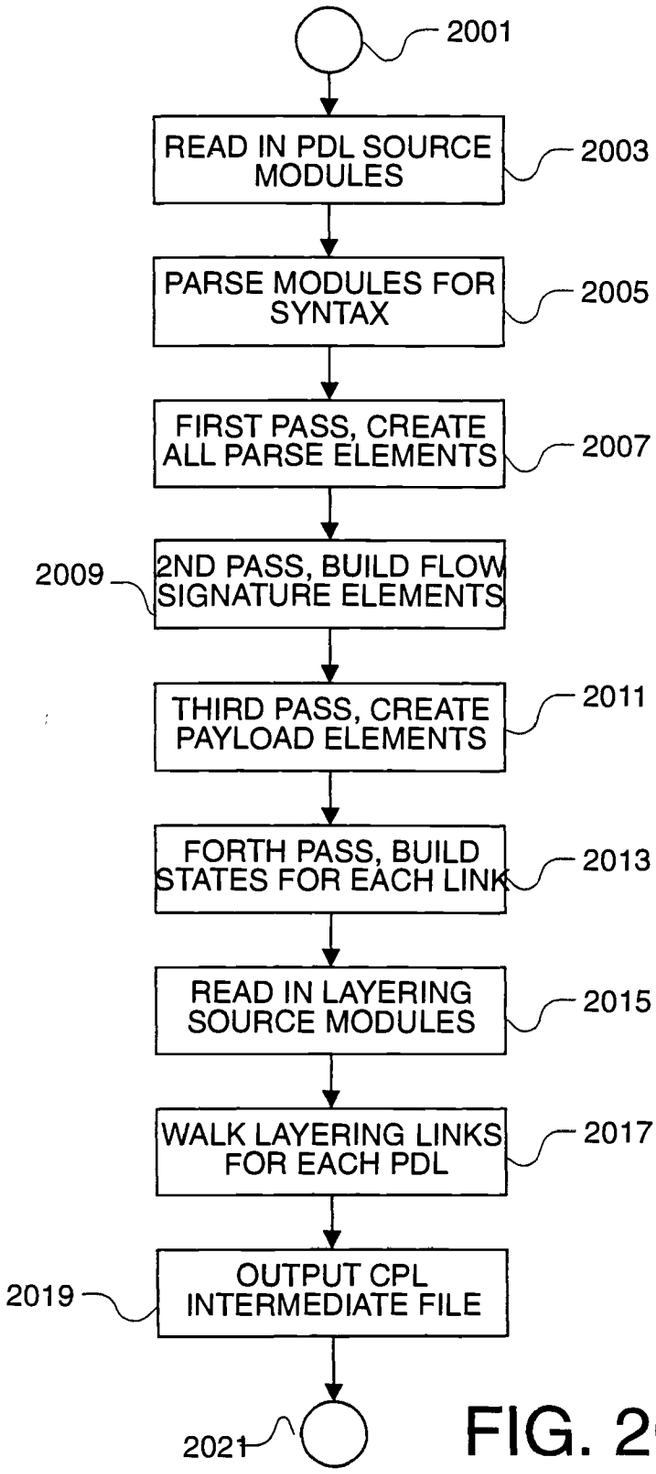


FIG. 20

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

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

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

LETTER TO OFFICIAL DRAFTSPERSON SUBMISSION OF FORMAL DRAWINGS

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

Dear Sir or Madam:

Attached please find 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

709 230

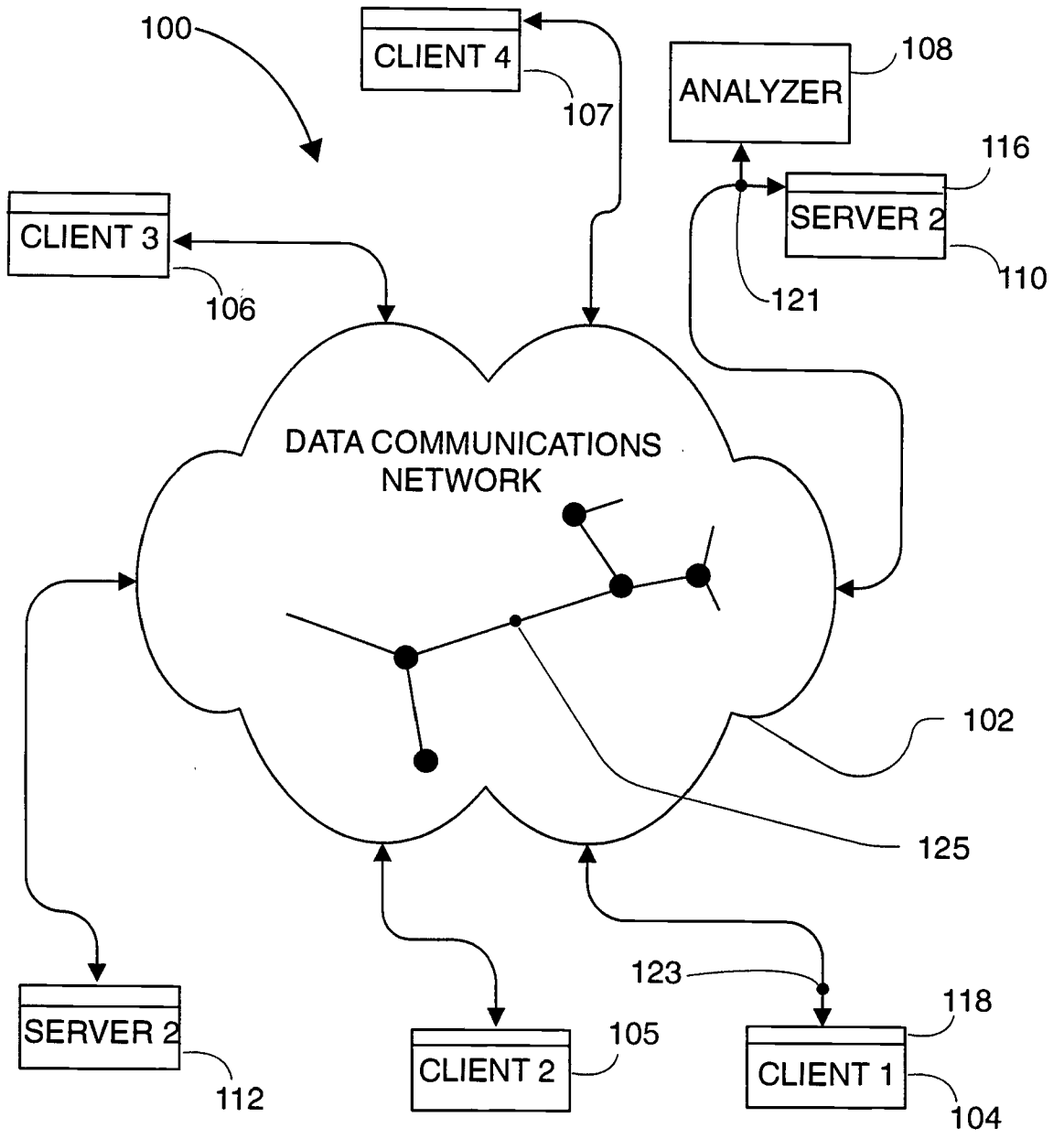

100
CLIENT 4
107
ANALYZER
108
116
SERVER 2
110
121
CLIENT 3
106
DATA COMMUNICATIONS NETWORK
102
125
123
118
SERVER 2
112
CLIENT 2
105
CLIENT 1
104

FIG. 1

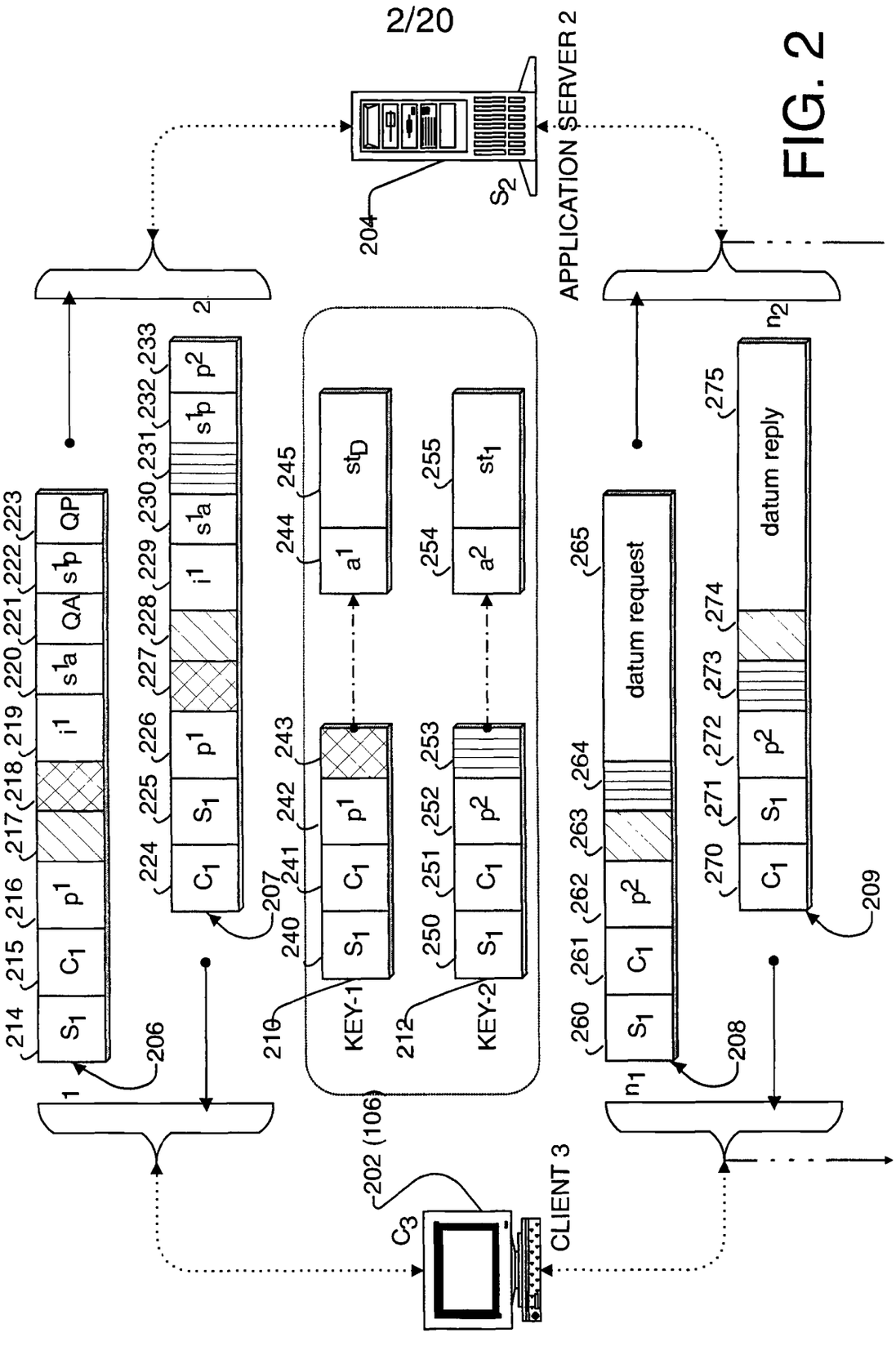

APPLICATION SERVER 2
S_2
204
CLIENT 3
C_3
202 (106)
KEY-1
KEY-2
datum request
datum reply
QP
S_1
C_1
p^1
i^1
s^1a
QA
s^1p
p^2
i^1
s^1a
s^1p
a^1
st_D
a^2
st_1
1
2
n_1
n_2
206
207
208
209
210
212

FIG. 2

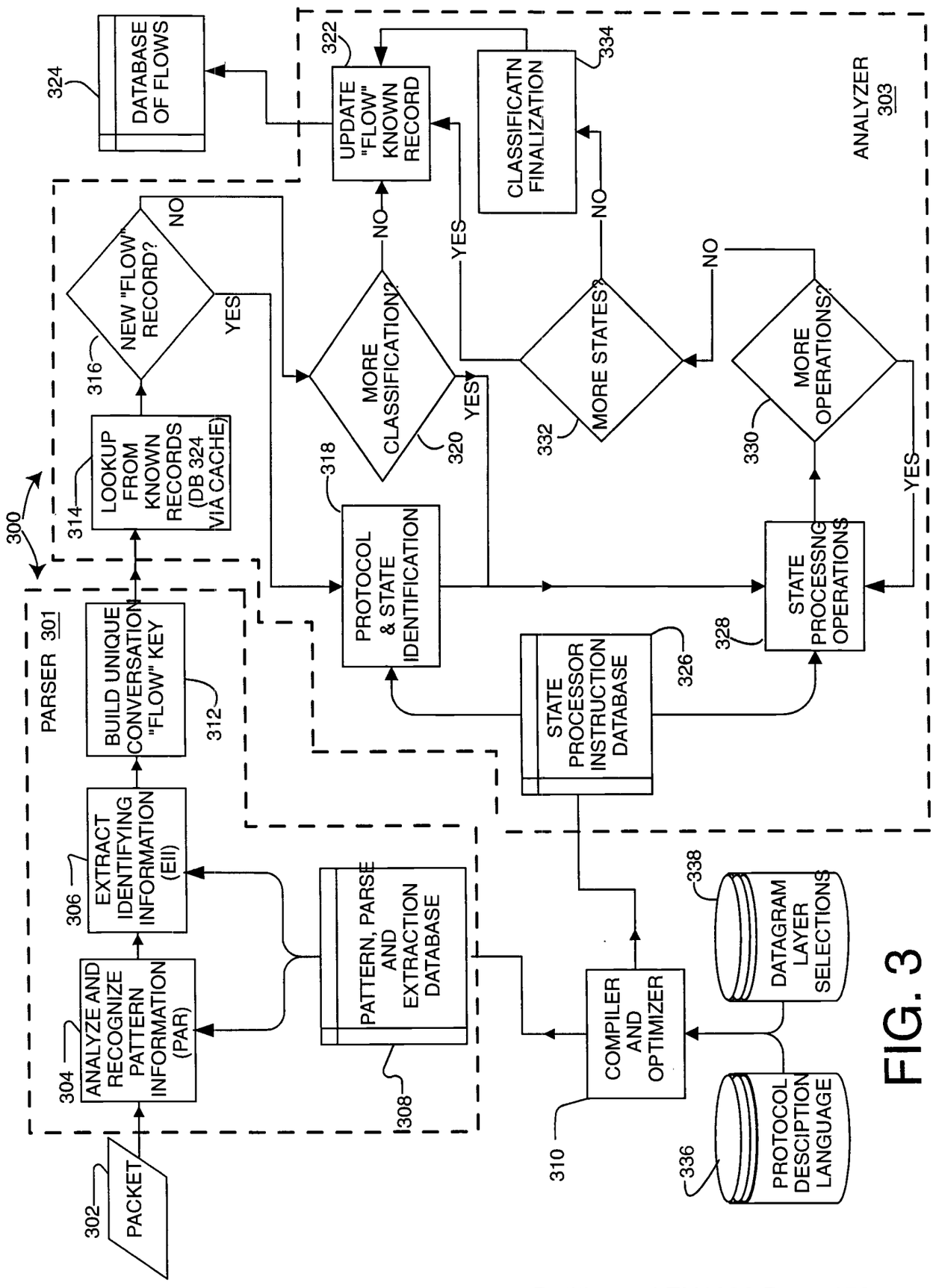

PACKET
302
PARSER 301
ANALYZE AND RECOGNIZE PATTERN INFORMATION (PAR)
304
EXTRACT IDENTIFYING INFORMATION (EII)
306
BUILD UNIQUE CONVERSATION "FLOW" KEY
312
PATTERN, PARSE AND EXTRACTION DATABASE
308
COMPILER AND OPTIMIZER
310
PROTOCOL DESCIPTION LANGUAGE
336
DATAGRAM LAYER SELECTIONS
338
ANALYZER 303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

FIG. 3

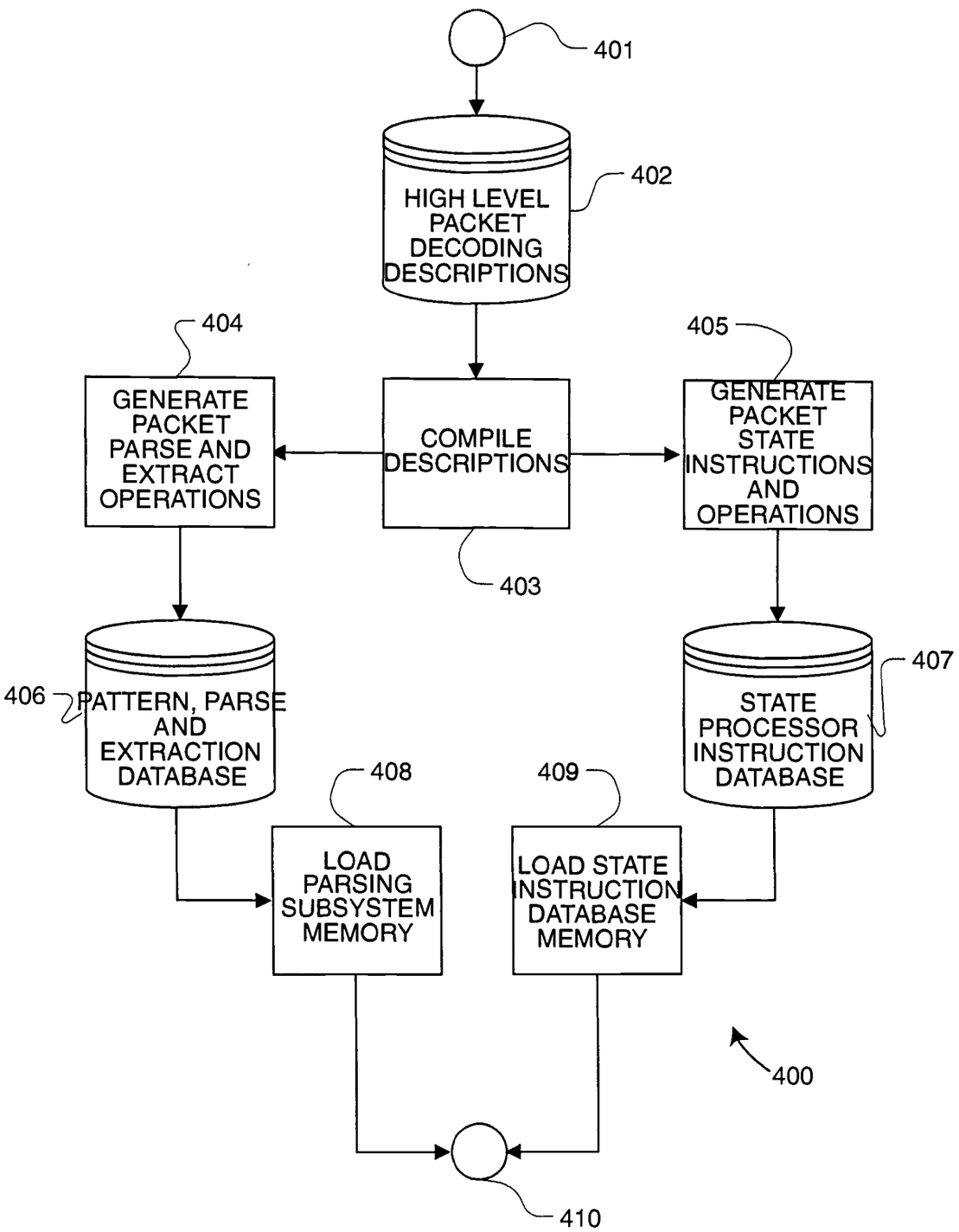

401
HIGH LEVEL PACKET DECODING DESCRIPTIONS
402
404
405
GENERATE PACKET PARSE AND EXTRACT OPERATIONS
COMPILE DESCRIPTIONS
GENERATE PACKET STATE INSTRUCTIONS AND OPERATIONS
403
406
PATTERN, PARSE AND EXTRACTION DATABASE
407
STATE PROCESSOR INSTRUCTION DATABASE
408
409
LOAD PARSING SUBSYSTEM MEMORY
LOAD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

FIG. 4

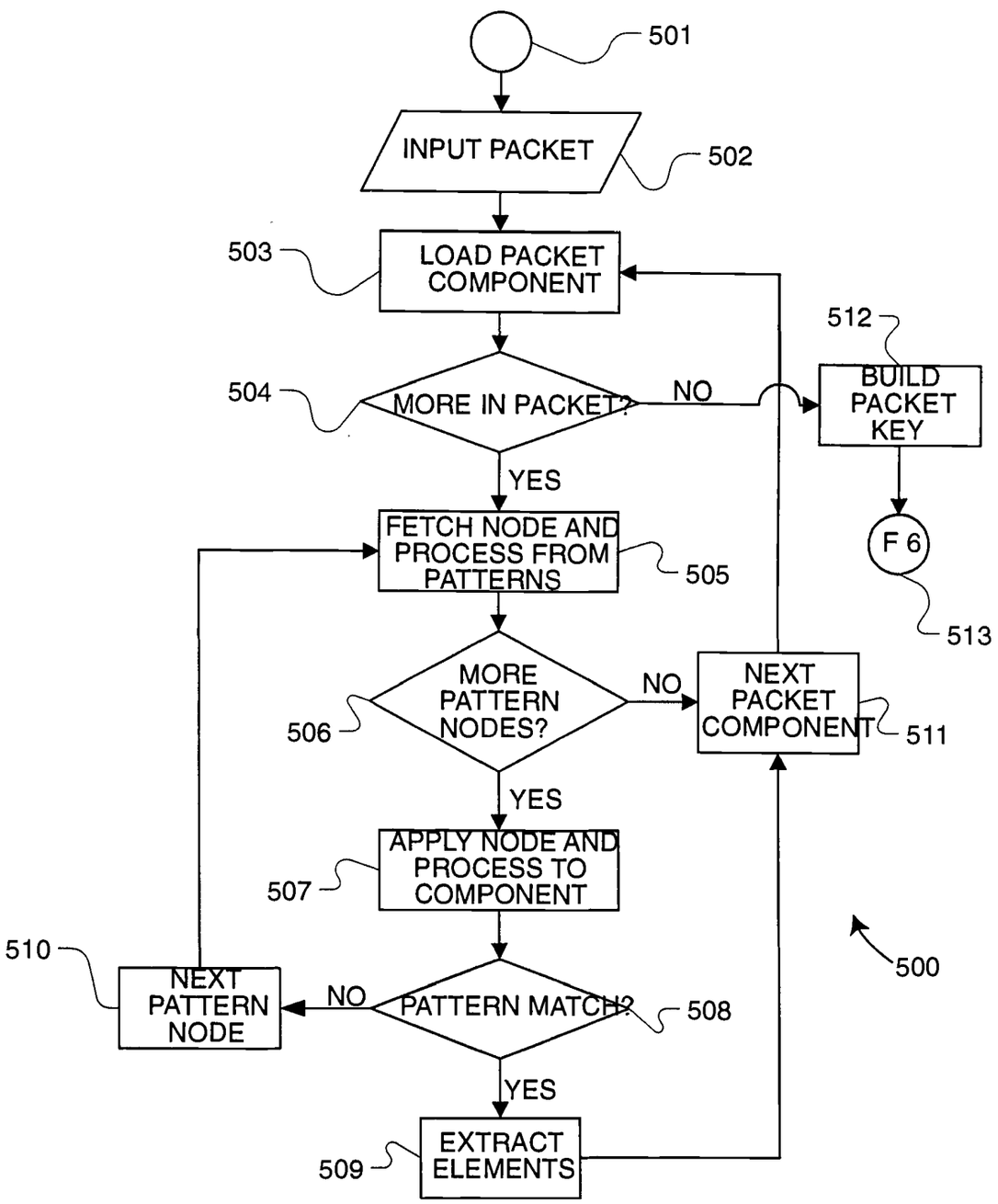

501
INPUT PACKET
502
503
LOAD PACKET COMPONENT
512
504
MORE IN PACKET?
NO
BUILD PACKET KEY
YES
F 6
513
FETCH NODE AND PROCESS FROM PATTERNS
505
MORE PATTERN NODES?
506
NO
NEXT PACKET COMPONENT
511
YES
APPLY NODE AND PROCESS TO COMPONENT
507
500
510
NEXT PATTERN NODE
NO
PATTERN MATCH?
508
YES
EXTRACT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

FIG. 5

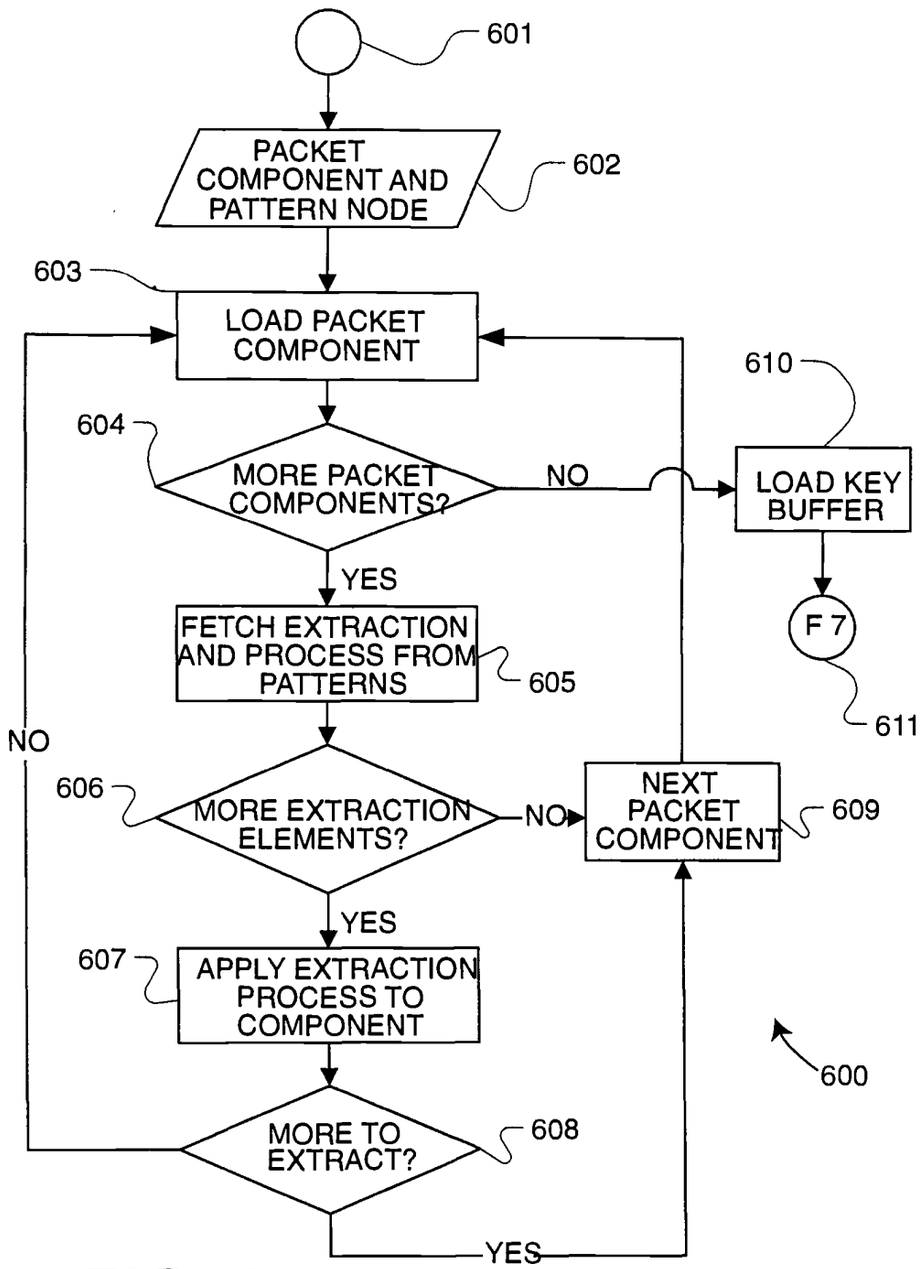

601
PACKET COMPONENT AND PATTERN NODE
602
603
LOAD PACKET COMPONENT
604
MORE PACKET COMPONENTS?
NO
610
LOAD KEY BUFFER
F 7
611
YES
FETCH EXTRACTION AND PROCESS FROM PATTERNS
605
NO
606
MORE EXTRACTION ELEMENTS?
NO
NEXT PACKET COMPONENT
609
YES
607
APPLY EXTRACTION PROCESS TO COMPONENT
600
MORE TO EXTRACT?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

FIG. 6

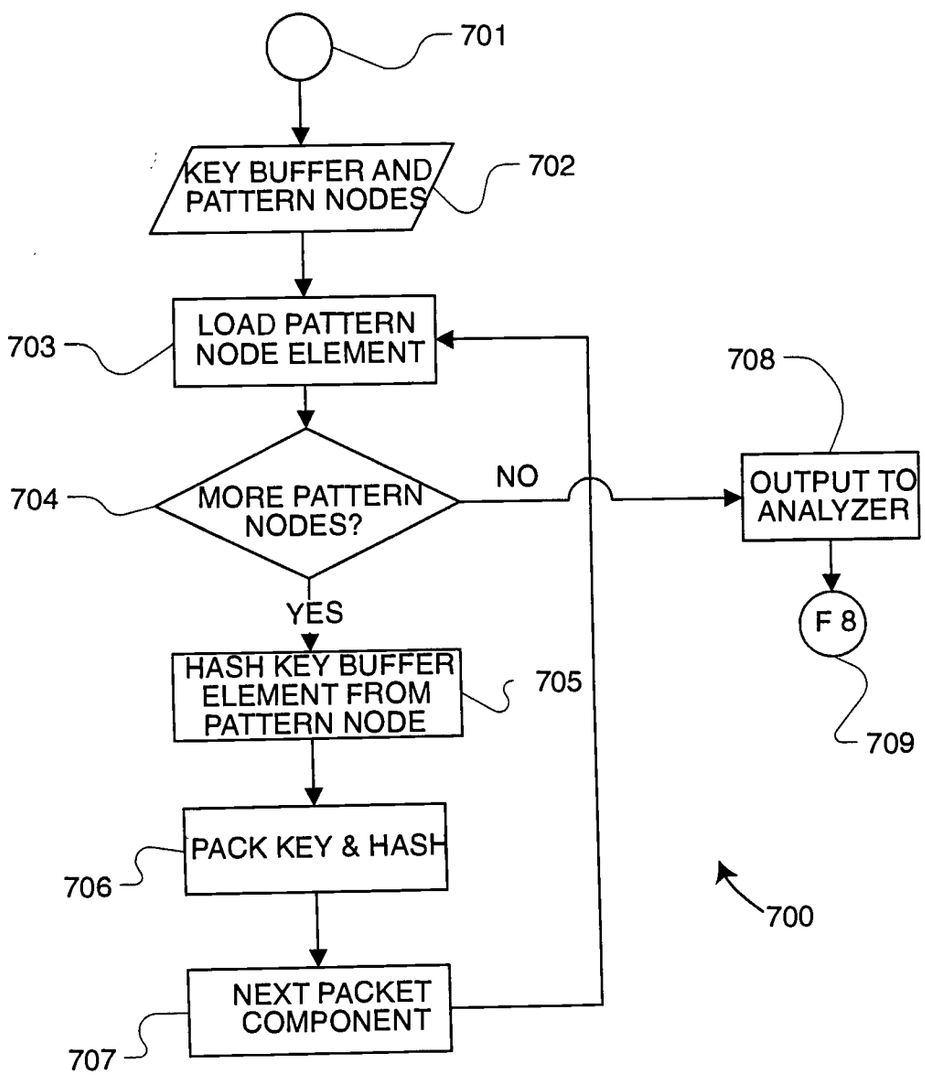

701
KEY BUFFER AND PATTERN NODES
702
LOAD PATTERN NODE ELEMENT
703
MORE PATTERN NODES?
704
NO
YES
OUTPUT TO ANALYZER
708
F 8
709
HASH KEY BUFFER ELEMENT FROM PATTERN NODE
705
PACK KEY & HASH
706
NEXT PACKET COMPONENT
707
700

FIG. 7

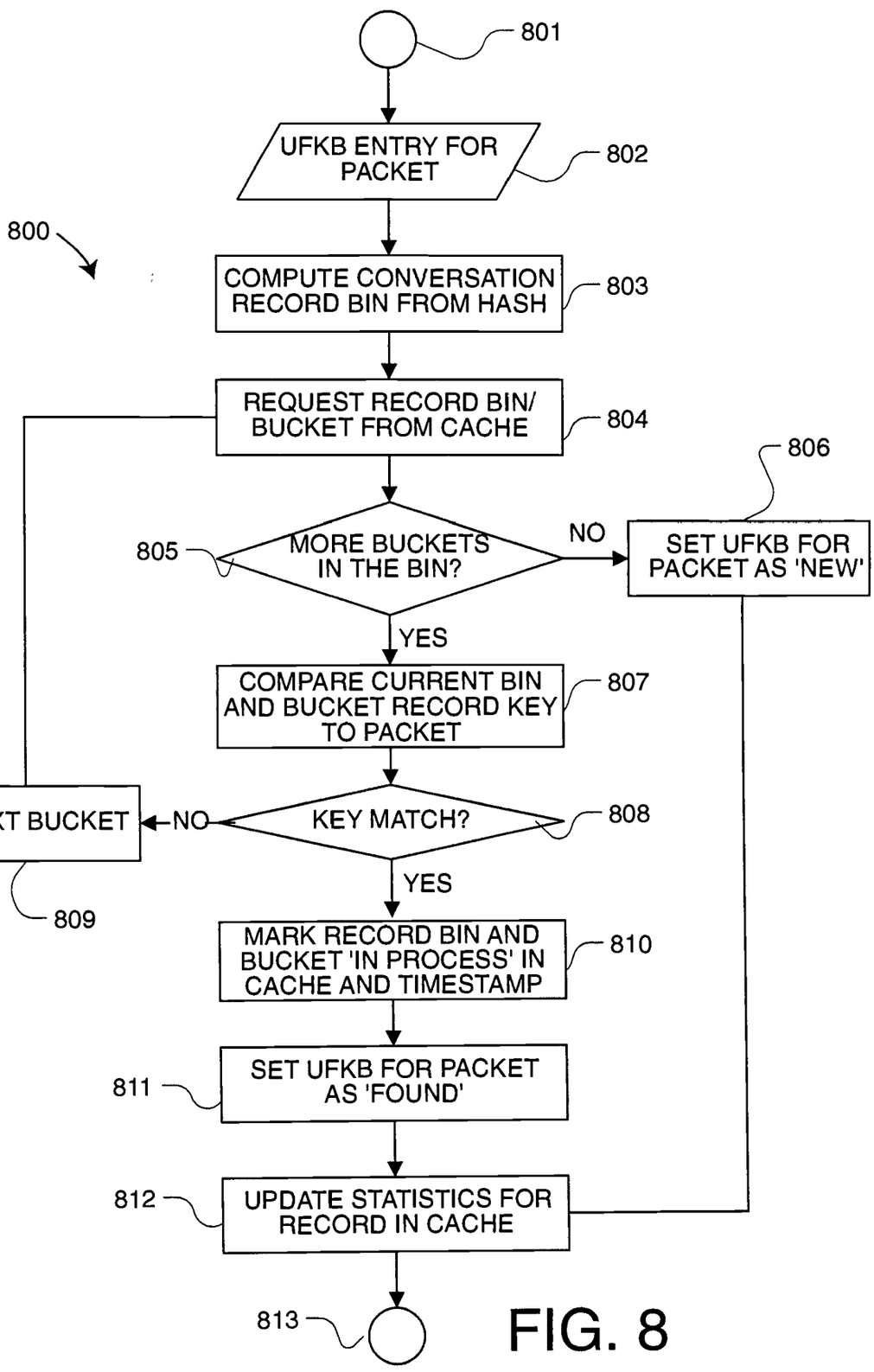

801
UFKB ENTRY FOR PACKET
802
800
COMPUTE CONVERSATION RECORD BIN FROM HASH
803
REQUEST RECORD BIN/ BUCKET FROM CACHE
804
806
805
MORE BUCKETS IN THE BIN?
NO
SET UFKB FOR PACKET AS 'NEW'
YES
COMPARE CURRENT BIN AND BUCKET RECORD KEY TO PACKET
807
NEXT BUCKET
NO
KEY MATCH?
808
YES
809
MARK RECORD BIN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

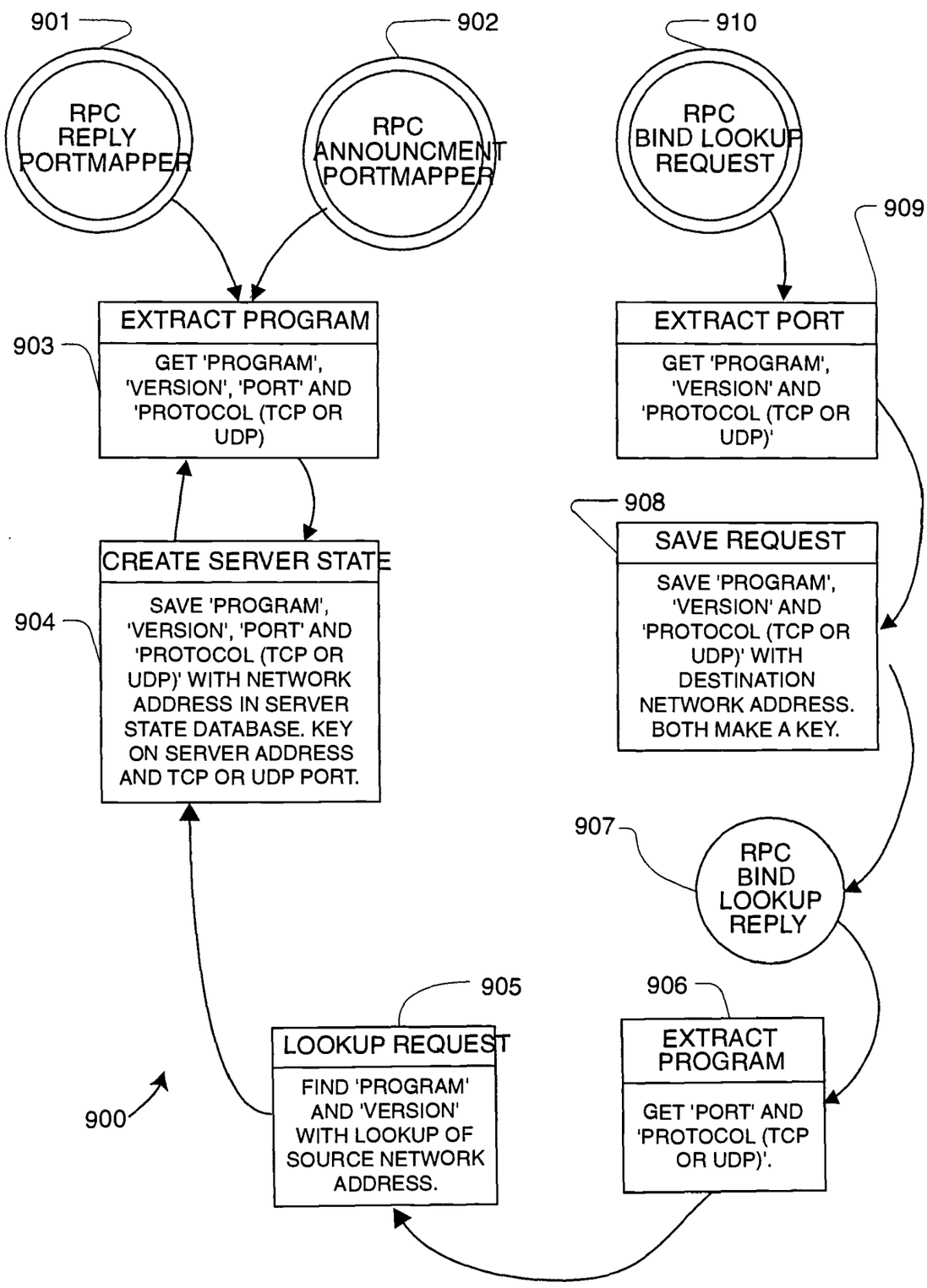

901
RPC REPLY PORTMAPPER
902
RPC ANNOUNCMENT PORTMAPPER
910
RPC BIND LOOKUP REQUEST
909
EXTRACT PROGRAM
GET 'PROGRAM', 'VERSION', 'PORT' AND 'PROTOCOL (TCP OR UDP)
903
EXTRACT PORT
GET 'PROGRAM', 'VERSION' AND 'PROTOCOL (TCP OR UDP)'
908
SAVE REQUEST
SAVE 'PROGRAM', 'VERSION' AND 'PROTOCOL (TCP OR UDP)' 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

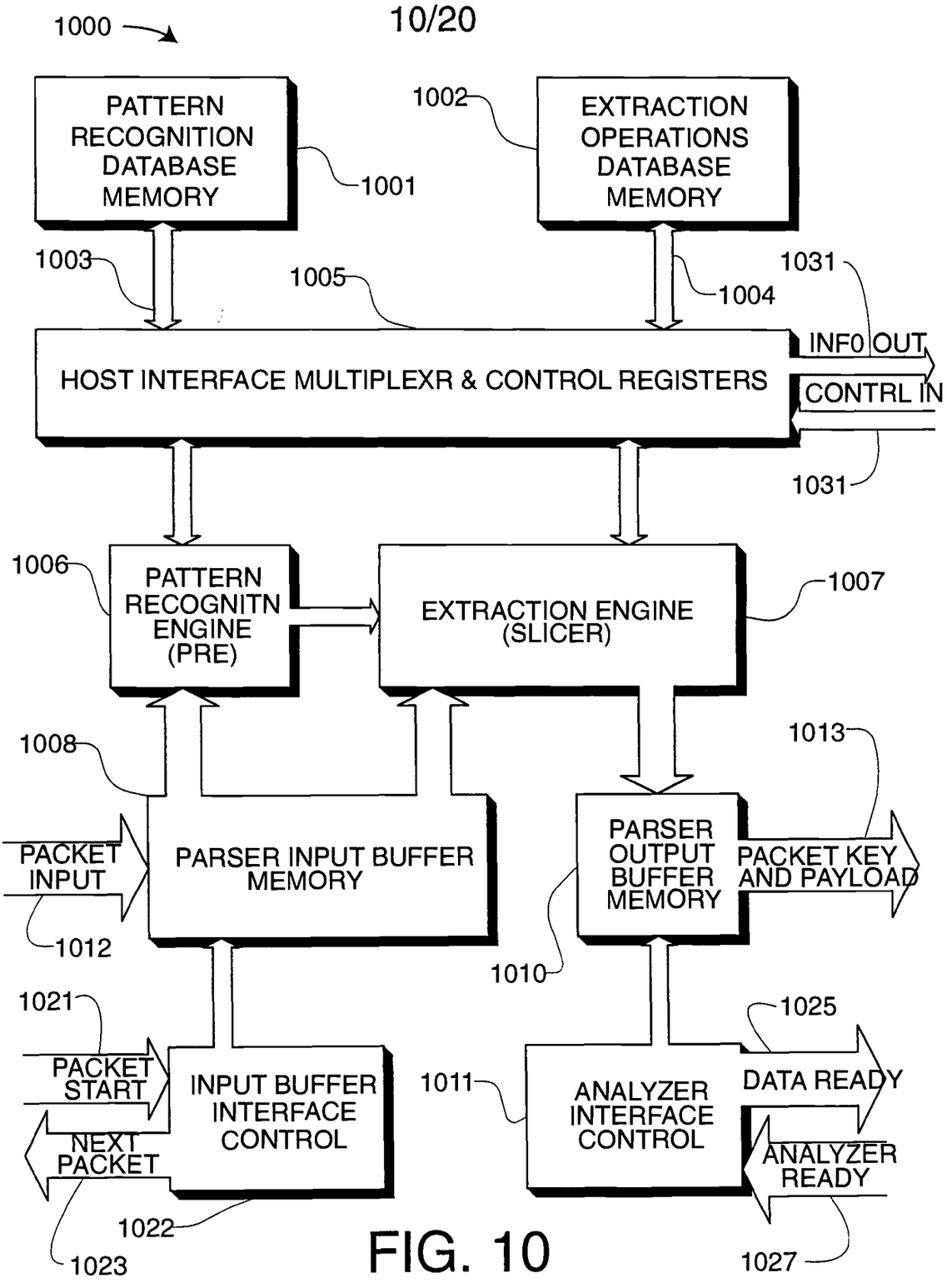

1000
PATTERN RECOGNITION DATABASE MEMORY
1001
1002
EXTRACTION OPERATIONS DATABASE MEMORY
1003
1004
1005
1031
INF0 OUT
CONTRL IN
1031
HOST INTERFACE MULTIPLEXR & CONTROL REGISTERS
1006
PATTERN RECOGNITN ENGINE (PRE)
EXTRACTION ENGINE (SLICER)
1007
1008
1013
PACKET INPUT
PARSER INPUT BUFFER MEMORY
PARSER 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

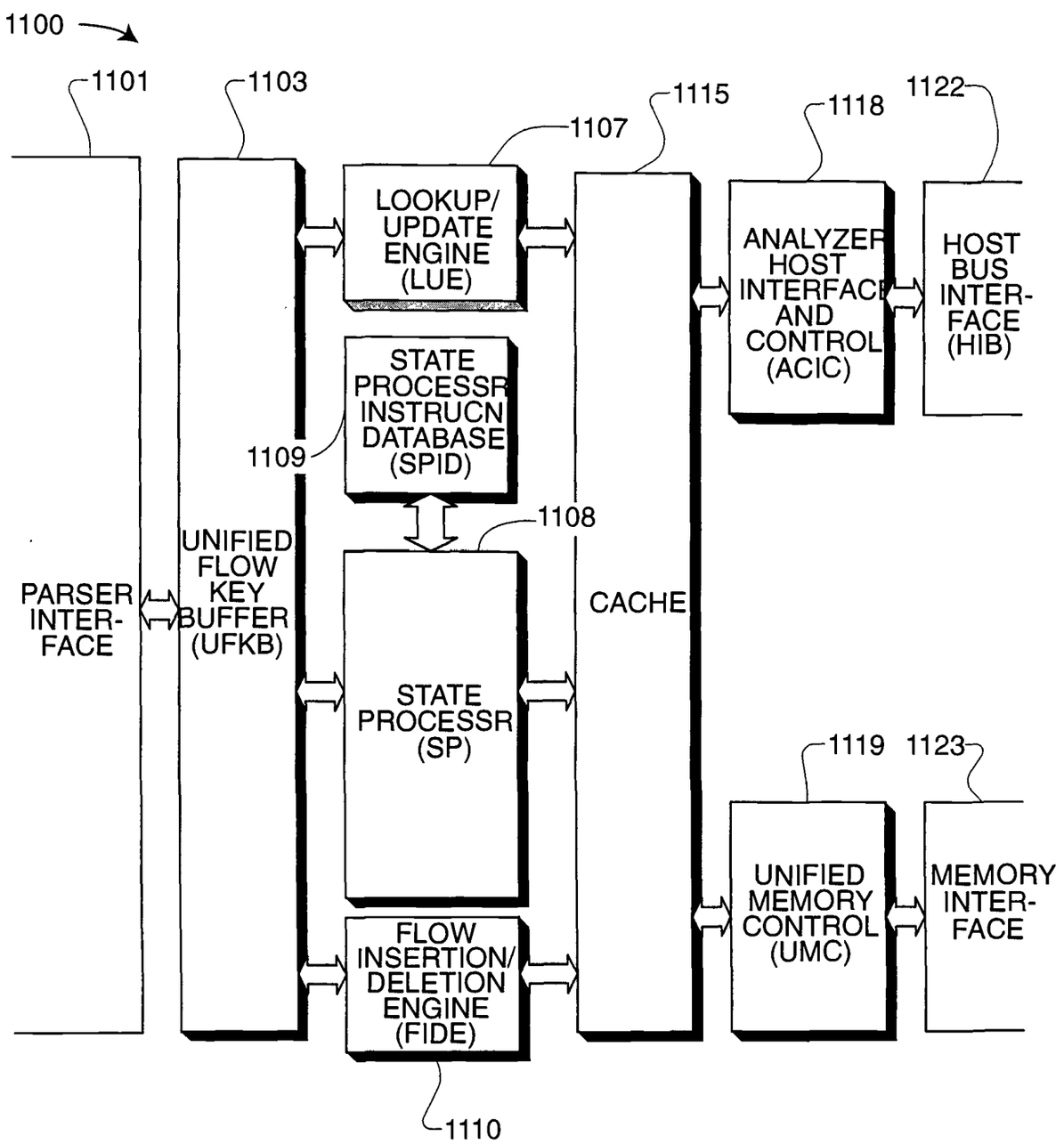

1100
1101
1103
1107
1115
1118
1122
LOOKUP/ UPDATE ENGINE (LUE)
ANALYZER HOST INTERFACE AND CONTROL (ACIC)
HOST BUS INTER-FACE (HIB)
STATE PROCESSR INSTRUCN DATABASE (SPID)
1109
1108
UNIFIED FLOW KEY BUFFER (UFKB)
PARSER INTER-FACE
CACHE
STATE PROCESSR (SP)
1119
1123
UNIFIED MEMORY CONTROL (UMC)
MEMORY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

FIG. 11

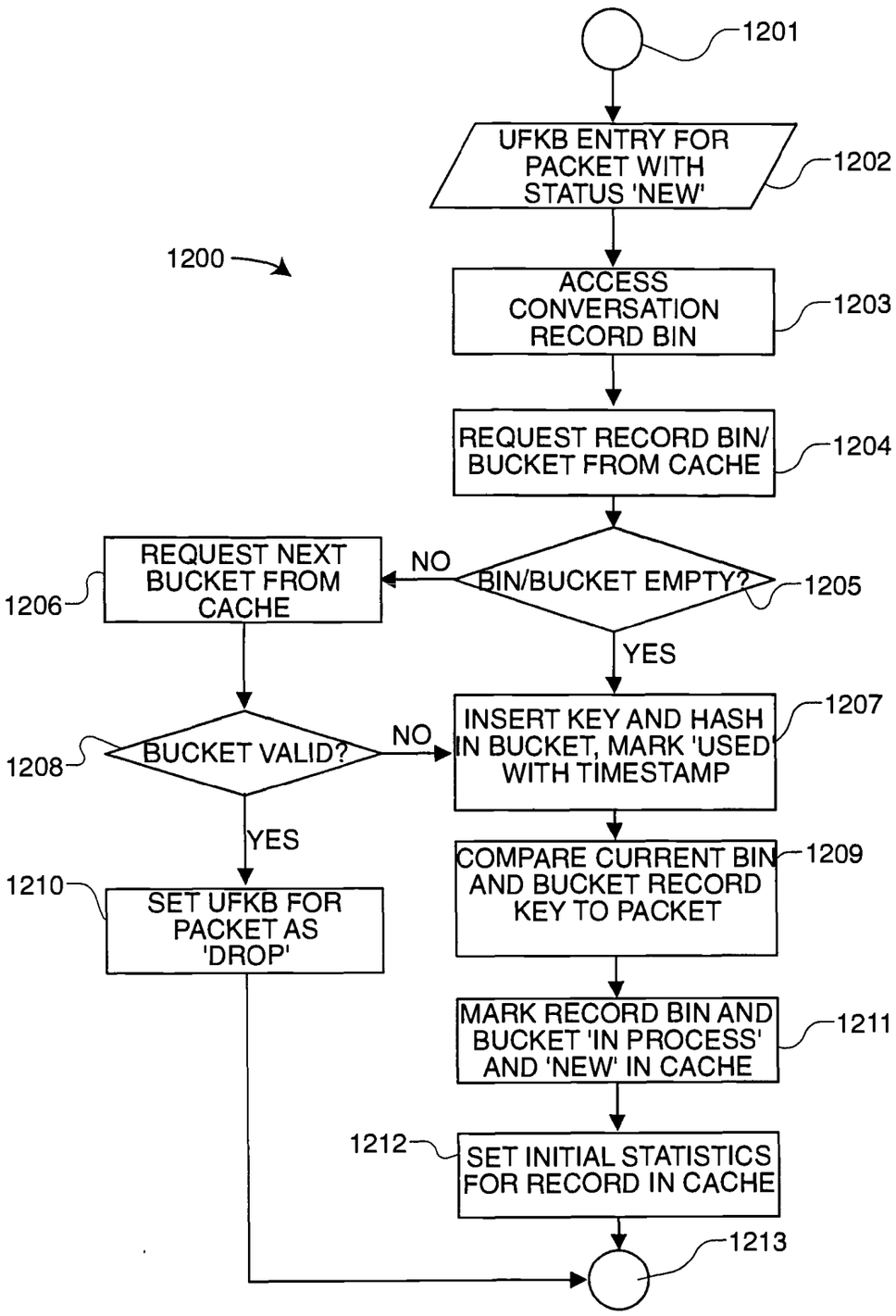

1201
UFKB ENTRY FOR PACKET WITH STATUS 'NEW'
1202
1200
ACCESS CONVERSATION RECORD BIN
1203
REQUEST RECORD BIN/ BUCKET FROM CACHE
1204
BIN/BUCKET EMPTY?
1205
NO
REQUEST NEXT BUCKET FROM CACHE
1206
YES
INSERT KEY AND HASH IN BUCKET, MARK 'USED' WITH TIMESTAMP
1207
BUCKET VALID?
1208
NO
YES
COMPARE CURRENT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

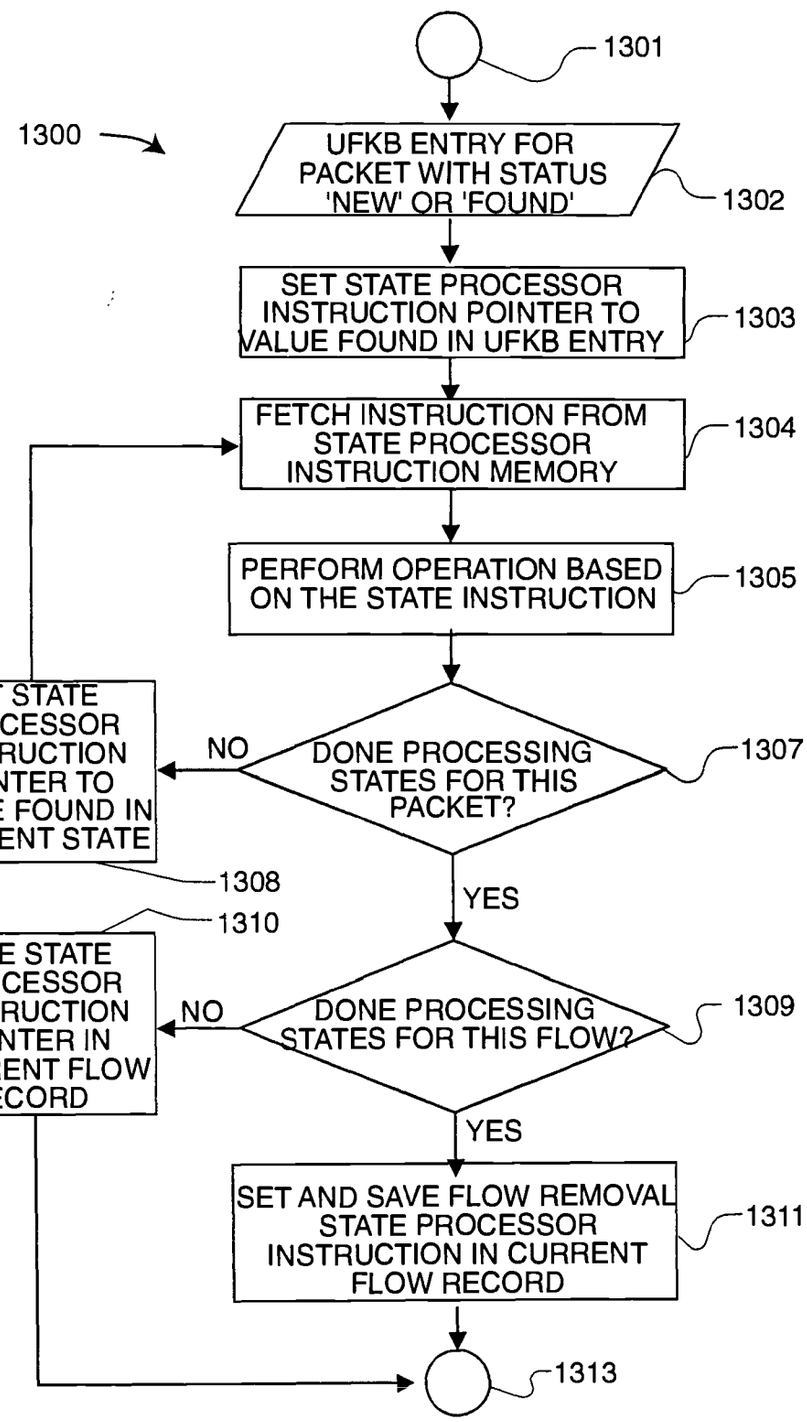

1301
1300
UFKB ENTRY FOR PACKET WITH STATUS 'NEW' OR 'FOUND'
1302
SET STATE PROCESSOR INSTRUCTION POINTER TO VALUE FOUND IN UFKB ENTRY
1303
FETCH INSTRUCTION FROM STATE PROCESSOR INSTRUCTION MEMORY
1304
PERFORM OPERATION BASED ON THE STATE INSTRUCTION
1305
DONE PROCESSING STATES FOR THIS PACKET?
1307
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

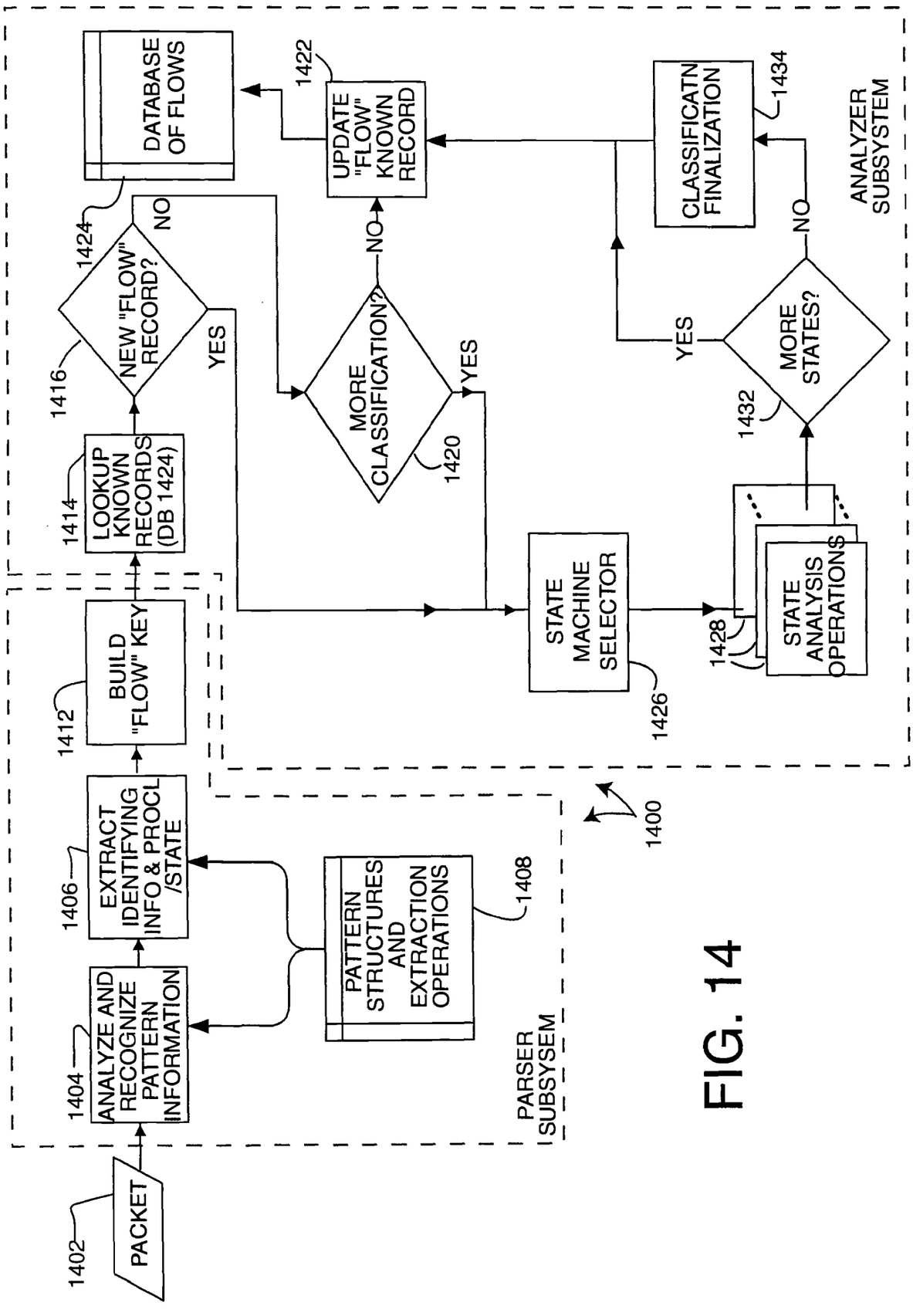

PACKET
1402
PARSER SUBSYSEM
ANALYZE AND RECOGNIZE PATTERN INFORMATION
1404
EXTRACT IDENTIFYING INFO & PROCL /STATE
1406
PATTERN STRUCTURES AND EXTRACTION OPERATIONS
1408
BUILD "FLOW" KEY
1412
LOOKUP KNOWN RECORDS (DB 1424)
1414
NEW "FLOW" RECORD?
1416
YES
NO
DATABASE OF FLOWS
1424
UPDATE "FLOW" KNOWN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

FIG. 14

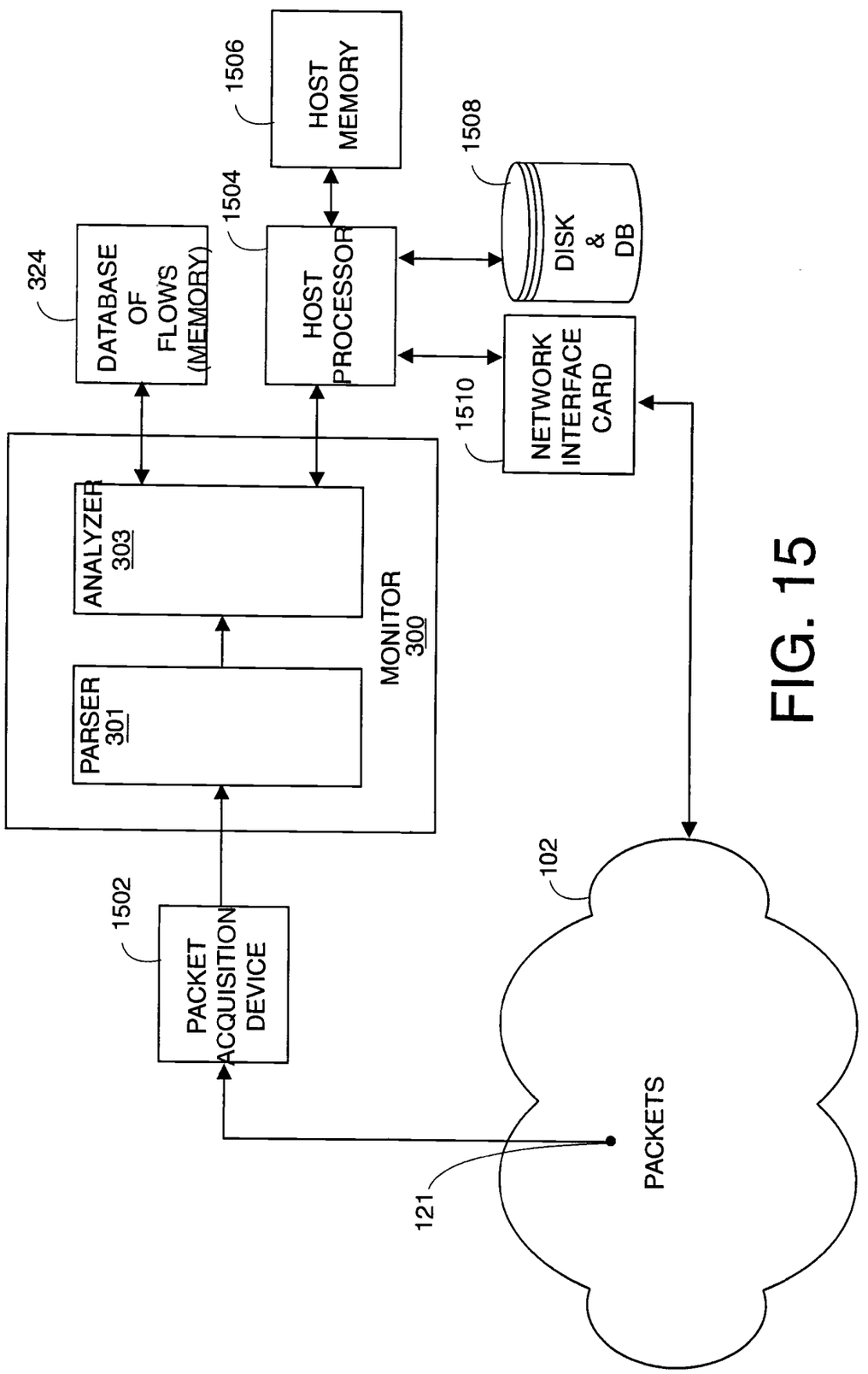

HOST MEMORY
1506
1508
DISK & DB
1504
HOST PROCESSOR
324
DATABASE OF FLOWS (MEMORY)
NETWORK INTERFACE CARD
1510
ANALYZER 303
PARSER 301
MONITOR 300
1502
PACKET ACQUISITION DEVICE
102
PACKETS
121

FIG. 15

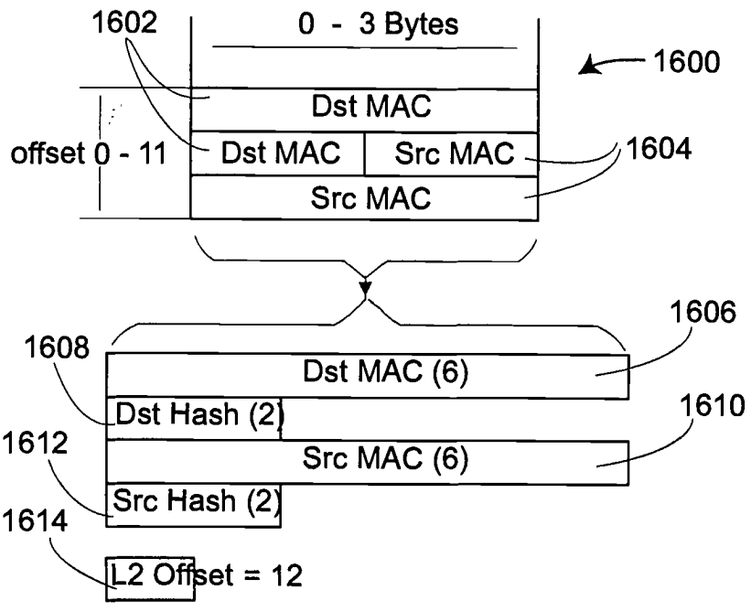

1602
0 - 3 Bytes
1600
Dst MAC
offset 0 - 11
Dst MAC
Src MAC
1604
Src MAC
1606
1608
Dst MAC (6)
Dst Hash (2)
1610
1612
Src MAC (6)
Src Hash (2)
1614
L2 Offset = 12

FIG. 16

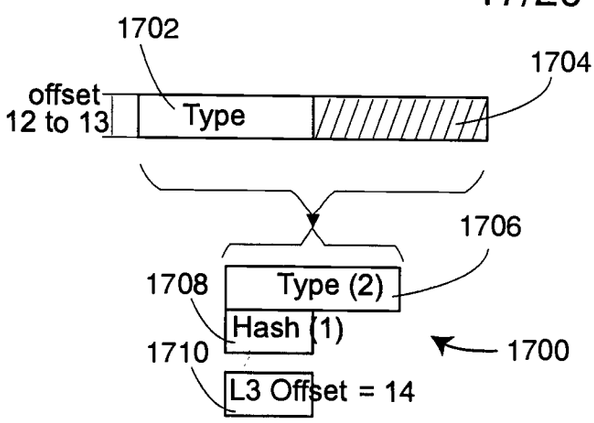


FIG. 17A

1712

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

Apple = 0x080007

* L3 Decoding
# L5 Decoding

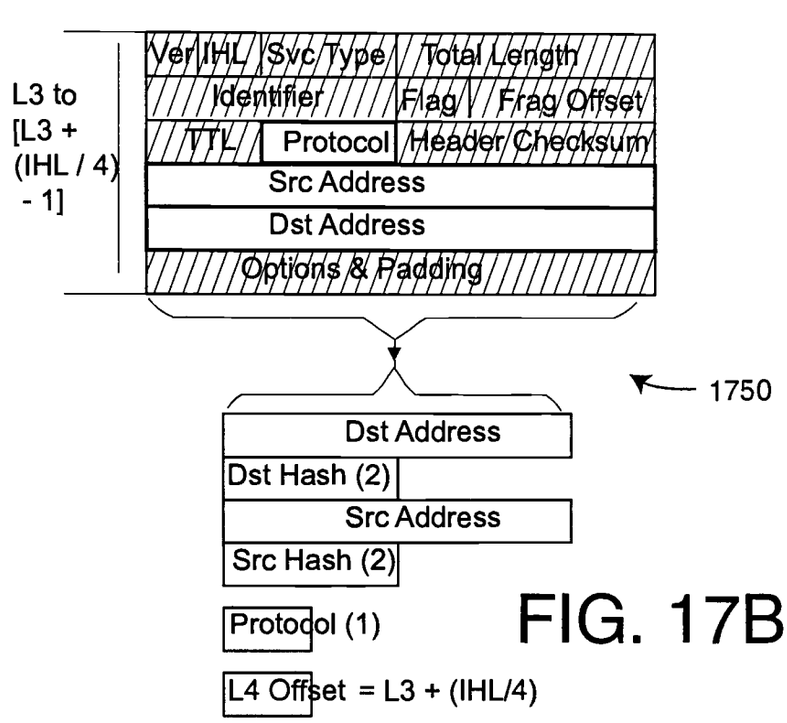


FIG. 17B

1752

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

* L4 Decoding
# L3 Re-Decoding

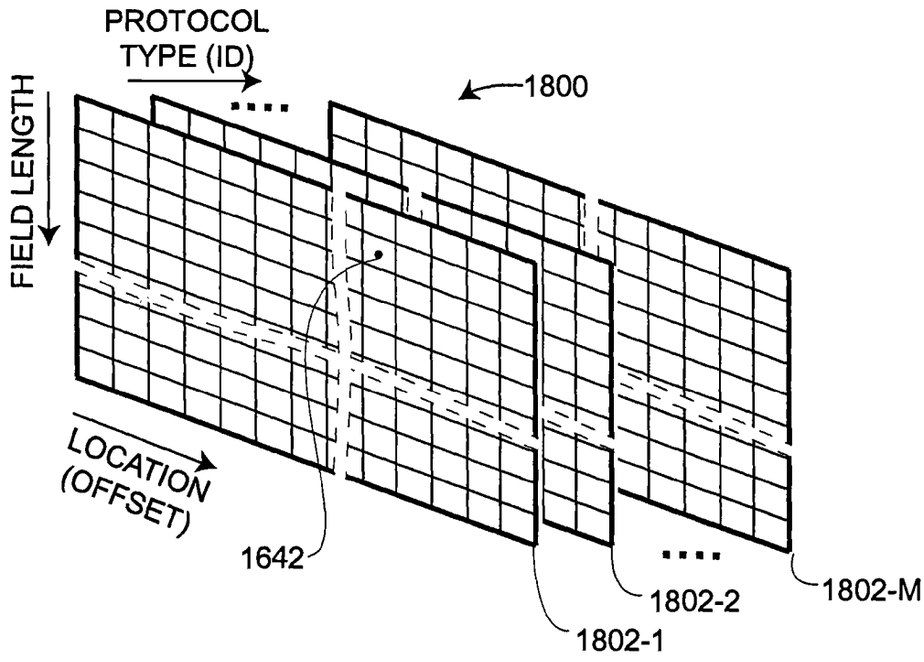


FIG. 18A

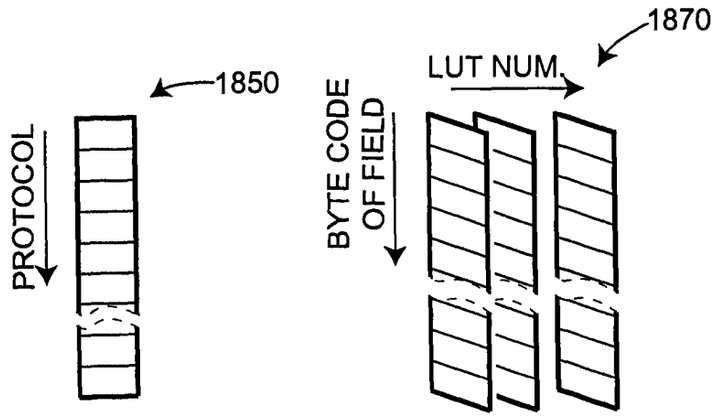


FIG. 18B

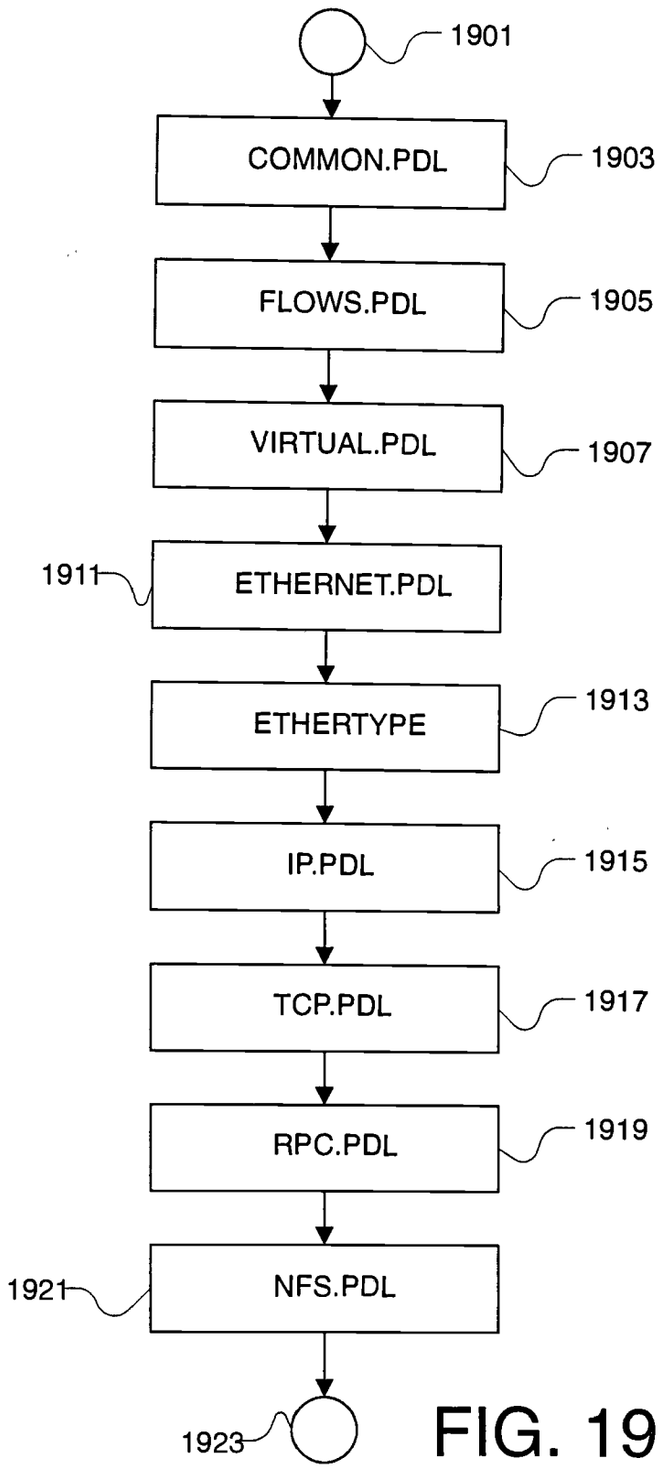

1901
COMMON.PDL
1903
FLOWS.PDL
1905
VIRTUAL.PDL
1907
1911
ETHERNET.PDL
ETHERTYPE
1913
IP.PDL
1915
TCP.PDL
1917
RPC.PDL
1919
1921
NFS.PDL
1923

FIG. 19

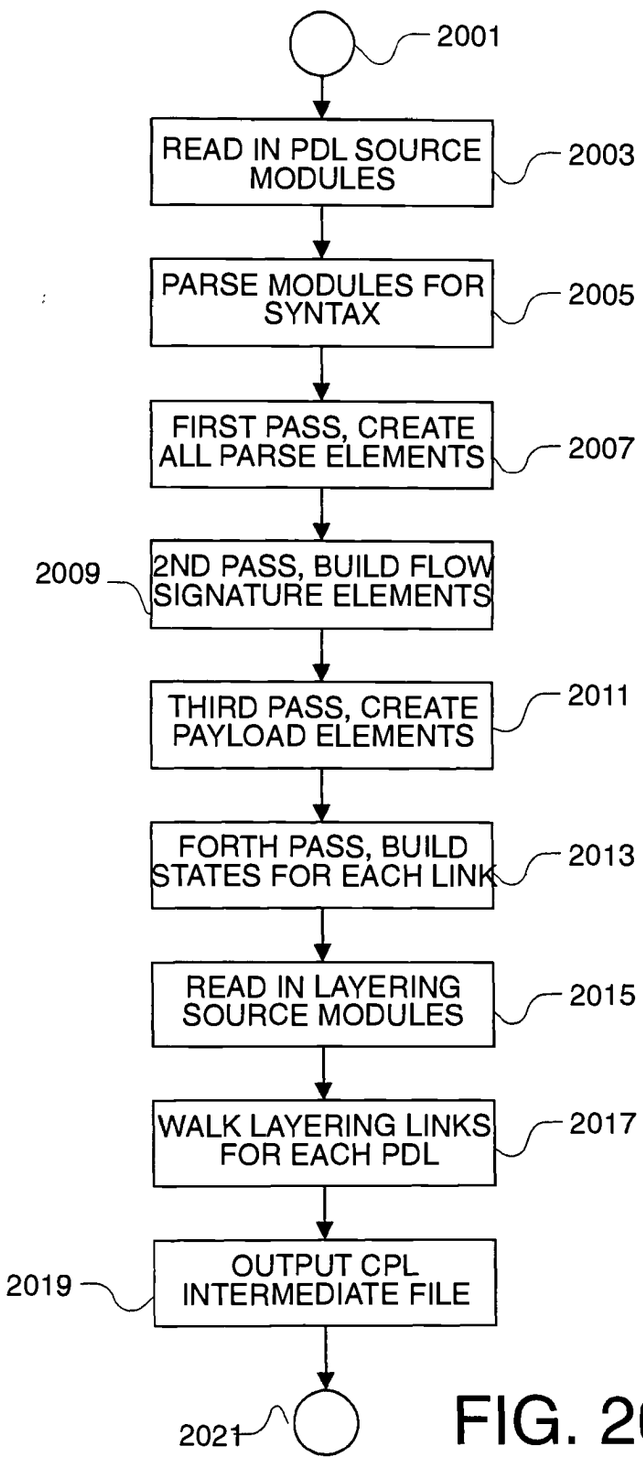

2001
READ IN PDL SOURCE MODULES
2003
PARSE MODULES FOR SYNTAX
2005
FIRST PASS, CREATE ALL PARSE ELEMENTS
2007
2009
2ND PASS, BUILD FLOW SIGNATURE ELEMENTS
THIRD PASS, CREATE PAYLOAD ELEMENTS
2011
FORTH PASS, BUILD STATES FOR EACH LINK
2013
READ IN LAYERING SOURCE MODULES
2015
WALK LAYERING LINKS FOR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

FIG. 20

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 3 - OFFICE COPY

OIPE
OCT 24 2000
PATENT & TRADEMARK OFFICE

Sector 4A

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. 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

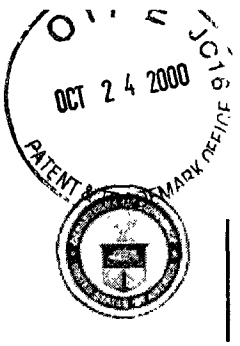
(A DUPLICATE OF THIS TRANSMITTAL IS ATTACHED):

Respectfully Submitted,

Oct 20, 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



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
40.00 OP
10/26/2000 SDENBOB1 00000068 500292 09609179
130.00 OP
690.00 OP
01 FC:581
02 FC:105
20.00 CH

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)

OIPE JC12 OCT 24 2000 PATENT & TRADEMARK OFFICE

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

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** 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**

OIPE
OCT 24 2000
PATENT & TRADEMARK OFFICE

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

_____ _____

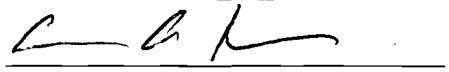
First Inventor's Signature **Date**

ADDITIONAL INVENTOR SIGNATURES:

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

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

Post Office Address: Same



Inventor's Signature

10/10/2000

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

OIPE JC16 OCT 24 2000 PATENT & TRADEMARK OFFICE

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 5607 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

James Torgerson 9/27/00

Inventor's Signature **Date**

OIPE
APR 12 2001
PATENT & TRADEMARK OFFICE

GP/2
215
#4
4-16-01

Our Docket ef. No.: APP' ..-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:

RECEIVED
APR 16 2001
Technology Center 2100

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

O I P E
APR 12 2001
PATENT & TRADEMARK OFFICE

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:

RECEIVED
APR 1 6 2001
Technology Center 2100

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

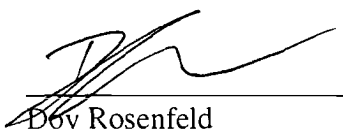
X Some of the references were cited in a search report from a foreign patent office in a counterpart foreign application. In particular, references AD, AF, AH, CI, EA, EB, EC, and ED were cited in a search report from a foreign patent office in a counterpart foreign application.

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

#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:

OIPE JC124 APR 11 2002 PATENT & TRADEMARK OFFICE

RECEIVED
APR 17 2002
Technology Center 2100

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 2002 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.

Respectfully submitted,

Date: 30 Mar 2002

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

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

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

Please find below and/or attached an Office communication concerning this application or proceeding.

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

APR 12 2001 PATENT & TRADEMARK OFFICE

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

OIPE JC59 APR 12 2001 PATENT & TRADEMARK OFFICE

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. #9	
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
WD	BA	5414650	May 9, 1995	Hekhuis	364	715.02	Mar. 24, 1993
WD	BB	5430709	Jul. 4, 1995	Galloway	370	13	Jun. 17, 1992
WD	BC	5432776	Jul. 11, 1995	Harper	370	17	Sep. 30, 1993
WD	BD	5493689	Feb. 20, 1996	Waclawsky et al.	~~395~~ 709	~~821~~ 206	Mar. 1, 1993
WD	BE	5500855	Mar. 19, 1996	Hershey et al.	370	17	Jan. 26, 1994
WD	BF	5568471	Oct. 22, 1996	Hershey et al.	370	17	Sep. 6, 1995
WD	BG	5574875	Nov. 12, 1996	Stansfield et al.	395	403	Mar. 12, 1993
WD	BH	5586266	Dec. 17, 1996	Hershey et al.	~~395~~ 709	~~200.11~~ 216	Oct. 15, 1993
WD	BI	5606668	Feb. 25, 1997	Shwed	~~395~~ 709	~~200.11~~ 216	Dec. 15, 1993
WD	BJ	5608662	Mar. 4, 1997	Large et al.	364	724.01	Jan. 12, 1995
WD	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	DATE CONSIDERED
Khanh Dinh	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.

OIPE JC59 APR 12 2001 PATENT & TRADEMARK OFFICE

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

RECEIVED APR 16 2001 Technology Center 2100

U.S. PATENT DOCUMENTS

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

PATENT & TRADEMARK OFFICE APR 12 2001

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

RECEIVED APR 16 2001 Technology Center 210

U.S. PATENT DOCUMENTS

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

OIPE
APR 12 2001
PATENT & TRADEMARK OFFICE 659

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: 6/30/2000	GROUP: ~~2756~~ 2155

RECEIVED
APR 16 2001
Technology Center 21

U.S. PATENT DOCUMENTS

*EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF BPPROPRIBTE
LW	EA	5917821	Jun. 29, 1999	Gobuyan et al.	370	392	Aug. 16, 1996
LW	EB	5414704	May 9, 1995	Spinney	370	60	Apr. 5, 1994
LW	EC	6014380	Jan 11, 2000	Hendel et al.	370	392	Jun. 30, 1997
LW	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

*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~~ 2855

INFORMATION DISCLOSURE STATEMENT

(Use several sheets if necessary)

OIPE JC124 APR 11 2002 PATENT & TRADEMARK OFFICE

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						

RECEIVED
APR 17 2002
Technology Center 2100

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.

Notice of References Cited	Application/Control No. 09/609,179	Applicant(s)/Patent Under 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.

#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.

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 Jun 03 Signed: _____

Name: Dov Rosenfeld, Reg. No. 38687

Received from < +1 510 291 2985 > at 6/13/03 7:41:12 PM [Eastern Daylight Time]

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

Received from < +1 510 291 2985 > at 8/13/03 7:41:12 PM [Eastern Daylight Time]

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)

11 ~~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.

A1

12 ~~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.

1 ~~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:

Received from < +1 510 291 2985 > at 6/13/03 7:41:12 PM [Eastern Daylight Time]

(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 ~~8.~~ (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.

Received from < +1 510 291 2985 > at 6/13/03 7:41:12 PM [Eastern Daylight Time]

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 ~~1~~ 12, 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.

15 10
~~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.

16 10
~~17~~. (Original) A method according to claim ~~15~~, 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:

Received from < +1 510 291 2985 > at 6/13/03 7:41:12 PM [Eastern Daylight Time]

(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

Received from < +1 510 291 2985 > at 6/13/03 7:41:12 PM [Eastern Daylight Time]

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

Received from < +1 510 291 2985 > at 6/13/03 7:41:12 PM [Eastern Daylight Time]

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

Received from < +1 510 291 2985 > at 6/13/03 7:41:12 PM [Eastern Daylight Time]

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

Received from < +1 510 291 2985 > at 6/13/03 7:41:12 PM [Eastern Daylight Time]

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

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

Received from < +1 510 291 2985 > at 6/13/03 7:41:12 PM [Eastern Daylight Time]

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

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,

Received from < +1 510 291 2985 > at 6/27/03 12:06:26 PM [Eastern Daylight Time]

(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

Pg

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)
3☐ Notice of Draftperson's Patent Drawing Review (PTO-948)
5☐ Information Disclosure Statements (PTO-1449), Paper No. ____.
7☐ Examiner's Comment Regarding Requirement for Deposit of Biological Material
2☐ Notice of Informal Patent Application (PTO-152)
4☐ Interview Summary (PTO-413), Paper No.____ .
6☐ Examiner's Amendment/Comment
8☐ Examiner's Statement of Reasons for Allowance
9☐ Other .

HOSAIN T. ALAM
PRIMARY EXAMINER

RY

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

NOTICE OF ALLOWANCE AND FEE(S) DUE

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.

PTOL-85 (REV. 05-03) Approved for use through 04/30/2004.

Petitioners' EX1039 Page 233

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.

TRANSMIT THIS FORM WITH FEE(S)

Petitioners' EX1039 Page 234

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
PO Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/609,179	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

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.

PTOL-85 (REV. 05-03) Approved for use through 04/30/2004.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address COMMISSIONER FOR PATENTS
PO Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/609,179	06/30/2000	Russell S. Dietz	APPT-001-2	2668

7590 07/01/2003

EXAMINER
DINH, KHANH Q

ART UNIT	PAPER NUMBER
2155	

Dov Rosenfeld
5507 College Avenue
Suite 2
Oakland, CA 94618
UNITED STATES

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.

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 (NYE.) LDD 10-22-03 entered LDD 10-27-03

OK to enter K.D. NP 10/23/03

Match and Return

INTRODUCTORY REMARKS:

Kindly amend this application as follows and kindly consider the following remarks.

Received from < +1 510 281 2985 > at 7/14/03 2:15:18 PM [Eastern Daylight Time]

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--

Received from < +1 510 291 2985 > at 7/14/03 2:15:18 PM [Eastern Daylight Time]

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

Received from <+1 510 291 2985 > at 7/14/03 2:15:18 PM [Eastern Daylight Time]

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 boing facsimile transmitted to the United States Patent and Trademark Office at telephone number 703-746-7238 ddressed the Commissioner for Parents, P.O. Box 1450, Alexandria, VA 22313-1450 on

Date: 7/14/03

Signed: _____

Name: Dov Rosenfeld, Reg. No. 38687

Received from < +1 510 281 2985 > at 7/14/03 2:15:18 PM [Eastern Daylight Time]

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. 38687
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. 38687		
Signature		Date	July 14, 2003

Received from < +1 510 291 2985 > at 7/14/03 2:15:18 PM [Eastern Daylight Time]

Sep 24 03 05:47p Dov senfeld +1-510- -2985 p.4

JC16 SEP 2 4 2003 PATENT & TRADEMARK OFFICE

CW

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] (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 50-0292 (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) [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 TTRAN2 00000067 09609179
01 FC:1501 1300.00 OP
02 FC:8001 30.00 OP

TRANSMIT THIS FORM WITH FEE(S)

PTOL-85 (REV. 05-03) Approved for use through 04/30/2004. OMB 0651-0033 U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Received from < +1 510 291 2985 > at 9/24/03 8:48:06 PM [Eastern Daylight Time]

INVENTEK

Dov Rosenfeld
5507 College Avenue, Suite 2
Oakland, CA 94618, USA
Phone: (510)547-3378; Fax: (510)653-7992
dov@inventek.com

Fax

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

Received from <+1 510 291 2985> at 9/24/03 8:48:08 PM [Eastern Daylight Time]

Match and Return

JC16 SEP 24 2003 PATENT & TRADEMARK OFFICE

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

Received from <+1 510 291 2985> at 9/24/03 5:48:05 PM [Eastern Daylight Time]

1999

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

OIPE
APR 0 8 2004
PATENT & TRADEMARK OFFICE

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

1 of 2
#12

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

APR 12 2004

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
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO : 6,665,725 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

Application or Docket Number

PATENT APPLICATION FEE DETERMINATION RECORD
Effective October 1, 2000

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

4u0

AMENDMENT A	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	* 18	Minus	** 20	=
Independent	* 3	Minus	*** 3	=
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=	
X40=		OR	X80=	
+135=		OR	+270=	
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=	
X40=		OR	X80=	
+135=		OR	+270=	
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=	
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.

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		

* If the difference in column 1 is less than zero, enter "0" in column 2

SMALL ENTITY TYPE ☐ RATE	FEE	OR	OTHER THAN SMALL ENTITY RATE	FEE
	345.00	OR		690.00
X$ 9=		OR	X$18=	
X39=		OR	X78=	
+130=		OR	+260=	
TOTAL		OR	TOTAL	

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 RATE	ADDITIONAL FEE	OR	OTHER THAN SMALL ENTITY RATE	ADDITIONAL 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	ADDITIONAL FEE	OR	RATE	ADDITIONAL 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	ADDITIONAL FEE	OR	RATE	ADDITIONAL 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.

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: 9/609179

Total Fee Calculation

	Fee Code Sm./Lg	Total # Claims		Number Extra		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. Autes

Office of Initial Patent Examination

Figure 7

FORM OIPE-RAM-01 (Rev. 12/97)

ISSUE SLIP STAPLE AREA (for additional cross references)

POSITION	INITIALS	ID NO.	DATE
FEE DETERMINATION	JL		7/12/02
O.I.P.E. CLASSIFIER		48	7/17/02
FORMALITY REVIEW			
RESPONSE FORMALITY REVIEW		64674	8-22

INDEX OF CLAIMS

✔ ..... Rejected
= ..... Allowed
— (Through numeral)... Canceled
÷ ..... Restricted
N ..... Non-elected
I ..... Interference
A ..... Appeal
O ..... Objected

Claim Final	Claim Original	Date
	1	✔ STAPLED 6/17/03
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

SEARCHED

Class	Sub.	Date	Exmr.
709	203	5/28/03	WD
	206		
	216		
	217		
	222		
	246		
	225		
	228		
	239		
	232		
703	26		
370	489		
	13		
	17		
updated	class search	6/18/03	WD
updated	search	6/19/03	WD

INTERFERENCE SEARCHED

Class	Sub.	Date	Exmr.
709	230	6/18/03	WD
	246		
	228		
370	489		

SEARCH NOTES
(INCLUDING SEARCH STRATEGY)

	Date	Exmr.
WEST	5/29/03	WD

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

☐ 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

☑ 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

☑ 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

☑ 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 | KWIC
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 | KWIC
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 | KWIC
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 | KWIC
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

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

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
Examiner Number: **74865 / DINH, KHANH**
Filing Date: **06/30/2000**
Group Art Unit: **2155**
Effective Date: **06/30/2000**
Class/Subclass: **709/236.000**
Application Received: **07/03/2000**
Lost Case: **NO**
Patent Number:
Interference Number:
Waiting for Response Desc.
Mail Non Final
Issue Date: **00/00/0000**
Unmatched Petition: **NO**
Date of Abandonment: **00/00/0000**
L&R Code: Secrecy Code:**1**
Attorney Docket Number: **APPT-001-2**
Third Level Review: **NO**
Secrecy Order: **NO**
Status: **41 /NON FINAL ACTION MAILED**
Status Date: **06/04/2003**
Confirmation Number: **2668**
Oral Hearing: **NO**
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

Petitioners' EX1039 Page 267

GTrace – A Graphical Traceroute Tool

Ram Periakaruppan, Evi Nemeth
University of Colorado at Boulder
Cooperative Association for Internet Data Analysis (CAIDA)
{ramanath, evi}@cs.colorado.edu

Abstract

Traceroute [Jacobson88], originally written by Van Jacobson in 1988, has become a classic tool for determining the routes that packets take from a source host to a destination host. It does not provide any information regarding the physical location of each node along the route, which makes it difficult to effectively identify geographically circuitous unicast routing. Indeed, there are examples of paths between hosts just a few miles apart that cross the entire United States and back, phenomena not immediately evident from the textual output of *traceroute*. While such path information may not be of much interest to many end users, it can provide valuable insight to system administrators, network engineers, operators and analysts. We present a tool that depicts geographically the IP path information that *traceroute* provides, drawing the nodes on a world map according to their latitude/longitude coordinates.

1. Introduction

Today's Internet has evolved into a large and complex aggregation of network hardware scattered across the globe, with resources accessed transparently with respect to their location, be it in the next room or on another continent. As the Internet becomes increasingly commercialized among many different corporate administrative entities, it is more difficult to ascertain the geographical routes that packets actually travel across the network. Knowledge of these geographical paths can provide useful insight to system administrators, network engineers, operators and analysts.

It is challenging to obtain the location for a given node of a path since there is no existing database that accurately maps hostnames or IP addresses to physical locations. Although RFC 1876 [RFC1876] defined a DNS resource record to carry such location information (the LOC record) for hosts, networks and subnets, very few sites maintain LOC records. Hence there is no straightforward way to determine the physical location of hosts.

GTrace is a graphical front end to *traceroute* that uses a number of heuristics to determine the location of a node. Often the name of a node in the path contains geographical information such as a city name/abbreviation or airport code. GTrace operates on the assumption that these codes and names indicate the physical location of the node. The locations obtained are connected together on a world map to show the geographical path that packets take from the source to destination host. GTrace also tries to verify the validity of each location obtained, eliminating ones that are incorrect.

The following sections review the *traceroute* tool and describe the design and implementation of GTrace. We also show example output from GTrace.

2. Traceroute

Traceroute is a tool that discovers the route an IP datagram takes through the Internet from a source host to a destination host. It works by exploiting the TTL (Time To Live) field of the IP Header. Each router that handles an IP datagram decrements the TTL field. When the TTL reaches zero, a router must discard the packet and send an error message to the originator of the datagram.

Traceroute uses this feature, initially sending a datagram with the TTL set to one. The first router along the path, upon receiving the datagram decrements the TTL, discards the datagram and sends back an ICMP error

message. *Traceroute* records this first IP address (source address of the error message packet) and then sends the next datagram with the TTL set to two. This process continues until the datagram finally reaches the target host, or until the maximum TTL threshold is reached.

3. Design and Implementation of GTrace

Recognizing that it is not possible to obtain precise physical location information for all existing IP addresses, our main design criteria for GTrace was that it be sufficiently flexible to support the addition of new databases and heuristics. We chose to implement GTrace in Java, for both its portability and its new Swing [Swing] user interface toolkit. GTrace operates in two phases. In the first phase GTrace executes *traceroute* to the destination host and tries to determine locations for each node along the path. During the second phase, GTrace verifies whether the locations obtained in the previous phase are reasonably correct.

GTrace is composed of the following seven key components: Graphical User Interface, Dispatcher Thread, Hop Threads, Lookup Client, NetGeo Server, Lookup Server and Location Verifier. Fig. 1 illustrates the overall architecture of the tool. The function of each component is described below.

3.1 Graphical User Interface

The Main Thread handles all features of the Graphical User Interface and is responsible for spawning the dispatcher thread when a destination host is specified. Fig. 2 shows a snapshot of GTrace on startup. The GUI has two sections, with a map on the top and traditional

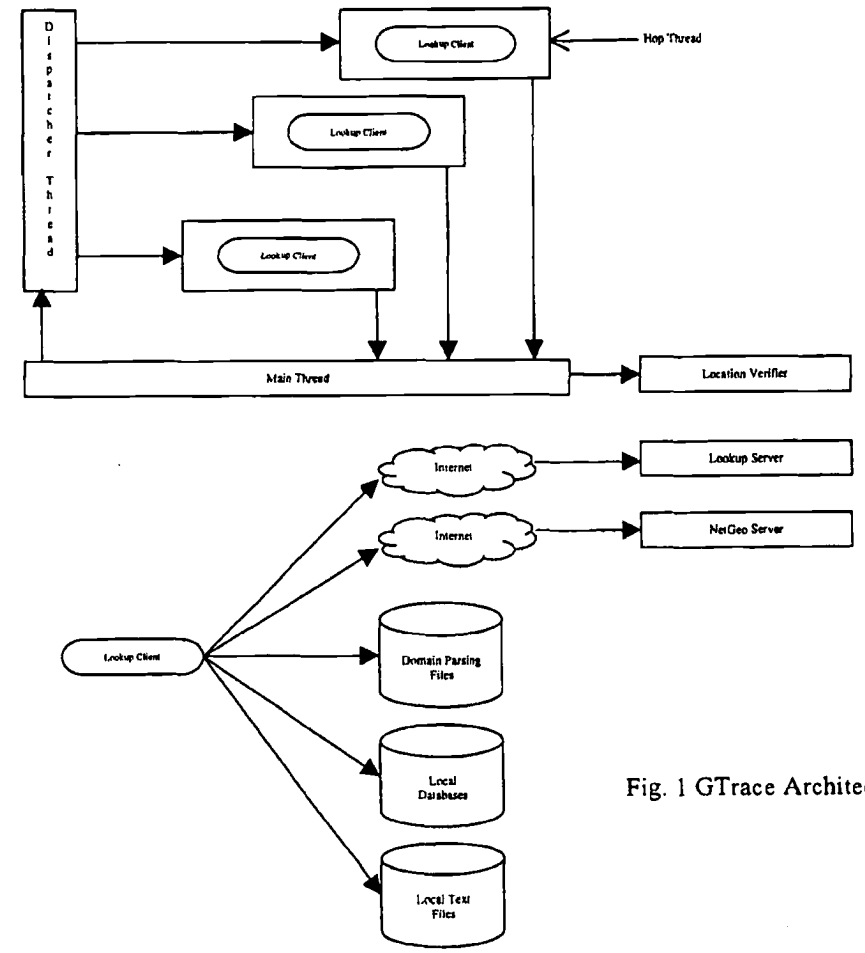


Fig. 1 GTrace Architecture

traceroute output below. The tool supports zooming in or out of particular regions of the maps. Twenty-three maps are available courtesy of VisualRoute [VisualRoute] and users can also add their own. We later provide an example that highlights some of the features of the GUI.

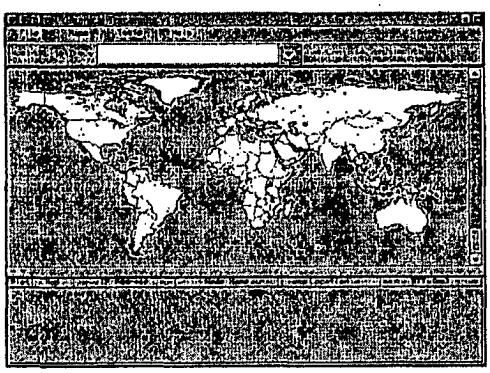


Fig. 2 GTrace's startup screen

3.2 Dispatcher Thread

The function of the dispatcher thread is to execute *traceroute* to the destination host. It then reads the output of *traceroute*, creating a new thread for each line of output. These threads are referred to as *hop threads*. The dispatcher thread can also read *traceroute* output from a file, which allows users to visualize traceroutes performed using third-party *traceroute* servers.

3.3 Hop Threads

Each hop thread parses its line of *traceroute* output and immediately notifies the main thread so that it can update the display with relevant *traceroute* fields for the corresponding hop. It then creates an instance of the Lookup Client, which tries to determine the location of the node and return the resulting information to the main thread before exiting.

3.4 Lookup Client

The Lookup Client tries to determine the location of a node by using a set of search heuristics. Many of the nodes in a typical *traceroute* path are in the ".net" domain. Often the names of these nodes have some geographical hint in them. The Lookup Client uses customized domain parsing files that specify rules for extracting these geographic hints. We have such files for several ".net" domains that use internally consistent naming conventions within their domain.

However this technique does not solve the problem of locating nodes that do not have embedded geographical hints. GTrace also utilizes databases from CAIDA [DBCAIDA] and NDG Software [DBNDG] that map hostnames and IP addresses to latitude/longitude coordinates. For nodes with no information in these databases, the Lookup Client uses the domain's registered address (unfortunately often only the headquarters for a geographically distributed infrastructure) obtained through a *whois* lookup to determine the location. Nodes for which the Lookup Client is unable to determine a location are listed in the text portion, but skipped in the geographical display.

The search algorithm is described below. We try each heuristic in turn, stopping as soon as one yields a location. The Lookup Client also makes a note of the search step that produced the location, providing this information to the user as well as the Location Verifier.

Search Algorithm:

1. Check the cache to see if the location for the IP address has already been determined from a previous trace.

2. Check if the host has a DNS LOC record. If not, reduce the hostname to the next higher level domain (i.e., remove the first component of the name) and check again for a LOC record. Continue until we have reached the last meaningful component of the name (for example foo.com in xxx.foo.com or bar.com.au in xxx.yyy.bar.com.au). Note that if a site has a LOC record for the whole domain, but machines are located outside the scope of that LOC record, GTrace would end up using incorrect data. If the Location Verifier detects such a situation, GTrace will notify the user and optionally can be configured to notify GTrace's author, who will contact the DNS administrator at the corresponding site to correct their LOC records.

3. Search for a complete match of the hostname/IP address in the databases and files specified in the GTrace configuration file.

4. If the hostname has a corresponding domain parsing file, use the rules defined in the file to extract geographical hints and proceed as indicated in the file.

5. Reduce the hostname to the next higher level domain as in step 2 and search for a match as in step 3. The process is repeated until we have reached the last meaningful component of the name.

6. Query the NetGeo [NetGeo] server with the IP address. NetGeo determines the location based on *whois* registrant information.

7. If still no match occurs and the last two letters of the hostname end in a two-letter country code, map it to the geographic center of that country.

The search algorithm is ordered in decreasing level of location reliability. Locations obtained from steps 2 and 3 are taken as authoritative, while those from step 4 onward are considered a guess. Cache entries will indicate whether the location was authoritatively determined or was a guess; this status determines the color of the lines connecting the nodes on the map.

The Lookup Client does not determine locations for IP addresses that fall in the ranges 10.0.0.0 - 10.255.255.255, 172.16.0.0 - 172.31.255.255 or 192.168.0.0 - 192.168.255.255, as these blocks are reserved for private internet use [RFC 1918]. Unfortunately some addresses in these blocks do occur in traces since some ISPs use this address space for internal router interfaces. These nodes are shown in the text portion of the display with the location marked as private internet use.

The Lookup Client queries the Lookup Server if one is defined in the GTrace configuration file and if location information has not been obtained through step 1, 2 or 3 of the search algorithm. GTrace compares the reply from the Lookup Server with any obtained previously from local lookups, with preference given to the location obtained through a lower numbered search step. Based on the GTrace configuration file, the Lookup Client also uses databases, text files and domain parsing files as follows.

Databases

The Lookup Client may need to perform lookups in many databases before determining a location. GTrace's database support is provided by the BerkeleyDB [BerkeleyDB] embedded database system, which supports a Java API that the Lookup Client uses to query the databases. The database interface allows multiple thread reads on the same database at the same time. Locking is not an issue, since Lookup Clients only read, do not write.

The following five databases are packaged with the GTrace distribution.

Machine.db [DBCAIDA]	Maps machine names to their latitude/longitude values.
Organization.db [DBCAIDA]	Maps organizations to their latitude/longitude values.
Hosts.db [DBNDG]	Maps IP addresses to their latitude/longitude values.
Cities.db [DBCAIDA]	Maps cities around the world to their latitude /longitude values.
Airport.db [AirportCodes]	Maps airport codes to their latitude/longitude values.

One can add a new database in BerkeleyDB format to GTrace with *GTraceCreateDB* and by adding an entry to the GTrace configuration file. The contents of the database ie., whether it maps hostnames, IP addresses, or both to latitude/longitude values, also have to be indicated in the configuration file. The user can also add records to existing databases using *GTraceAddRec*. *GTraceCreateDB* and *GTraceAddRec* are Java classes packaged with the GTrace distribution.

Text Files

Users may also specify new locations for nodes in text files, though it is more efficient to create a database for large data sets. New files have to be listed in the GTrace configuration file

in order for the search algorithm to have access to them.

Domain Parsing files

Files describing properties of each domain are used to ferret out geographical hints embedded in hostnames. These files define parsing rules using Perl5 compatible regular expressions. GTrace uses the regular expression library from ORO Inc. [OROMatcher] for parsing. New files can be added and existing ones modified without requiring any changes to GTrace.

For example, ALTER.NET (a domain name used by UUNET, a part of MCI/WorldCom) names some of their router interfaces with three letter airport codes as shown below:

193.ATM8-0-0.GW2.EWR1.ALTER.NET
(EWR -> Newark, NJ)

190.ATM8-0-0.GW3.BOS1.ALTER.NET
(BOS -> Boston, MA)

198.ATM6-0.XR2.SCL1.ALTER.NET
(Exception)

199.ATM6-0.XR1.ATL1.ALTER.NET
(ATL -> Atlanta, GA)

Fig. 3 shows an example of a GTrace domain parsing file that would work for ALTER.NET hosts. The file first defines the regular expressions, followed by any domain specific exceptions. The exceptions are strings that match the result of the regular expressions. The user may identify the exception's location either by city or by latitude/longitude value using the format shown below:

```
exception=city,state,country
        city,country
          L: latitude, longitude
```

In the former case, the user should also use *GTraceQueryDB* to ensure that the cities database has a latitude/longitude entry for the city specified. The first line in Fig. 3 defines a substitution operation, which when matched against 193.ATM8-0-0.GW2.EWR1.ALTER.NET, would return "EWR". The contents following the last " / " of the first line indicate what to do with a successful match, namely in this case to instruct the program to first check for a match in the data specified in the current file and then for a match in the airport database.

```
s/.*?\.([^\.]+)\d\.ALTER\.NET/$1/this,airport.db
scl=santaclara, ca, us
tco=tysonscorner, va, us
nol=neworleans, la, us
```

Fig. 3 Example of a domain parsing file for ALTER.NET.

The reason for checking the domain parsing file first is that sometimes the naming scheme for a given domain is not consistent. For example, a search for SCL obtained from 198.ATM6-0.XR2.SCL1.ALTER.NET in the airport database would return a location for Santiago de Chile. In the case of ALTER.NET, they also use three letter codes that are not airport codes but abbreviations for US cities (Fig. 3 illustrates three such abbreviations.) Note that if this exception list were not present and SCL did get mapped to Chile, the Location Verifier would likely have eliminated it using the Round Trip Time (RTT) heuristic described later, which would have recognized the RTT as much too small to get a packet to Chile and back.

Sometimes ISPs name their hosts with more than one geographical hint in them. For example VERIO.NET names some of their hosts in the following format: den0.sjc0.verio.net, which typically suggests source and destination of the interface. If there is no rule on whether the convention is to use the source or destination label first in the hostname, the rule could be defined to extract both and GTrace could use the Location Verifier's heuristics to guess.

The advantage of this technique is that one can describe an entire domain as a set of rules without needing database entries for every host in the domain. The limitation of the technique is that it will fail for domains that do not use internally consistent naming schemes.

3.5 NetGeo Server

The original design of the Lookup Client performed and parsed results of *whois* lookups directly, which required storage of a prohibitively large number of mappings of world

locations to latitude/longitude values. Distributing such a large database with GTrace was not ideal. CAIDA's NetGeo [NetGeo] tool, with its ability to determine geographical locations based on the data available in *whois* records, provided a vital resource.

NetGeo is a database and collection of Perl scripts used to map IP addresses to geographical locations. Given an IP address, NetGeo will first search its own local database. If a record for the target address is found in the database, NetGeo will return the requested location information, e.g., latitude and longitude. If NetGeo finds no matching record in its database, it will perform one or more *whois* lookups until it finds a *whois* record for the appropriate network. The NetGeo Perl scripts will then parse the *whois* record and extract location information, which NetGeo both returns to the client and stores in its local database for future use.

The NetGeo database contains tables for mapping world location names (city, state/province/district, country) or US zip codes to latitude/longitude values. Most *whois* records provide enough address information for NetGeo to be able to associate some latitude/longitude value with the IP address. Occasionally the *whois* record only suggests a country or state, in which case NetGeo returns a generic latitude/longitude for that country or state. In preliminary testing, NetGeo has been able to parse addresses and find (albeit sometimes imprecise) latitude/longitude information for 89% of 17,000 RIPE *whois* records, 76% of 700 APNIC *whois* records and for more than 95% of 30,000 ARIN *whois* records.

3.6 Lookup Server

The Lookup Server handles requests from Lookup Clients and tries to determine the location of a host or IP address by executing steps 3, 4 and 5 of the search algorithm. This information is sent back to the client, which then decides whether to use the location information or not depending on the locations it might have received from other Lookup Servers or lookups it performed locally. The Lookup Client selects the location that was obtained from the lowest numbered search step.

The Lookup Server can also be requested by the Lookup Client to execute step 2 of the search algorithm. This is because not all versions of *nslookup* support queries for LOC records. GTrace tests the version of *nslookup* on the machine it is running on to determine if such a request is necessary.

3.7 Location Verifier

The Main Thread invokes the Location Verifier once all the hop threads have died and the trace is complete. The task of the Location Verifier is to check whether the locations obtained for nodes along the path are reasonable. The verifier does not determine new locations for nodes, it only indicates to the user why an existing location might be wrong and where the node could possibly be located.

The verifier algorithm is based on the fact that IP packets can not travel faster than the speed of light. Light travels across different mediums at different speeds: 3.0×10^8 m/s in vacuum, 2.3×10^8 m/s in copper and 2.0×10^8 m/s in fiber [Peterson]. GTrace uses the speed of light in copper for all of its calculations.

For each successive pair of hops that have locations, the verifier algorithm uses the deltas of the round-trip times (RTT) returned by *traceroute* to rule out locations that are physically not possible. *Traceroute* measures RTT rather than one way latency, as this would require control over both end nodes and delays are often not symmetric. Also, one must be cautious with the RTT values since they incorporate several components of delay. The RTT between two nodes has four components: the speed-of-light propagation delay, the amount of time it takes to transmit the unit of data, queuing delays inside the network and the processing time at the destination node to generate the ICMP time exceeded message. *Traceroute* typically sends 40-byte UDP datagrams, so it is safe to assume negligible transmit time. Ideally, for the verifier algorithm one would like the RTT to represent only the propagation delay, but this is not the case due to variable queuing and processing delays, hence it is not possible to set the upper bound on the RTT to a hop. Accordingly the verifier algorithm uses the minimum RTT returned by *traceroute*, as this would represent the best approximation of the propagation delay. Things are further complicated by the fact that the RTT delta between hops k and $k+1$ can be biased because

the return path the ICMP packet takes from hop k can be totally different from the return path it takes from hop $k+1$. The Location Verifier tries to re-determine RTT values for hops it thinks are biased using *ping*.

By default, *traceroute* sends three datagrams each time it increments the TTL to search for the next hop. Changing the value of the q parameter in the GTrace configuration file will modify this behavior. The larger the value of q, the more accurate the estimate of the propagation delay, but large values of q also slow down GTrace as *traceroute* has to send q packets for each hop.

Knowing the geographical distance between two nodes, GTrace can calculate the time-of-flight RTT (the propagation delay at the speed-of-light in copper), compare it against *traceroute*'s value and flag a problem if the RTT is smaller than physically possible. In such a case either the location of the source or of the destination or both is incorrect. The details of the verification algorithm are as follows:

Verifier Algorithm:

1. Ideally, the RTT to hop k in a path should always be less than the RTT to hop $k+1$ or $k+2$... But this is not always true due to queuing delays, asymmetric paths and other delays. We allow a 1ms fudge factor to cover such discrepancies. Thus the RTTs between hops k and $k+1$ should be such that $RTT(k) \leq RTT(k+1) + 1ms$. If this condition does not hold true then the RTT to each of the out-of-order hops preceding hop k is estimated again with *ping*, i.e. till the first hop j preceding k such that $RTT(j) \leq RTT(k+1) + 1ms$. If the RTT estimates obtained using *ping* still do not satisfy the condition $RTT(k) \leq RTT(k+1) + 1ms$, then hop k is not used in the later stages of the verifier algorithm.

2. Cluster the *traceroute* path into regions having similar RTT values. This is based on the assumption that nodes with similar RTTs will tend to be in the same geographic region.

3. For each region identified in the previous step, calculate the time-of-flight RTT for pairs of hops that have locations. If the RTT delta reported by *traceroute* for that pair of hops is smaller than the time-of-flight RTT, flag the pair of hops so that it is corrected in step 5.

4. Repeat step 3 for hops falling on the edges of adjacent regions.

5. Try to "correct" unreasonable location values that were identified in steps 3 and 4 using the reliability of the search step that produced the location match. Adjacent nodes between regions are corrected first because they represent larger and probably more inaccurate locations. Correcting the nodes identified in step 3 follows this. By correct, we mean trying different alternatives for the incorrect location based on the cluster in which it falls, flagging it to the user and not plotting it in the display.

Example:

Consider the trace shown in Fig. 4, where locations are expressed as city names for ease of illustration. The *Search Step* column indicates which step of the search algorithm produced the location for that hop. Step 1 of the verifier algorithm would mark hop 13 as unusable since its RTT is greater than its subsequent hops. In this case it is probably due to the return path from hop 13 being longer than that from hop 14. Next, step 2 of the algorithm would cluster the *traceroute* path into the following regions: 1-4, 5, 6-8, 9-10, 11-12 and 14-16. Step 3 would flag that there is a problem between hops 7 and 8 since it is not possible for a packet to travel from San Francisco to New Jersey in less than a millisecond. Likewise, step 4 would flag a problem between hops 10 and 11. Step 5 would first try to correct hops 10 and 11 since they fall in different regions. Seeing that the location for hop 11 was obtained through step 3 of the search algorithm and hop 10 was from a higher step, the Location Verifier would change hop 10's location to that of hop 11's, in this example to Washington and rerun the algorithm from step 3. This process is repeated until all locations from one hop to the next are physically realistic. In the end the Location Verifier would have indicated to the user that hop 8 is incorrect and is most probably located somewhere near San Francisco. Hops 9 and 10 are also incorrect and may be in Washington with their interfaces labeled San Francisco to

Hop	Node Name	IP Address	Search Step	Location	RTT (ms)
1	pinot-fe2-0-0	(192.172.226.65)	6	San Diego	0.917ms
2	medusa.sdsc.edu	(198.17.46.10)	3	San Diego	0.881ms
3	sdsc-gw.san-bb1.cerf.net	(192.12.207.9)	4	San Diego	1.944 ms
4	pos0-0-155M.san-bb6.cerf.net	(134.24.29.130)	4	San Diego	4.640 ms
5	atm6-0-1-622M.lax-bb4.cerf.net	(134.24.29.142)	4	Los Angeles	9.598 ms
6	pos6-0-622M.sfo-bb3.cerf.net	(134.24.29.233)	4	San Francisco	15.317 ms
7	pos10-0-0-155M.sfo-bb1.cerf.net	(134.24.32.86)	4	San Francisco	16.813 ms
8	192.205.31.29	(192.205.31.29)	6	New Jersey	16.917 ms
9	att-gw.sf.cw.net	(192.205.31.78)	4	San Francisco	81.281 ms
10	corerouter2.SanFrancisco.cw.net	(204.70.9.132)	4	San Francisco	81.254 ms
11	core1.Washington.cw.net	(204.70.4.129)	3	Washington	89.727 ms
12	mix1-fddi-0.Washington.cw.net	(204.70.2.14)	4	Washington	89.708 ms
13	**vsnlpoone.Washington.cw.net**	**(204.189.152.134)**	**4**	**Poone**	**706.301 ms**
14	202.54.6.17	(202.54.6.17)	6	Madras	697.946 ms
15	202.54.6.254	(202.54.6.254)	6	Madras	702.893 ms
16	giasmda.vsnl.net.in	(202.54.6.161)	4	Madras	704.856 ms

Fig. 4 A sample *traceroute* output produced by the first phase of GTrace.

identify the other end of that link.

4. Configuration Files

The configuration options in GTrace are quite flexible. How it functions and executes the search algorithm depends on the contents of two configuration files: *GTrace.conf* and *GTraceMaps.conf*

4.1 GTrace.conf

GTrace.conf specifies the location of the commands GTrace uses and lists databases, text files, Lookup Servers if any, to use in the search algorithm. Fig. 5 shows an example configuration file. This file is automatically generated by the configure scripts while installing GTrace.

4.2 GTraceMaps.conf

The *GTraceMaps.conf* configuration file specifies attributes of the maps that GTrace uses in displays. Users can add their own maps as part of or independent from the existing world hierarchy. Independent maps allow users to describe their own intranet topology and then use GTrace as a graphical debugging tool within their network.

```
#GTrace configuration file

#Paths
TRACEROUTE=/usr/sbin/traceroute –q 3
WHOIS=/usr/bin/whois
PING= /usr/sbin/ping
NSLOOKUP=/usr/sbin/nslookup
DOMAINFILES=/home/ram/gtrace/data
DATABASES=/home/ram/gtrace/db

#Names of databases and text files to be used
#for location lookups. Order is important, list
#them in the order they should be searched.
CITIES=cities.db
AIRPORTS=airport.db

HOSTSLOC=Machine.db,hostnames/ipaddr;
         Hosts.db,ipaddr;
         Organization.db,hostnames/ipaddr;

TEXTFILES=England.txt,hostnames/ipaddr;


#Location of Lookup Servers if any
LOOKUPSRVS=
```

Fig. 5 Sample *GTrace.conf* file

5. GTrace Features

Fig. 6 shows an example of a trace that was executed from University of Colorado, Boulder to CAIDA in San Diego. On the display, the colors of the lines on the map indicate the

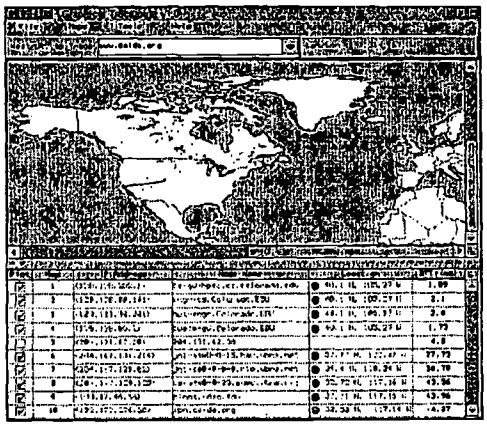


Fig. 6 Example of a trace produced by GTrace

reliability of the location obtained for the endpoints. The colors are decided based on the following criteria:

Green	Both endpoints are authoritative locations.
Yellow	One endpoint is authoritative and the other is a guess whose location is not a country center, state center or obtained from a *whois* record.
Blue	Both endpoints are guesses and the locations of both the endpoints are not a country center, state center or obtained from a *whois* record.
Red	One endpoint is a location that is a country center, state center or obtained from a *whois* record.

The table in the lower section of the display consists of six columns. The first column provides the user with a checkbox that is enabled for each location plotted on the map. The user can disable a checkbox and the corresponding location will be skipped. Locations that are flagged as unreasonable by the Location Verifier are not plotted by default.

The second, third and fourth columns display the hop number, IP address and host name respectively. Clicking on columns three and four will bring up *whois* information for the node.

Column five provides the latitudes and longitudes obtained for each hop. Clicking on this column will provide an explanation of how the location was determined and whether the Location Verifier detected any problems. A small colored ball in front of the latitude and longitude value indicates which search step produced the location. The colors and the search step they represent are given below:

Green	Step 2 LOC record.
Yellow	Step 3 Complete match
Blue	Step 4 Domain parsing file
Cyan	Step 5 Hostname reduction match
Red	Step 6 *whois* record
Gray	Step 7 Country code

The last column shows the smallest of the round trip times returned by *traceroute*. The color of the value indicates how many packets timed out: black implies that no packets timed out, blue implies that one packet timed out, and a value in red indicates that two or more packets timed out.

6. Using GTrace in the Local Environment

System Administrators often use *traceroute* as a debugging tool to identify problems in their network. GTrace provides a visual representation that can facilitate understanding and debugging of their network. It can be used to discover routing loops as well as for deciding routes. For example in a large campus if a path from host A to host B (located in the same building) goes across campus and back, the routing could be fixed to avoid such inefficient paths. GTrace can also be useful from an end user perspective. Students can use the tool to work out the topology of their campus network.

7. Conclusion

GTrace is a handy tool for identifying network topology and routing problems as well as gaining more macroscopic insight into the Internet infrastructure. While GTrace uses several heuristics to determine locations and its

approach does not guarantee accuracy, it is robust and extensible. New databases, new Lookup Servers and learned insights into ISP's naming conventions can easily be added to GTrace. We hope that users and system administrators will find GTrace useful and contribute their own domain parsing files, or even run their own Lookup Servers for community use.

The practical success of GTrace lies in the rules defined for the ".net" domains, since these comprise the majority of hops in many traceroutes. Looking up a ".net" name in the *whois* database is only useful for small localized ISPs. Relying on *whois* heuristics would result in backbone providers' ".net" nodes to all uselessly map to a single corporate headquarters for that provider.

The accuracy of this tool would be much improved if the Internet community maintained LOC records in the DNS. Unfortunately since LOC records are optional, non-trivial in effort to support and without any clear payoff to ISPs, pervasive use of them will probably never occur and geographic visualization of arbitrary Internet infrastructure will continue to require heuristics to determine physical location of nodes.

8. Acknowledgments

We would like to thank kc claffy at CAIDA for suggesting the idea to develop this tool. We would also like to mention a special word of thanks to the following people and institutions: VisualRoute for permission to use their maps and labels, Sleepycat Software for the BerkeleyDB Package, Jim Donohoe for developing NetGeo and to the entire research team at CAIDA who helped with many aspects during the development of GTrace.

Several students (Colorado: Robert Cooksey, Brent Halsey, Jamey Wood, Jeremy Bargen and UCSD: Jim Anderson) wrote graphical *traceroute* tools as class projects in Evi Nemeth's Network System's class. Many good ideas from these students' projects were incorporated into GTrace.

9. Availability and Support

GTrace-1.0 is the current release and it can be downloaded from the GTrace home page at *http://www.caida.org/Tools/GTrace*. The source code comes with the GTrace distribution. Further information on using the tool or how you can contribute domain parsing files can be found on the GTrace home page.

10. Author Information

Ram Periakaruppan is pursuing his Master's degree in Computer Science at the University of Colorado, Boulder. He can be reached at *<ramanath@cs.colorado.edu>*.

Evi Nemeth has been a computer science faculty member at the University of Colorado for years. Currently she is on leave doing the IEC (Internet Engineering Curriculum) project at CAIDA (Cooperative Association for Internet Data Analysis) on the UCSD campus and working furiously to make the publisher's deadline for the third edition of the UNIX System Administration Handbook. She can be reached at *<evi@cs.colorado.edu>*.

References

[AirportCodes] Listing of Airport Codes, http://www.mapping.com/airportcodes.html

[BerkeleyDB] BerkeleyDB Package Distribution, http://www.sleepycat.com

[DBCAIDA] Database files compiled by CAIDA, http://www.caida.org/NetGeo/NetGeo/

[DBNDG] Database file compiled by NDG Software, http://www.dtek.chalmers.se/~d3august/xt/dl/

[Jacobson88] Van Jacobson, Traceroute source code and documentation. Available from: ftp://ftp.ee.lbl.gov/traceroute.tar.Z.

[NetGeo] The Internet Geographic Database, http://www.caida.org/Tools/NetGeo

[OROMatcher] OROMatcher - Regular Expression Package for Java, http://www.savarese.org

[Peterson] Peterson, Larry L., & Davie, Bruce S., Computer Networks - A Systems Approach, Morgan Kaufmann, (1996).

[RFC1876] RFC 1876, Davis, C., Vixie, P., Goodwin, T., and Dickinson I., A means for Expressing Location Information in the Domain Name System, January (1996).

[RFC1918] RFC 1918, Rekhter, Y., Moskowitz, B., Karrenberg, D., Groot, G. J., Lear E., Address Allocation for Private Internets, February (1996).

[Swing] Java Foundation Classes – Swing http://java.sun.com/products/jfc/

[VisualRoute] Maps from VisualRoute, http://www.visualroute.com

Dr. Dobb's Software Tools for the Professional Programmer

Packet Filtering in the SNMP Remote Monitor

Controlling remote monitors on a LAN

William Stallings

William is president of Comp-Comm Consulting of Brewster, MA. This article is based on his recent book, SNMP, SNMPv2, and CMIP: The Practical Guide to Network Management Standards (Addison-Wesley, 1993). He can be reached at stallings@acm.org.

The Simple Network Management Protocol (SNMP) architecture was designed for managing complex, multivendor internetworks. To achieve this, a few *managers* and numerous *agents* scattered throughout the network must communicate. Each agent uses its own management-information database (MIB) of managed objects to observe or manipulate the local data available to a manager.

The remote-network monitoring (RMON) MIB, defined as part of the SNMP framework, provides a tool that an SNMP or SNMPv2 manager can use to control a remote monitor of a local-area network. The RMON specification is primarily a definition of a data structure containing management information. The effect, however, is to define standard network-monitoring functions and interfaces for communicating between SNMP-based management consoles and remote monitors. In general terms, the RMON capability provides an efficient way of monitoring LAN behavior, while reducing the burden on both other agents and management stations; see Figure 1.

The accompanying text box entitled, "Abstract Syntax Notation One (ASN.1)" gives details on defining the communication formats between agents and managers.

The key to using RMON is the ability to define "channels"--subsets of the stream of packets on a LAN. By combining various filters, a channel can be configured to observe a variety of packets. For example, a monitor can be configured to count all packets of a certain size or all packets with a given source address.

To use RMON effectively, the person responsible for configuring the remote monitor must understand the underlying filter and channel logic used in setting it up. In this article, I'll examine this filter and channel logic.

The RMON MIB contains variables that can be used to configure a monitor to observe selected packets on a particular LAN. The basic building blocks are a data filter and a status filter. The data filter allows the monitor to screen observed packets based on whether or not a portion of the packet matches a certain bit pattern. The status filter allows the monitor to screen observed packets on the basis of their status (valid, CRC error, and so on). These filters can be combined using logical AND and OR operations to form a complex test to be applied to incoming packets. The stream of packets that pass the test is referred to as a "channel," and a count of such packets is maintained. The channel can be configured to generate an alert if a packet passes through the channel when it is in an enabled state. Finally, the packets passing through a channel can be captured in a buffer. The logic defined for a single channel is quite complex. This gives the user enormous flexibility in defining the stream of packets to be counted.

Filter Logic

At the lowest level of the filter logic, a single data or status filter defines characteristics of a packet. First, consider the logic for defining packet characteristics using the variables *input* (the incoming portion of a packet to be filtered), *filterPktData* (the bit pattern to be tested for), *filterpktDataMask* (the relevant bits to be tested for), and *filterPktDataNotMask* (which

Petitioners' EX1039 Page 279

indicates whether to test for a match or a mismatch). For the purposes of this discussion, the logical operators AND, OR, NOT, XOR, EQUAL, and NOT-EQUAL are represented by the symbols o, +, --, _, =, and _, respectively.

Suppose that initially, you simply want to test the input against a bit pattern for a match. This could be used to screen for packets with a specific source address, for example. In the expression in Example 1(a), you would take the bit-wise exclusive-OR of *input* and *filterPktData*. The result has a 1 bit only in those positions where *input* and *filterPktData* differ. Thus, if the result is all 0s, there's an exact match. Alternatively, you may wish to test for a mismatch. For example, suppose a LAN consists of a number of workstations and a server. A mismatch test could be used to screen for all packets that did not have the server as a source. The test for a mismatch would be just the opposite of the test for a match; see Example 1(b). A 1 bit in the result indicates a mismatch.

The preceding tests assume that all bits in the input are relevant. There may, however, be some "don't-care" bits irrelevant to the filter. For example, you may wish to test for packets with any multicast destination address. Typically, a multicast address is indicated by one bit in the address field; the remaining bits are irrelevant to such a test. The variable *filterPktDataMask* is introduced to account for "don't-care" bits. This variable has a 1 bit in each relevant position and 0 bits in irrelevant positions. The tests can be modified; see Example 1(c).

The XOR operation produces a result that has a 1 bit in every position where there is a mismatch. The AND operation produces a result with a 1 bit in every relevant position where there is a mismatch. If all of the resulting bits are 0, then there is an exact match on the relevant bits; if any of the resulting bits is 1, there is a mismatch on the relevant bits.

Finally, you may wish to test for an input that matches in certain relevant bit positions and mismatches in others. For example, you could screen for all packets that had a particular host as a destination (exact match of the DA field) and did not come from the server (mismatch on the SA field). To enable these more complex tests to be performed, use *filterPktDataNotMask*, where:

- The 0 bits in *filterPktDataNotMask* indicate the positions where an exact match is required between the relevant bits of *input* and *filterPktData* (all bits match).
- The 1 bits in *filterPktDataNotMask* indicate the positions where a mismatch is required between the relevant bits of *input* and *filterPktData* (at least one bit does not match).

For convenience, assume the definition in Example 2(a). Incorporating *filterPktDataNotMask* into the test for a match gives Example 2(b).

The test for a mismatch is slightly more complex. If all of the bits of *filterPktDataNotMask* are 0 bits, then no mismatch test is needed. By the same token, if all bits of *filterPktDataNotMask* are 1 bits, then no match test is needed. However, in this case, *filterPktDataNotMask* is all 0s, and the match test automatically passes *relevant_bits_differento*0=0. Therefore, the test for mismatch is as in Example 2(c).

The logic for the filter test is summarized in Figure 2. If an incoming packet is to be tested for a bit pattern in a portion of the packet, located at a distance *filterPktDataOffset* from the start of the packet, the following tests will be performed:

- Test #1: As a first test (not shown in the figure), the packet must be long enough so that at least as many bits in the packet follow the offset as there are bits in *filterPktData*. If not, the packet fails this filter.
- Test #2: Each bit set to 0 in *filterPktDataNotMask* indicates a bit position in which the relevant bits of the packet portion should match *filterPktData*. If there is a match in every desired bit position, then the test passes; otherwise the test fails.
- Test #3: Each bit set to 1 in *filterPktDataNotMask* indicates a bit position in which the relevant bits of the packet portion should not match *filterPktData*. In this case, the test is passed if there is a mismatch in at least one desired bit position.

A packet passes this filter if and only if it passes all three tests.

Why use the filter test? Consider that you might want to accept all Ethernet packets that have a destination address of "A5"h but do not have a source address of "BB"h. The first 48 bits of the Ethernet packet constitute the destination address and the next 48 bits, the source address. Example 3 implements the test. The variable *filterPktDataOffset* indicates that the pattern matching should start with the first bit of the packet; *filter PktData* indicates that the pattern of interest consists of "A5"h in the first 48 bits and "BB"h in the second 48 bits; *filter PktDataMask* indicates that the first 96 bits are relevant; and

Petitioners' EX1039 Page 280

filterPktDataNotMask indicates that the test is for a match on the first 48 bits and a mismatch on the second 48 bits.

The logic for the status filter has the same structure as that for the data filter; see Figure 2. For the status filter, the reported status of the packet is converted into a bit pattern. Each error-status condition has a unique integer value, corresponding to a bit position in the status-bit pattern. To generate the bit pattern, each error value is raised to a power of 2 and the results are added. If there are no error conditions, the status-bit pattern is all 0s. An Ethernet interface, for example, has the error values defined in Table 1. Therefore, an Ethernet fragment would have the status value of 6(21+22).

Channel Definition

A channel is defined by a set of filters. For each observed packet and each channel, the packet is passed through each of the filters defined for that channel. The way these filters are combined to determine whether a packet is accepted for a channel depends on the value of an object associated with the channel (*channelAcceptType*), which has the syntax *INTEGER {acceptMatched(1), acceptFailed(2)}*.

If the value of this object is *acceptMatched(1)*, packets will be accepted for this channel if they pass both the packet-data and packet-status matches of at least one associated filter. If the value of this object is *acceptFailed(2)*, packets will be accepted to this channel only if they fail either the packet-data match or the packet-status match of every associated filter.

Figure 3 illustrates the logic by which filters are combined for a channel whose accept type is *acceptMatched*. A filter is passed if both the data filter and the status filter are passed; otherwise, that filter has failed. If you define a pass as a logical 1 and a fail as a logical 0, then the result for a single filter is the AND of the data filter and status filter for that filter. The overall result for a channel is then the OR of all the filters. Thus, a packet is accepted for a channel if it passes at least one associated filter pair for that channel.

If the accept type for a channel is *acceptFailed*, then the complement of the function just described is used. That is, a packet is accepted for a channel only if it fails every filter pair for that channel. This would be represented in Figure 3 by placing a NOT gate after the OR gate.

Channel Operation

The value of *channelAcceptType* and the set of filters for a channel determine whether a given packet is accepted for a channel or not. If the packet is accepted, then the counter *channelMatches* is incremented. Several additional controls are associated with the channel: *channelDataControl*, which determines whether the channel is on or off; *channelEventStatus*, which indicates whether the channel is enabled to generate an event when a packet is matched; and *channelEventIndex*, which specifies an associated event.

When *channelDataControl* has the value off, then, for this channel, no events may be generated as the result of packet acceptance, and no packets may be buffered. If *channelDataControl* has the value on, then these related actions are possible.

Figure 4 summarizes the channel logic. If *channelDataControl* is on, then an event will be generated if: 1. an event is defined for this channel in *channelEventIndex*; and 2. *channelEventStatus* has the value *eventReady* or *eventAlwaysReady*. If the event status is *eventReady*, then each time an event is generated, the event status is changed to *eventFired*. It then takes a positive action on the part of the management station to reenable the channel. This mechanism can therefore be used to control the flow of events from a channel to a management station. If the management station is not concerned about flow control, it may set the event status to *eventAlwaysReady*, where it will remain until explicitly changed.

Summary

The packet-filtering facility of RMON provides a powerful tool for the remote monitoring of LANs. It enables a monitor to be configured to count and buffer packets that pass or fail an elaborate series of tests. This facility is the key to successful remote-network monitoring.

Abstract Syntax Notation One (ASN.1)

Steve Witten

Petitioners' EX1039 Page 281

Steve, a software engineer for Hewlett-Packard, specializes in network testing and measurement. You can contact him at stevewi@hpspd.spd.hp.com.

SNMP protocol and MIB are formally defined using an abstract syntax. This allowed SNMP's authors to define data and data structures without regard to differences in machine representations. This abstract syntax is an OSI language called "abstract syntax notation one" (ASN.1). It is used for defining the formats of the packets exchanged by the agent and manager in the SNMP protocol and is also the means for defining the managed objects.

ASN.1 is a formal language defined in terms of a grammar. The language itself is defined in ISO #8824. The management framework defined by the SNMP protocol, the SMI, and the MIB use only a subset of ASN.1's capabilities. While the general principles of abstract syntax are good, many of the bells and whistles lead to unnecessary complexity. This minimalist approach is taken to facilitate the simplicity of agents.

Listings One through Three show an MIB, using a fictitious enterprise called SNMP Motors. Listing One is an ASN.1 module that contains global information for all MIB modules. Listing Two, another ASN.1 module, contains the definitions of specific MIB objects. Finally, Listing Three illustrates manageable objects.

Once data structures can be described in a machine-independent fashion, there must be an unambiguous way of transmitting those structures over the network. This is the job of the transfer-syntax notation. Obviously, you could have several transfer-syntax notations for an abstract syntax, but only one abstract-syntax/transfer-syntax pair has been defined in OSI. The basic encoding rule (BER) embodies the transfer syntax. The BER is simply a recursive algorithm that can produce a compact octet encoding for any ASN.1 value.

At the top level, the BER describes how to encode a single ASN.1 type. This may be a simple type such as an *Integer*, or an arbitrarily complex type. The key to applying the BER is understanding that the most complex ASN.1 type is nothing more than several simpler ASN.1 types. Continuing the decomposition, an ASN.1 simple type (such as an *Integer*) is encoded.

Using the BER, each ASN.1 type is encoded as three fields: a tag field, which indicates the ASN.1 type; a length field, which indicates the size of the ASN.1 value encoding which follows; and a value field, which is the ASN.1 value encoding.

Each field is of variable length. Because ASN.1 may be used to define arbitrarily complex types, the BER must be able to support arbitrarily complex encodings.

It is important to note how the BER views an octet. Each octet consists of eight bits. BER numbers the high-order (most significant) bit as bit 8 and the low-order (least significant) bit as bit 1. It's critical that this view be applied consistently because different machine architectures use different ordering rules.

Figure 1 RMON description.

Figure 2 Logic for the filter test.

Figure 3 Logic by which filters are combined for a channel whose accept type is acceptMatched.

Figure 4: Logic for channel filter.

```
procedure packet_data_match;
begin
  if (result = 1 and channelAcceptType = acceptMatched) or
     (result = 0 and channelAcceptType = acceptFailed)
  then begin
     channelMatches := channelMatches + 1;
     if channelDataControl = on
     then begin
             if (channelEventStatus _ eventFired) and
                (channelEventIndex _ 0) then generate_event;
             if (channelEventStatus = eventReady) then
                  channelEventStatus := eventFired
             end;
      end;
```


Petitioners' EX1039 Page 282

```
end;
```

Example 1: Testing the input against a bit pattern for a match.

```
(a) (input XOR filterPktData) = 0 --> match

(b) (input XOR filterPktData) (does not equal) 0 --> mismatch

(c) ((input XOR filterPktData) (and) filterPktDataMask) =
      0 --> match on relevant bits
    ((input XOR filterPktData) (and) filterPktDataMask) (does not equal)
      0 --> mismatch on relevant bits
```

Table 1: Ethernet-interface error values.

Bit	Error
0	Packet is longer than 1518 octets.
1	Packet is shorter than 64 octets.
2	Packet experienced a CRC or alignment error.

Example 2: Assuming the definition in (a), incorporating filterPktDataNotMask into the test for a match, you end up with (b). Test for a mismatch is shown in (c).

```
(a) relevant_bits_different =
      (input XOR filterPktData) (and) filterPktDataMask

(b) (relevant_bits_different (and) filterPktDataNotMask') =
      0 --> successful match

(c) ((relevant_bits_different (and) filterPktDataNotMask) (does not equal) 0) +
      (filterPktDataNotMask = 0) --> successful mismatch
```

Example 3: Launching a filter test.

```
filterPktDataOffset   = 0
filterPktData         = "00 00 00 00 00 A5 00 00 00 00 00 BB"h
filterPktDataMask     = "FF FF FF FF FF FF FF FF FF FF FF FF"h
filterPktDataNotMask  = "00 00 00 00 00 00 FF FF FF FF FF FF"h
```

Listing One

```
SNMP-motors-MIB DEFINITIONS ::= BEGIN
IMPORTS
    enterprises
        FROM RFC1155-SMI;
SNMP-motors OBJECT IDENTIFIER ::= { enterprises 9999 }
expr        OBJECT IDENTIFIER ::= { SNMP-motors 2 }
END
```

Listing Two

```
SNMP-motors-car-MIB DEFINITIONS ::= BEGIN
IMPORTS
    SNMP-motors
        FROM SNMP-motors-MIB;
IMPORTS
```


Petitioners' EX1039 Page 283

```
    OBJECT TYPE, ObjectName, NetworkAddress,
    IpAddress, Counter, Gauge, TimeTicks, Opaque
        FROM RFC1155-SMI;
car OBJECT IDENTIFIER ::= { SNMP-motors 3 }
-- this is a comment
-- Implementation of the car group is mandatory
-- for all SNMP-motors cars.
-- ( the rest of the SNMP-motors-car-MIB module )
END
```

Listing Three

```
carName OBJECT TYPE
    SYNTAX DisplayString (SIZE (0..64))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "A textual name of the car."
    ::= { car 1 }
carLength OBJECT TYPE
    SYNTAX INTEGER (0..100)
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The length of the car in feet."
    ::= { car 2 }
carPassengers OBJECT TYPE
    SYNTAX INTEGER (0..4)
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The number of passengers in the car."
    ::= { car 3 }
carPassengerTable OBJECT TYPE
    SYNTAX SEQUENCE OF CarPassengerEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
        "A table describing each passenger."
    ::= { car 4 }
carPassengerEntry OBJECT TYPE
    SYNTAX SEQUENCE OF CarPassengerEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
        "A entry table describing each passenger."
    ::= { carPassengerTable 1 }
CarPassengerEntry ::= SEQUENCE {
    carPindex
        INTEGER,
    carPname
        DisplayString,
    carPstatus
        INTEGER
}
carPindex OBJECT TYPE
    SYNTAX INTEGER (1..4)
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "Index for each passenger which ranges from
                1 to the value of carPassengers."
    ::= { carPassengerEntry 1 }
carPname OBJECT TYPE
```


Petitioners' EX1039 Page 284

```
    SYNTAX DisplayString (SIZE (0..64))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
        "The name of the passenger."
    ::= { carPassengerEntry 2 }
carPstatus OBJECT TYPE
    SYNTAX INTEGER { other(1),driver(2) }
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
        "The status of the passenger."
    ::= { carPassengerEntry 3 }
```

Copyright © 1994, *Dr. Dobb's Journal*

Copyright © 2004 Dr. Dobb's Journal, Privacy Policy. Comments about the web site: webmaster@ddj.com

Title: Advanced Methods for Storage and Retrieval in Image Databases

Prototype Lead: Dr. Cris Koutsougeras, (504)862-3369, ck@eecs.tulane.edu

Proposed Funding Source: ESDIS Prototyping

Type: Prototype

Category: Engineering

Primary Purpose: Technology Evolution

Key Requirements Addressed: IMS-0150, IMS-0160, and IMS-190

Key Risks Addressed: 089 and 105

Results Need Date: N/A

Objective

We will continue to develop image/granule analysis and retrieval methods based on queries over metadata descriptions of the images. We will be working closely with members of the MODIS science team and, for development and testing purposes, will base our research on MODIS products. In particular, we will use a land surface Level 3 product (MOD13 including NDVI) and an atmospheric product. The atmospheric product is likely to be the Level 2 Cloud Product (MOD06) but discussions are still underway with MODIS participants about this and about a possible oceanic product as well. The prototype that currently exists at Tulane University will be extended to include these products. The principle on which the Tulane prototype is based is that an abstraction of an image/granule in the form of metadata is immediately accessible, not the image/granule itself. Queries by researchers are performed over the metadata. Selected images/granules are then transmitted to the researcher on a delayed basis. Extending the prototype using these diverse MODIS products will enable us to not only directly benefit the MODIS science team but also to develop methods that will assist DAAC users in general.

On a broader front, we will be addressing the issue of a uniform interface for heterogeneous data (Level 3 requirement IMS-0150), providing different levels of user interaction support (IMS-0160), and investigating how to save knowledge between metadata searches (IMS-0190). The effort will be a step toward defining precisely the set of user services needed (Risk 089) and reducing the likelihood that significant data will be overlooked due to the large volumes of data managed by EOSDIS (Risk 105).

Approach

A prototype has been implemented as an end-to-end, client-server testbed (Fig. 1). This testbed, provides a close analog to the ECS Science Data Processing Segment (SDPS) subsystems. The prototype has at its core the ObjectStore database and most of the current contents are based on the NDVI product. The interface is interactive and web-based. Queries are performed over simple, atomic data as well as some structured data, e.g., image histograms. A set of metadata has been defined but is subject through this study to modification as we concentrate on new products and interact with the MODIS science team.

Petitioners' EX1039 Page 286

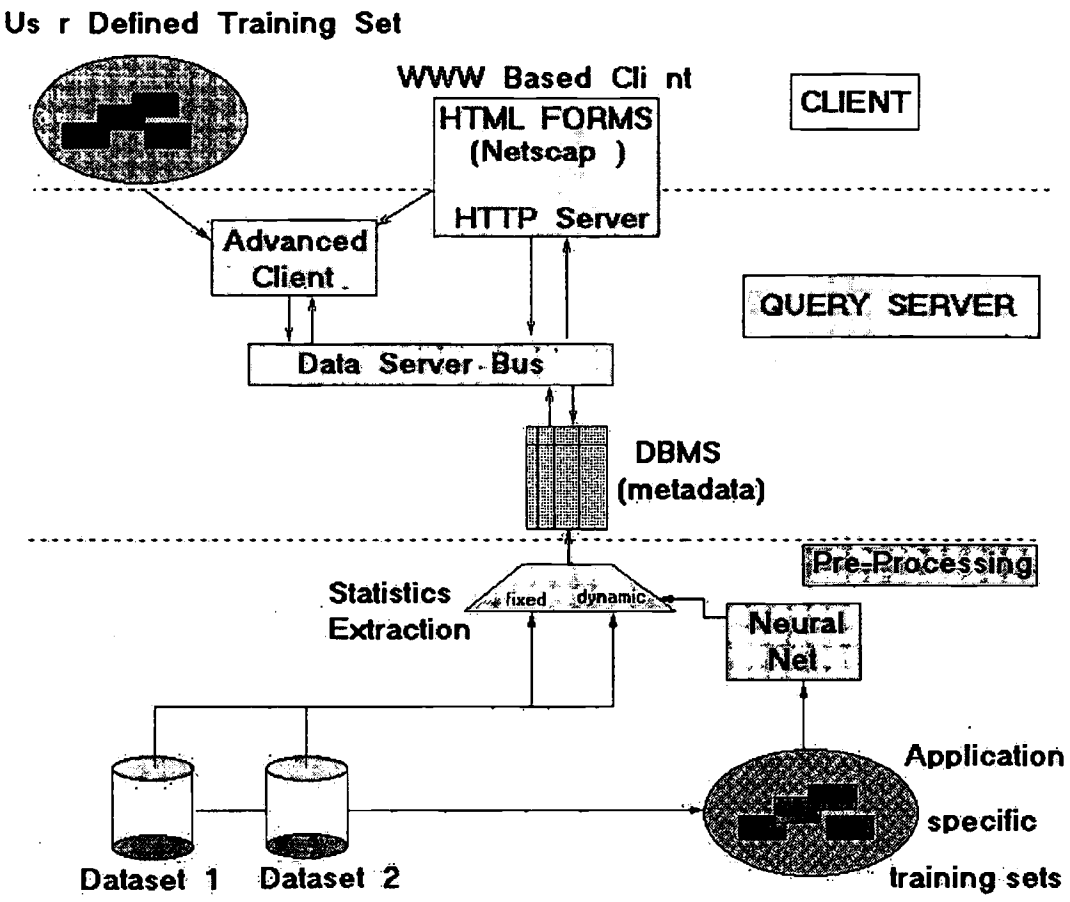


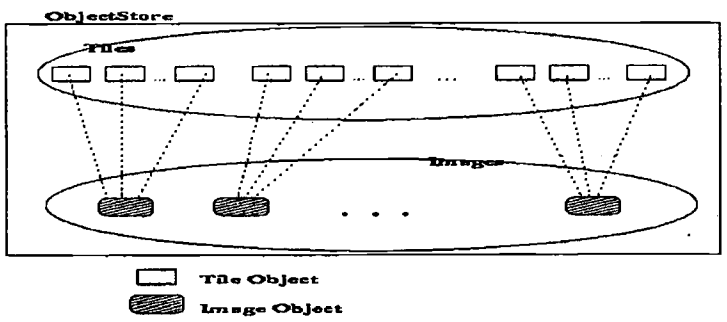
Fig. 1: Architecture of the Tulane EOS Query Prototype

Architecture of Present System. Because ObjectStore is an object-oriented database. the schema is discussed in terms of the object-oriented concepts of classes. data members. and methods. It is important to distinguish between geodata and metadata. Geodata. sometimes called granule level metadata. deals with the context of the image and includes information such as: date. flight. time. perspective (angle). instrument. latitude. data format. longitude, and mission.

Metadata deals with the content of the image and includes such things as the mean irradiance. cloud cover percentage. and texture measures.

The Database Classes. There are two classes employed -- image and tile. An object of class image contains an image and the geodata associated with it. An object of class tile contains the descriptive metadata and the identities of the image/subimage to which the object pertains. "Tile" is the word we use to designate the statistics associated with any one member of the quad tree frames of an image. The tile objects are collected into a container set called **tiles** and the image objects are collected into a container set called **images.** Images are stored in the database strictly for development purposes. In a deployable system. the images would be stored on another media in a data warehouse.

Fig. 2 depicts the relationship among the objects and their container sets. There is a many-to-one relationship between the tile objects and image objects. Each tile object contains the metadata for a portion of the image. Each tile object is stored separate from the image object to which it refers. This permits (1) uniform querying of metadata at the image and subimage levels (2) changing the tiling protocol or philosophy without having to change the database implementation.



Petitioners' EX1039 Page 287

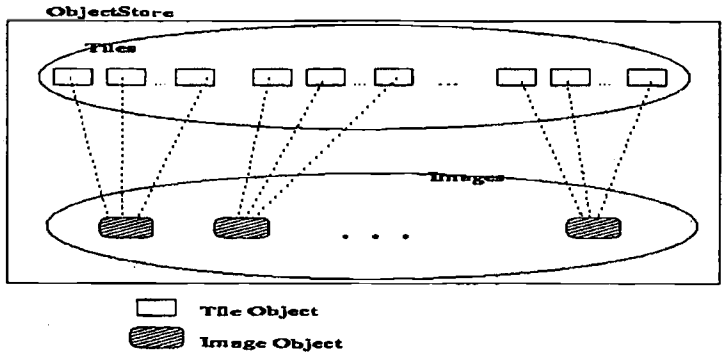


Fig. 2: Relationship Among Objects

Table 1 shows several of the two dozen metadata members of a tile object in the prototype implementation. The first three members, TileID, ImageId, and Channel, identify the tile and correlate it to an image object. The remaining data members are metadata values over which queries are performed.

Table 1: Tile Object Data Members

Metadata	Data Type	Description
TileId	String	Tile identity as described in Fig. 2
ImageId	Integer	Image of which tile is a subimage
Channel	Integer	Instrument band number
PixMean	Real	Spatial mean of all pixels
PixStdDev	Real	Standard deviation of pixel values
PixMode	Integer	Pixel value mode statistic
PixMin	Integer	Minimum pixel value
PixMax	Integer	Maximum pixel value
Histogram	Integer[264]	Number of pixels in each histogram bin
CloudPixels	Real	The fraction of the tile having cloud cover
NullPixels	Real	The fraction of pixels having null values

Petitioners' EX1039 Page 288

FourPhase	Real[20][20]	The phase angles of the Fourier transform
FourAmp	Real[20][20]	The amplitude of the Fourier transform
Wavelet	Real[20][20]	Wavelet (Haar) coefficients
Amplitude	Real[20]	Slope Real Fractal dimension

Sample Queries for the Proposed System. The purpose of metadata collection is to make accessible to the Earth science community the granules collected daily. To profile by example the kinds of uses that an EOS DAAC must be responsive to, we present two typical query scenarios.

Scenario 1 User Goal: Find areas experiencing recent and ongoing deforestation

> **Approach** Using the MODIS NDVI 100km by 100km, look for both "high" standard deviation and "high" roughness. Omit tiles having a high percentage of cloudiness.
>
> **Discussion** The tiles having the above characteristics are likely the ones representing the boundaries of the forest.

Scenario 2 User Goal: Study lee-wave cloud formation – clouds having a distinctive linear cloud pattern.

> **Approach** Find (using any means) a set of tiles having the desired pattern and another set lacking it within the cloud product (MOD06). Cluster the training set using the Fourier signatures to define the texture of the desired tiles. Use the query system to search for images having a similar texture.
>
> **Discussion** This is an example of an "advanced client" query. The researcher must put more effort into the initial search. Afterwards, the work can be reused by the researcher as well as by others.

Relationship of the Proposed Research to the Present Tulane Prototype. Presently, a researcher must be familiar with the nature of the metadata and their potential applications. This is because the researcher has to be able to express what (s)he is searching for and express the search requirements in terms of the metadata. The prototype web site performs simple queries over single-valued metadata variables such as

Retrieve images and subimages having a standard deviation of intensity less than x .

On structured data such as histograms, somewhat more advanced queries are possible such as submitting an exemplar histogram and retrieving the images/subimages that have similar histograms.

The next step is putting in place an interface that assists this translation in accordance with the spirit of Level 3 requirement IMS-0160. Although one could envision this interface as being so sophisticated that it is able to translate a sort of natural language query into a query involving the ground procedures, classes, and metadata of the database, it would first be necessary to establish the concept at a more modest scale.

Evaluation Method/Criteria

We will work in conjunction with MODIS science team members. These members will have remote access to the web site and will be able to provide specific requirements and provide feedback based on hands-on experience with the prototype.

Potential Impact

The prototype will address three significant IMS level 3 requirements: (1) uniform user interfaces, (2) interaction methods for different researchers with different skill levels, and (3) saving search knowledge between query sessions by researchers. It will address two key risks: (1) lack of well-defined user services and (2) overlooking significant information due to the volume of data managed. Additionally, there is a short-term impact that should not be overlooked. It will be of direct and immediate benefit to the MODIS science team members.

Milestones and Deliverables

Scope of Work. Using the current prototype as a basis, Tulane University will extend the system by emphasizing two (and possibly three) MODIS Level 2 and 3 products. The extensions include providing query support that requires different skill levels of researchers using the system, saving search knowledge between query sessions, and developing better approaches to characterizing the information at the subimage/subgranule level. The prototype made available to MODIS science team members may entail revising the metadata as now

Petitioners' EX1039 Page 289

defined based on their feedback.

The duration of the work outlined in this SOW is 12 months.

Summary of Tasks. The efforts during this period of performance will target further refinements to the existing work. More specifically, we target the issues of efficient storage (space problem) and efficient retrieval. We can build on the framework that has been created, refine it, and make it more user friendly.

Further development of the prototype will follow two directions: (1) Development of methods that use the available storage space more effectively (see Tasks 2 and 3); (2) Development of interface mechanisms that allow greater query flexibility for researchers having more expertise in data management (see Tasks 4 and 5).

Task 1: Project Management. Provide monthly progress reports to the Project Manager and Technical Lead. These 1-2 page reports will summarize the previous month's progress, plans for the upcoming month, and any identified issues or impediments to progress. These reports will also provide estimated dollar and labor contract expenditures for that month.

Task 2: Image Decomposition. Decomposition by quad tree, as described in the Approach section, is often preferred by scientists using land products. Frequently the phenomenon sought by such scientists is not distributed across the entire image. Almost always, it is contained in a subimage and the criteria used to find it will often be masked by the image as a whole. That is why the present prototype computes the metadata over each tile in a quad tree.

Yet, the quad tree has drawbacks. Atmospheric and oceanic scientists sometimes prefer other forms of decomposition. In a quad tree, the chief object in an image, say, a hurricane, is may be partitioned among different tiles. Clearly metadata particular to the storm will be washed out by the background that surrounds the parts. Other decomposition methods require the use of criteria based on image content. This might be pixel values or texture. However, decomposing the image into "blobs" that represent areas of uniformity is a plausible approach. Blobs fall short of complete image segmentation by region in that they need not fully partition the image and may overlap. They constitute an augmentation of the quad tree decomposition, not a replacement for it. Metadata should be computed for each blob in addition to each quad tree tile. A blob will more likely contain a coherent object in its entirety.

To illustrate, examine Fig. 3. Using ellipses as the basic geometry of a blob, the significant features on an image are enclosed. Shapes other than ellipses will be investigated. With respect to the database schema, minimal changes are necessary. A blob description will just become an additional tile in the illustration of Fig. 2. The metadata for a blob is the same as that for a tile.

Task 3: Data Space Conservation. Data concerning terabytes of mages must be immediately accessible. To conserve space, we will investigate ways to discover parts of images (partial images) that reoccur and thus can be seen as building blocks. These can be helpful in composing an image (or many images) but metadata descriptors of each of these components need to be stored only once. As a simplistic example, consider an image that contains large bodies of water over which there is a uniform distribution of pixel values. If these areas were represented as repetitions of a single tile, we would only need to store a detailed (sub)image description of this tile and then describe the large bodies of water as a (structured) collection of pointers to it, thus saving substantially in the space required to store the image metadata.

Fig. 3: Blob Decomposition of an Image

There are a couple of different possibilities for discovering building blocks. First, we can try to identify reoccurring tiles within an image. Second, we can try to discover tiles that are useful in many images. The process of discovery must be automated and run at ingest time. This problem has properties similar those of binpacking and thus an optimal solution will not be easy to find. However, evolutionary algorithm techniques are promising and may provide good practical solutions The issues to be addressed in this task are: (1) determining the geometry of the tiles, (2) tradeoffs in space savings in conjunction with the geometry of the tiles (this relates to the frequency of reoccurrence), and (3) evolutionary algorithm techniques for optimal decompositions.

Petitioners' EX1039 Page 290

Task 4: Advanced and Reconstructive Metadata. During the year just past, we examined and chose metadata that are statistically significant, that have high information content, and that researchers are already familiar with. We did not (and were not required to) incorporate mechanisms for using the more highly structured metadata (such as texture and transform coefficients) into queries. We did go beyond the requirements of the previous grant and develop means of using histogram information within queries.

In the coming year we will develop query mechanisms to complement all the metadata. Additionally, because we have defined and are collecting metadata from which the image can be reconstructed at reduced resolution, we will develop mechanisms by which the researcher can define her/his personal metadata and collect it from any set of images defined via less advanced queries. For example, consider the Fourier coefficients. They require much storage but contain much information. A researcher might reconstruct the image or tiles from an image using the Fourier coefficients then compute the eigenvalues of the pixel autocorrelation matrix. This is the sort of query for which we can provide the basic structure through this task.

Task 5: Query Reuse. The result of a query is a set of image and tile id's. The simplest approach to query reuse (and the one we will start with) is to associate with various sets a "cluster condition," a Boolean statement which each member of the set satisfies. This leads to the ability to reuse a set when the condition reoccurs. Next we will elaborate this in the form of an inverted file for which the key is a tile id from which the complete set of cluster conditions (seen thus far) satisfied by that tile can be accessed. This will achieve a powerful query reuse capability.

Deliverables. NASA will have the right to examine the software at any time. Provision to FTP software to NASA will be made on an informal basis. Three additional deliverables are proposed.

> **Item A: Decomposition approach presented as written document. This document will define the approaches used for decomposition (other than the quad tree) and the reasons for which one was selected. The issues of feasibility of performing the method upon ingest will be described together with experiences using the method. This is the culmination of Task 2.**
>
> **Item B: Operational end-to-end WWW-based prototype allowing direct query of fixed (statistical) metadata, utilization of advanced queries that incorporate structured metadata, and allow a researcher to utilize reconstructive metadata (e.g., Fourier coefficients) to define her/his personal metadata.**
>
> **Item C: Final report summarizing "lessons learned," focusing primarily on the architecture of the prototype, the design of the components that constitute the query interface, and changes that would be appropriate if another database product were utilized as the core of the system.**

Bear in mind that in additional to the deliverables, the MODIS science team will be able to gain direct benefit from the project beginning in late September and continuing throughout the prescribed period of performance.

Petitioners' EX1039 Page 291

Measurement and Analysis of the Digital DECT Propagation Channel*

Fulvio Babich[1], Giancarlo Lombardi[1], Steno Schiavon[2], Elvio Valentinuzzi[1]

1: Dipartimento di Elettrotecnica, Elettronica e Informatica,
Università di Trieste, Via A. Valerio 10, I-34127 Trieste, Italy
Phone: +39-40-676-3458; Fax: +39-40-676-3460
babich, lombardi, valent@stelvio.univ.trieste.it
http://ingsun1.univ.trieste/~{babich, lombardi}

2: TELITAL S.p.A., Viale Stazione di Prosecco 5/B, I-34010 Sgonico (TS), Italy,
Phone: +39-40-4192-244; Fax: +39-40-251257; steno.schiavon@rs1.telital.it

Abstract – In this paper, an experimental setup is presented, to measure bit error patterns over a DECT indoor radio channel. A prefixed bit sequence has been exchanged on the air between a mobile and a fixed part, using a DECT modem. DECT interferers were active in the environment during the experiments. Error patterns have been obtained from received sequences, aligning and comparing them with the transmitted ones. They have been stored in real time on a mass memory, by means of a data acquisition board, built for this purpose. Some results are shown, inherent to the Packet Error Distribution (PED) and to the burst and interburst length distributions, obtained from the acquired database. Finite State Markov Channel models have been determined, using measurement conditions, to reproduce and verify empiric results.

1 Introduction

Wireless communications are experiencing a considerable growth, due to their flexibility. The quick increase of the subscriber number requires to reconsider system architecture, in order to adapt it to the propagation environment, as carefully as possible, and, consequently, to obtain capacity gain. Given that urban and indoor wireless communications are the most requested from users, Cordless Telecommunication (CT) systems are being perfectioned, because they allow better coverage and capacity, with lower power expense, than traditional cellular systems. Among European standards, DECT is the most modern digital standard, providing a broad range of services.

Being the mobile propagation channel (with interference) the main reason of degradation of the transmitted signal, it may be useful to evaluate the channel correlation properties, to increase system capacity by taking into account channel memory [1]. In fact, the traditional techniques of coding and interleaving destroy error correlation at the receiver, but do limit system capacity. It is useful, then, to study the correlation properties of bit error sequences and to find models reproducing their behavior. Generally, bit error streams on a wireless channel are obtained, through simulation or as post-processing of experimental analog measurements [2, 3, 4].

In this paper, an experimental setup is presented, to measure and store bit error patterns over a DECT indoor radio channel. Measurements have been performed, building a database of bit error sequences in an indoor environment. The packet error distribution (PED) and the burst/interburst length distributions have been obtained from some streams of the database and compared with the distribution obtained from a Finite State Markov Channel model, proposed in [5] and whose parameters are estimated from measurement conditions.

The measurement setup is described in Section 2, while measurement execution is outlined in Section 3. Results about PED and burst/interburst length distributions are discussed in Section 4 and 5, respectively, while some conclusions are drawn in Section 6.

2 Measurement Setup

The measurements have been performed, using as transmitter and as receiver, two radio communication testers for DECT systems (CMD60, manufactured by R&S), respectively. The scheme of measurement apparatus is displayed in Figure 1. A continuous TTL

*This work has been partially supported by MURST "ex-quota 40%". Italy.

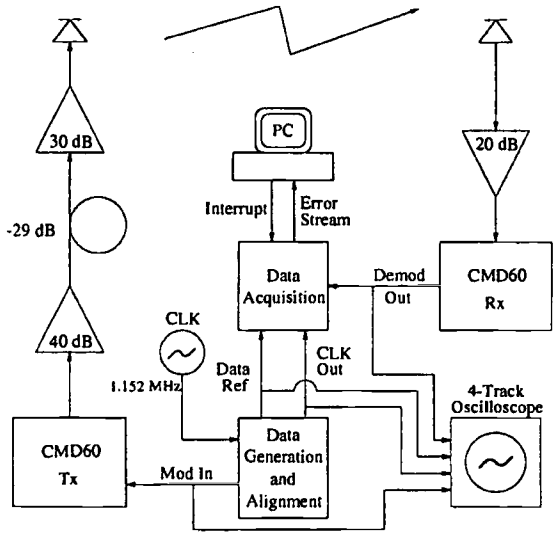


Figure 1: Scheme of the Measurement Setup.

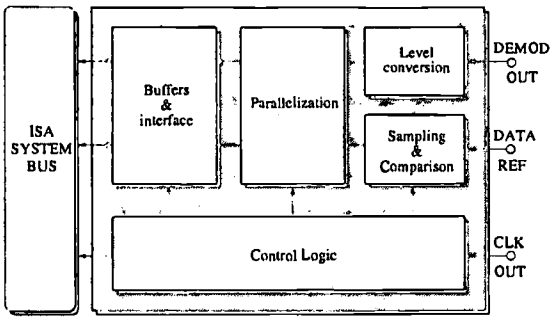


Figure 2: Scheme of the data acquisition board.

binary sequence, having DECT bit rate (1.152 Mb/s), is generated and supplied to the first tester, where it modulates a DECT carrier at 1897.344 MHz in the DECT GMSK format ($BT = 0.5$). This signal is amplified by 40 dB, transmitted over a 80 m coaxial cable (attenuation 29 dB) to the mobile end, where it is amplified again, by 30 dB, before being fed to a discone transmitting antenna.

The receiving part is made by another discone antenna, similar to previous, connected by a 5 m coaxial cable to a 20 dB low noise amplifier, that supplies the signal to the second tester, where the received binary sequence is obtained by a frequency discriminator. This sequence is aligned with the transmitted sequence, used as reference, because the delay of the measurement chain is about one and a half bit.

The comparison between transmitted and received sequence and the storage of the error sequence inside a PC Pentium 133 MHz are made, using a dedicated acquisition digital board, controlled via software using the interrupt management. The sampling and decision circuitry of the received signal are also on the board. It is composed by the blocks showed in Figure 2. The demodulated sequence (DEMOD OUT), having a Gaussian bit shape, is sampled and converted to TTL levels, to be compared with the reference sequence (DATA REF). The high rate of the data requires a parallel acquisition: shift registers, suitably joined to the serial data line, are used. The parallelized data are stored inside buffers, to be available on the ISA System Bus, carrying them to the PC. The control circuitry is for address and interrupt management.

3 Measurement Execution

The measurements have been executed in the laboratory and office wing of a PCS factory, near Trieste. The plan can be found in [6]. The basic structure of the floor is given by a very long hallway, terminated by a large laboratory and interrupted by metal fire-cut doors. Along the hallway, several medium sized room (in mean 5 × 5 m) and stairs are displaced. The furnishing is typical of offices and laboratories, with several metal cabinets and work benches. Further details are given in [6, 7].

The experiments have been carried on after office time, so that very few people was in the environment and stationarity can be assumed. On the other side, several DECT terminals were active and it has been observed that interference is the main reason of error during measurement execution. The receiving antenna is fixed on a 2.10 m dielectric pole, while the transmitting one is moving and kept at a 1.90 m height. The mobile antenna has been continuously displaced along straight and circular paths. Straight paths have a minimum length of 7 m (in the rooms) and a maximum length of 35 m (in the main hallway). The antenna was held on a vertical pole mounted on a trolley and moved by a person, with speed $\sim$ 0.4 m/s. Circular paths have a ray of 1.5 m. The antenna is held on a horizontal arm, mounted on a vertical pole rotating around its axis by means of an electromagnetic engine. The angular speed impressed by the engine is 2π rad/min, corresponding to an antenna peripheric speed of 0.16 m/s.

Three positions of the receiving antenna have been chosen, two in rooms along the hallway and one in the lab terminating the hallway. The paths of the transmitter has been chosen, trying to cover many rooms of the floor. The measurement along every path has been carried on with transmitter constant power. The experiment at any path has been repeated varying the signal power at modulator output among the values -15, -25 and -35 dBm, respectively (about 40 dB have

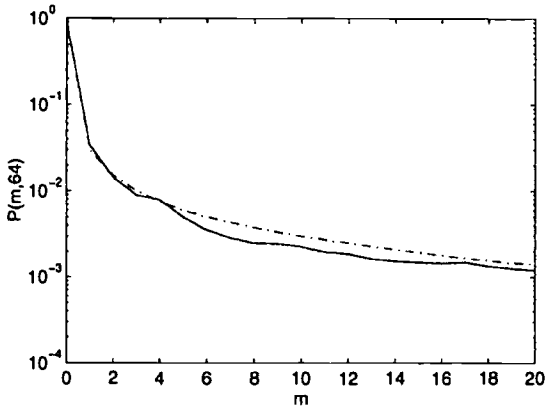


Figure 3: Packet Error Distribution $P(n,m)$ with $n = 64$. Solid line: Measure; Dashed Line: Theory.

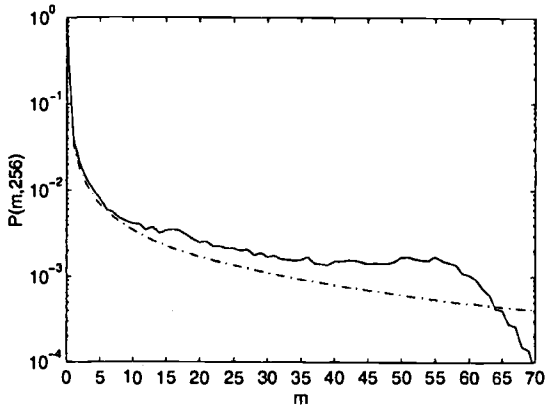


Figure 4: Packet Error Distribution $P(n,m)$ with $n = 256$. Solid line: Measure; Dashed Line: Theory.

to be added to these values to obtain radiated power). The final database contains 32 error streams obtained from straight paths and 33 error streams obtained from circular paths.

4 PED Evaluation

The database has been pre-processed to transform the bit error sequences in gap sequences, according to definitions in [8].

The first quantity evaluated from experimental data is Packet Error Distribution (PED). PED $P(n,m)$ is defined as the probability that a data block, made by n bits, contains n errors and is connected to Packet Error Rate PER by the equation:

$$\mathrm{PER} = \sum_{m=t+1}^{n} P(n,m) = 1 - \sum_{m=0}^{t} P(n,m), \quad (1)$$

being t the number of errors in the block, that can be corrected.

Signal fading rate is very low: $f_D T_b = 8.7 \cdot 10^{-7}$, for circular paths, and $f_D T_b = 2.2 \cdot 10^{-6}$, for linear paths, where $T_b = 868$ ns is bit duration, $f_D = \frac{v}{\lambda}$ is channel Doppler spread, v is mobile speed and $\lambda = 15.7$ cm is the wavelength. However, errors are caused mainly by interference, originating from few base stations, transmitting signaling packets (96 bit long) in every slot ($T_I = 417\mu$s). To represent the instant signal-to-interference ratio (SIR), it seems reasonable to use the Finite State Markov Channel model (FSMC), proposed in [5], with fading rate $f_D T_I$. This means assuming that interference has approximately the same effect of fading with $f'_D = f_D \frac{T_I}{T_b}$. The model considers Rayleigh fading, quantized on L levels, by means of $L-1$ thresholds (referred to SIR [Section 3]) $\{A_k\}_{k=1}^{L-1}$. L fading states $\{S_k\}_{k=0}^{L-1}$ are obtained, so that $\{S_k : \mathrm{SIR} \in [A_k\ A_{k+1})\}_{k=0}^{L-1}$, where $A_0 = 0$ and $A_L = +\infty$.

The FSMC model is completely characterized by the transition matrix $\mathbf{T} = \{t_{j,k}\}_{j,k=0}^{L-1}$ and by the crossover probabilities $\mathbf{e} = \{e_k\}_{k=0}^{L-1}$ (where e_k is the average error probability over the Binary Symmetric Channel, corresponding to the SIR in the state S_k). The transition probabilities are given by:

$$t_{k,k+1} \approx \frac{N_{k+1}}{R_t^{(k)}}, \quad k = 0, \ldots, L-2, \quad (2a)$$

$$t_{k,k-1} \approx \frac{N_k}{R_t^{(k)}}, \quad k = 1, \ldots, L-1, \quad (2b)$$

$$t_{i,j} \approx 0, \quad \forall i,j : |i-j| > 1, \quad (2c)$$

$$t_{k,k} = 1 - \sum_{l \neq k} t_{k,l}, \quad (2d)$$

where $R_t^{(k)} = \frac{p_k}{T_b}$, $N_k = \sqrt{\frac{2\pi A_k}{\varrho}} f'_D \exp\left(-\frac{A_k}{\varrho}\right)$, ϱ is the average SIR, $p_k = \exp\left(-\frac{A_k}{\varrho}\right) - \exp\left(-\frac{A_{k+1}}{\varrho}\right), k = 0, \ldots, L-1$ are the steady state probabilities for each state S_k. The crossover probabilities for an incoherently demodulated 2-FSK, well approximating GMSK, are given by:

$$e_k = \frac{1}{p_k(2+\varrho)} \left\{ \exp\left[-A_k\left(\frac{1}{\varrho} + \frac{1}{2}\right)\right] - \exp\left[-A_{k+1}\left(\frac{1}{\varrho} + \frac{1}{2}\right)\right] \right\}, \quad (3)$$
$$k = 0, \ldots, L-1.$$

PED has been evaluated from this model with the algorithm, described in [8, 3].

Figs. 3 and 4 compare measured PED with PED obtained from FSMC model, for a particular circular path measurement, with -15 dBm power level at

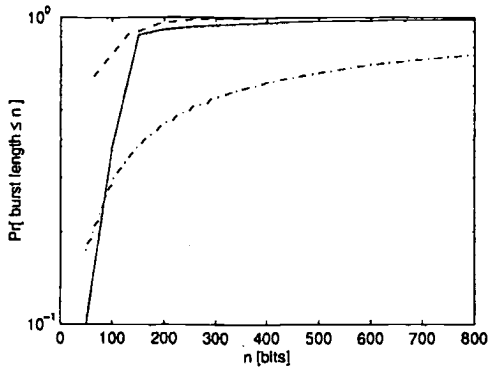


Figure 5: Burst length distribution. ——: Measure; -·-·: First order model at bit level; - - -: First order model at block level.

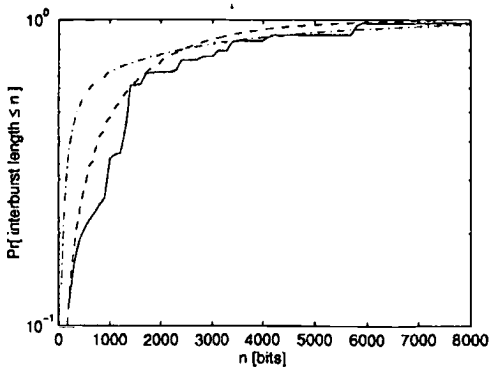


Figure 6: Interburst length distribution. ——: Measure; -·-·: First order model at bit level; - - -: First order model at block level.

modulator (SR1). In the former, packet length n is 64, while in the second it is 256. The mean bit error rate is $P_b = 1.46 \cdot 10^{-2}$, so that an average SIR $\varrho = 18.2$ dB is estimated from $P_b = \frac{1}{2+\varrho}$ [9]. Continuous line represents PED obtained from data, while dashed line represents PED obtained from FSMC model, evaluated using eqns. (2) and (3). The model has 12 quantization levels, using as thresholds $\{A_k = \varrho - 2(11-k)\}_{k=1}^{11}$ dB.

It can be noticed that FSMC model follows well experimental results in Figure 3, while it shows some departure in Figure 4. Furthermore, it has been checked that PED behavior is quite insensitive to the value of $f_D T$ (if $f_D T \lesssim 10^{-3}$). It can be noticed that the effect of interference on PED is well represented by the FSMC model, defined by eqns. (2) and (3), used in presence of fading with AWGN [3]. This means that the effect of interference, also when interferers are temporally deterministic and in small number, can be considered Gaussian. The major departure of theory from measurement in Figure 4 puts on evidence the limits of the approximation.

5 Burst Distribution

The error stream has been also analyzed, considering the burst and interburst length distribution. A *burst* is defined as a group of bit, starting and ending with a wrong bit, such that the maximum separation between any couple of wrong bits is never higher than a fixed number N_G (*guard interval*) [10]. In this work, it is $N_G = 100$. The group of bits between two consecutive bursts is an *interburst*. Figs. 5 and 6 show with solid line the distribution of burst and interburst length, respectively, obtained from the same experimental data, used in Section 4. Inside each figure, the dash-dotted line represents the behavior obtained from the FSMC, therein described. The burst and interburst length distributions are approximated by the residence time distribution in the state set S_1–S_9 and S_{10}–S_{12}, considered as error and no-error states at bit level, respectively. The evaluation method is described in [11].

The dashed line represents the results obtained from a first order Markov model at block level, discussed in the following. Let $n = 64$ be the block length. Let G and B be the events of correct block (without errors) and of wrong block (containing wrong bits), respectively. Let a two-state FSMC model at the bit level be obtained, quantizing SIR with respect to a threshold F, to represent the process at bit level. A typical value, also used here, is $F = \varrho - 4$ (dB). Equations (2) give its transition matrix:

$$\mathbf{T}_{\text{bit}} = \begin{bmatrix} p_{bb} & p_{bg} \\ p_{gb} & p_{gg} \end{bmatrix},$$

and its stationary distribution $\mathbf{p}_{\text{bit}} = [p_b \; p_g]$, where g and b are the no-error and error states at the bit level, respectively.

A first order Markov chain describing the G-B process can be built from $\mathbf{T}_{\text{bit}}, \mathbf{p}_{\text{bit}}$ and its transition matrix is given by:

$$\mathbf{T}_{\text{block}} \simeq \begin{bmatrix} P_{BB} = 1 - P_{BG} & P_{BG} = p_g\,(p_{gg})^{n-1} \\ P_{GB} = 1 - P_{GG} & P_{GG} = (p_{gg})^n \end{bmatrix}$$

where $P_{BG} \simeq P(b_{m+1} = 0, \ldots, b_{m+n} = 0 | b_m = 1) p_b + P(b_{m+1} = 0, \ldots, b_{m+n} = 0 | b_m = 0) p_g$ and $P_{GG} \simeq P(b_{m+1} = 0, \ldots, b_{m+n} = 0 | b_m = 0)$ (given $n \gg 1$).

The burst and interburst length distributions are obtained from this model, as residence time in the B and G state, respectively.

The second model shows a closer resemblance with the experimental results, because the state definition matches better the burst and interburst definitions. Therefore, from the burst/interburst distribution point of view, the channel with interference is well represented even by an on-off Markov model, where the 'on' condition corresponds to SIR being above a suitable threshold F. Moreover, the first model exhibits larger deviation in the burst length case, because the approximation of a burst, as a sequence of totally wrong bits, is an oversimplification.

6 Conclusions and Future Work

In this paper, a measurement system has been described, for the acquisition of bit error streams on a DECT digital channel. A database of error patterns has been obtained in a laboratory and office environment, considering also the effect of some DECT interferers thereby placed. Some processing has been made on it, consisting in evaluation of Packet Error Distribution (PED) and of burst and interburst length distributions. The experimental results for PED match with the ones given by a FSMC model, typically adopted for fading channel with AWGN. This puts on evidence that the interference effect can be assumed Gaussian. Burst/interburst length distributions are correctly modeled even by a simple on-off model. Though, some deviations between models and experimental results have been also observed, due to the deterministic features of interference.

Future work will consist in examining more complex models, that are capable of giving more accurate results and that can be applied to further characterizing features of the channel, as the gap distribution. The aim is to find a class of models, capable to give a unitary description of digital wireless channel behavior, both at the bit level and at the packet level, so that parameters useful to system project are readily obtained from it.

Acknowledgments

The authors wish to thank the TELITAL S.p.A., for having supplied measurement instrumentation and having followed its execution with continuous and qualified technical support.

References

[1] A. Goldsmith and P. Varaiya, "Capacity, Mutual Information and Coding for Finite-State Markov Channels", *IEEE Trans. Inform. Theory*, vol. 42, no. 3, pp. 868 – 886, May 1996.

[2] A. J. Goldsmith, L. J. Greenstein, and G. J. Foschini, "Error Statistics of Real-Time Power Measurements in Cellular Channels with Multipath and Shadowing", *IEEE Trans. Veh. Technol.*, vol. 43, no. 3, pp. 439 – 446, Aug. 1994.

[3] H. Bischl and E. Lutz, "Packet Error Rate in the Non-Interleaved Rayleigh Channel", *IEEE Trans. Commun.*, vol. 43, no. 2/3/4, pp. 1375 – 1382, Feb./Mar./Apr. 1995.

[4] P. M. Crespo, R. Mann Pelz, J. P. Cosmas, and J. García-Frías, "Results of Channel Error Profiles for DECT", *IEEE Trans. Commun.*, vol. 44, no. 8, pp. 913 – 917, Aug. 1996.

[5] H. S. Wang and N. Moayeri, "Finite-State Markov Channel: A Useful Model for Radio Communication Channel", *IEEE Trans. Veh. Technol.*, vol. 44, no. 1, pp. 163 – 171, Feb. 1995.

[6] F. Babich, G. Lombardi, and E. Valentinuzzi, "Indoor Propagation Characteristics in DECT Band", in *Proceedings of IEEE VTC '96*, Atlanta, GA, Apr. 27 – May 2, 1996, pp. 574 – 578.

[7] F. Babich, G. Lombardi, L. Tomasi, and E. Valentinuzzi, "Indoor Propagation Measurements at DECT Frequencies", in *Proc. of IEEE MELECON '96*, Bari, Italy, May 13 – 16, 1996, pp. 1355 – 1359.

[8] L. N. Kanal and A. R. K. Sastry, "Models for Channels with Memory and Their Applications to Error Control", *Proc. IEEE*, vol. 66, no. 7, pp. 724 – 744, July 1978.

[9] J. G. Proakis, . *Digital Communications*, McGraw–Hill, New York, 1989.

[10] E. A. Newcombe and S. Pasupathy, "Error Rate Monitoring for Digital Communications", *Proc. IEEE*, vol. 70, no. 8, Aug. 1982.

[11] Attila Csenki, *Dependability for Systems with a Partitioned State Space*, vol. 90 of *Lecture Notes in Statistics*, Springer-Verlag, 1994.