UNITED STATES DISTRICT COURT
DISTRICT OF MASSACHUSETTS
EASTERN DIVISION

ENTERASYS NETWORKS, INC.,	)	Civil Action No: 05-CV-11298 (DPW)
Plaintiff,	)	
v.	)	
FOUNDRY NETWORKS, INC. AND EXTREME NETWORKS, INC.,	)	
Defendants.	)	

DECLARATION OF MARC N. HENSCHKE IN SUPPORT OF OPENING CLAIM CONSTRUCTION BRIEF OF PLAINTIFF ENTERASYS NETWORKS, INC.

I, MARC N. HENSCHKE, declare as follows:

1. I am an attorney with the law firm of Robins, Kaplan, Miller & Ciresi L.L.P., counsel of record for the named Plaintiff Enterasys Networks, Inc. ("Enterasys") in the above-referenced action. I submit this Declaration in support of Enterasys' Opening Claim Construction Brief filed concurrently herewith under separate cover. I have personal knowledge of the facts stated herein and, if sworn as a witness, I could and would testify competently thereto.

2. Attached hereto as Exhibit 1 is a true and correct copy of U.S. Patent No. 5,251,205 (the "'205 Patent") entitled *Multiple Protocol Routing*.

3. Attached hereto as Exhibit 2 is a true and correct copy of U.S. Patent No. 5,390,173 (the "'173 Patent") entitled *Packet Format In Hub For Packet Data Communications System*.

4. Attached hereto as Exhibit 3 is a true and correct copy of U.S. Patent No. 6,128,665 (the "'665 Patent") entitled *System For Broadcasting Messages To Each Of Default VLAN Ports In Subset Of Ports Defined As VLAN Ports.*

5. Attached hereto as Exhibit 4 is a true and correct copy of U.S. Patent No. 6,147,995 (the "'995 Patent") entitled *Method For Establishing Restricted Broadcast Groups In A Switched Network*.

6. Attached hereto as Exhibit 5 is a true and correct copy of U.S. Patent No. 6,539,022 (the "'022 Patent") entitled *Network Device With Multicast Forwarding Data*.

7. Attached hereto as Exhibit 6 is a true and correct copy of U.S. Patent No. 6,560,236 (the "'236 Patent") entitled *Virtual VLANS*.

8. Attached hereto as Exhibit 7 is a true and correct copy of a printout of a web page from *www.worldwebonline.com* providing a definition for the term "prepend."

9. Attached hereto as Exhibit 8 is a true and correct copy of excerpts from *The American Heritage Dictionary Of The English Language (Fourth Ed. 2000)* providing a definition for the term "prefix."

10. Attached hereto as Exhibit 9 is a true and correct copy of a networking standard promulgated in March of 1986 by the International Organization for Standardization ("ISO") formally known as "Protocol for Providing Connectionless-mode Network Service (ISO 8473)."

11. Attached hereto as Exhibit 10 is a true and correct copy of a networking standard promulgated in September of 1981 by the Internet Engineering Task Force (the "IETF") formally known as "Internet Protocol (RFC 791)."

12. Attached hereto as Exhibit 11 is a true and correct copy of excerpts from *The American Heritage Dictionary Of The English Language (New College Ed. 1976)* providing a definition for the term "insert."

13. Attached hereto as Exhibit 12 is a true and correct copy of the Declaration Of James F. Kurose In Support Of Opening Claim Construction Brief Of Plaintiff Enterasys Networks, Inc. dated August 16, 2007.

14. Attached hereto as Exhibit 13 is a true and correct copy of a printout of a web page from *www.m-w.com/dictionary* providing a definition for the term "e.g."

15. Attached hereto as Exhibit 14 is a true and correct copy of a printout of a web page from *www.m-w.com/dictionary* providing a definition for the term "i.e."

16. Attached hereto as Exhibit 15 is a true and correct copy of excerpts from *Random House Unabridged Dictionary, Second Edition (1993)* providing a definition for the term "any."

I declare under penalty of perjury that the foregoing is true and correct and that this Declaration is executed on this 17th day of August, 2007 at Boston, Massachusetts.

/s/ Marc N. Henschke
MARC N. HENSCHKE

CERTIFICATE OF SERVICE

I, Marc N. Henschke, hereby certify that on August 17, 2007, I caused a true and correct copy of the attached *Declaration of Marc N. Henschke In Support Of Opening Claim Construction Brief Of Plaintiff Enterasys Networks, Inc.* to be served electronically pursuant to the Court's CM/ECF electronic filing system upon Defendants' respective counsel of record as indicated by asterisks below:

Counsel for Defendant Extreme Networks, Inc.:

William L. Anthony, Jr., Esq.*
I. Neel Chatterjee, Esq.*
Sanjeet K. Dutta, Esq.*
Matthew H. Poppe, Esq.*
ORRICK, HERRINGTON & SUTCLIFFE LLP
1000 Marsh Road
Menlo Park, CA 94025

Steven M. Bauer, Esq.*
Jeremy P. Oczek, Esq.*
John W. Pint, Esq.*
PROSKAUER ROSE LLP
One International Place
Boston, MA 02110-2600

Counsel for Defendant Extreme Networks, Inc.:

Terrence P. McMahon, Esq.*
Vera M. Elson, Esq.*
MCDERMOTT, WILL & EMERY LLP
3150 Porter Drive
Palo Alto, CA 94303-1212

Emily E. Smith-Lee, Esq.*
Peter L. Resnik, Esq.*
MCDERMOTT, WILL & EMERY LLP
28 State Street
Boston, MA 02109-1775

Christopher D. Bright, Esq.
MCDERMOTT, WILL & EMERY LLP
18191 Von Karman Avenue, Suite 400
Irvine, CA 92612-7107

Dated: August 17, 2007

/s/ Marc N. Henschke
Marc N. Henschke

Exhibit 1

US005251205A

United States Patent [19]

Callon et al.

[11] Patent Number: 5,251,205

[45] Date of Patent: Oct. 5, 1993

[54] MULTIPLE PROTOCOL ROUTING

[75] Inventors: Ross W. Callon, Bedford; Radia J. Perlman, Acton; Eric C. Rosen, Arlington, all of Mass.; John Harper, Reading, England

[73] Assignee: Digital Equipment Corporation, Maynard, Mass.

[21] Appl. No.: 577,437

[22] Filed: Sep. 4, 1990

[51] Int. Cl.[5] H04J 3/26; H04L 12/56

[52] U.S. Cl. 370/60; 370/94.1; 370/54; 370/79; 370/85.13

[58] Field of Search 370/94.1, 85.13, 85.14, 370/60, 79, 54

[56] References Cited

U.S. PATENT DOCUMENTS

4,706,081	11/1987	Hart et al.	370/61
4,831,620	5/1989	Conway et al.	340/825.05
4,893,307	1/1990	McKay et al.	370/94.1
5,018,133	5/1991	Tsukakoshi et al.	370/85.13
5,031,174	7/1991	Natsume	370/85.14

OTHER PUBLICATIONS

John M. McQuillan et al.; "The New Routing Algorithm for the ARPANET"; *IEEE;* vol. Com-28, No. 5; May 1980.

E. W. Dijkstra; "A Note on Two Problems in Connexion with Graphs"; *Numerische Mathmatik;* pp. 269–271 1959.

"Protocol for Providing the Connectionless-Mode Network Service"; ISO 8473; Mar. 1987.

"Intermediate system to Intermediate system Intra-Domain routeing exchange protocol for use in Conjunction with the Protocol for providing the Connectionless-mode Network Service"; *ISO* 10589; Oct. 15, 1989.

"Information processing systems-Telecommunications and information exchange between systems-End system to Intermediate system routeing exchange protocol for use in conjunction . . . ;" *ISO* 9542; Mar. 26, 1988.

"Internet Protocol," Darpa Internet Program, Protocol Specification; RFC 791; Sep. 1981.

J. Postel; "Internet Control Message Protocol," Darpa Internet, Program Protocol Specification; RFC 792; Sep. 1981.

R. Braden, J. Postel; "Requirements for Internet Gateways," RFC 1009; Jun. 1987.

J. Moy; "The OSPF Specification," RFC 1131; Oct. 1989.

C. L. Hedrick, L. Bosack; "An Introduction to IGRP," Jul. 26, 1989.

Callon et al., "Routing in an Internetwork Environment," pp. 4–9.

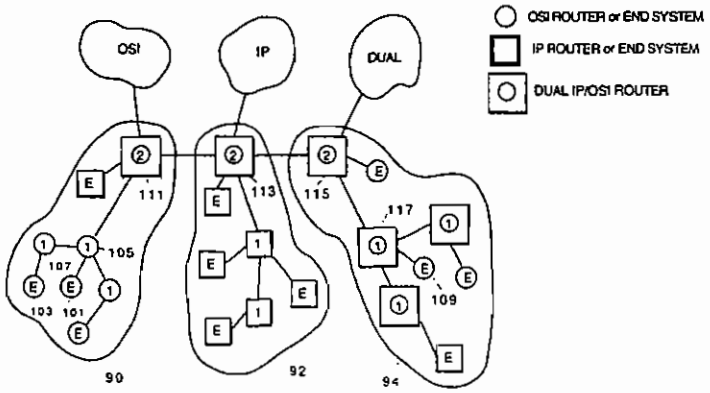
Shoch et al., "Mutual Encapsulation of Internetwork Protocols," IEN 140, Apr. 1980.

Primary Examiner—Douglas W. Olms
Assistant Examiner—Min Jung
Attorney, Agent, or Firm—Fish & Richardson

[57] ABSTRACT

A method for connecting a network so that TCP/IP and OSI 8473 packets may be routed in the same domain. The independence of the addresses is maintained: one device in the network may be assigned only a TCP/IP address, and another device may be assigned only a ISO 8473 address. Furthermore, all of the routers share link state information by using a common link state packet format (such as the ISO 10589 format); thus routes through the network may be computed without regard for the protocols supported by the routers along the route. Where necessary, packets are encapsulated and forwarded through routers which are not capable in the protocol of the packet. In some disclosed embodiments, all of the routers in a given area support a given protocol (or, in fact, have identical capabilities, in which case encapsulation is not required). In these embodiments, the encapsulation is performed by suitable modifications to each router's packet forwarding procedures. In other disclosed embodiments, these topological restrictions are removed, and the network is expanded to support additional protocols. In these embodiments, the Dijkstra algorithm is also modified to generate information on how to encapsulate and forward packets through the network.

24 Claims, 20 Drawing Sheets



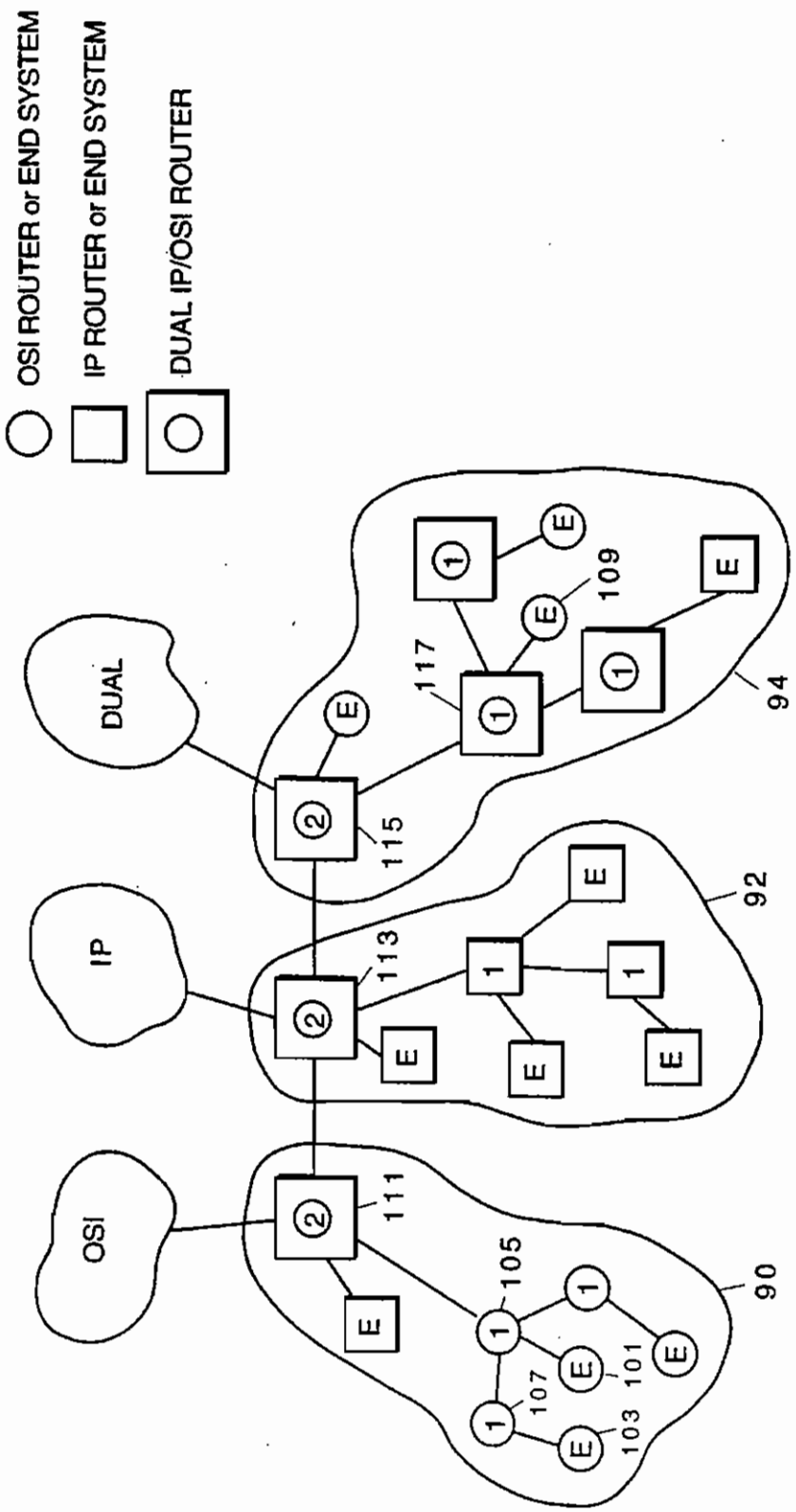

OSI ROUTER or END SYSTEM
IP ROUTER or END SYSTEM
DUAL IP/OSI ROUTER
DUAL
IP
OSI
E
1
2
90
92
94
101
103
105
107
109
111
113
115
117

FIG. 1A

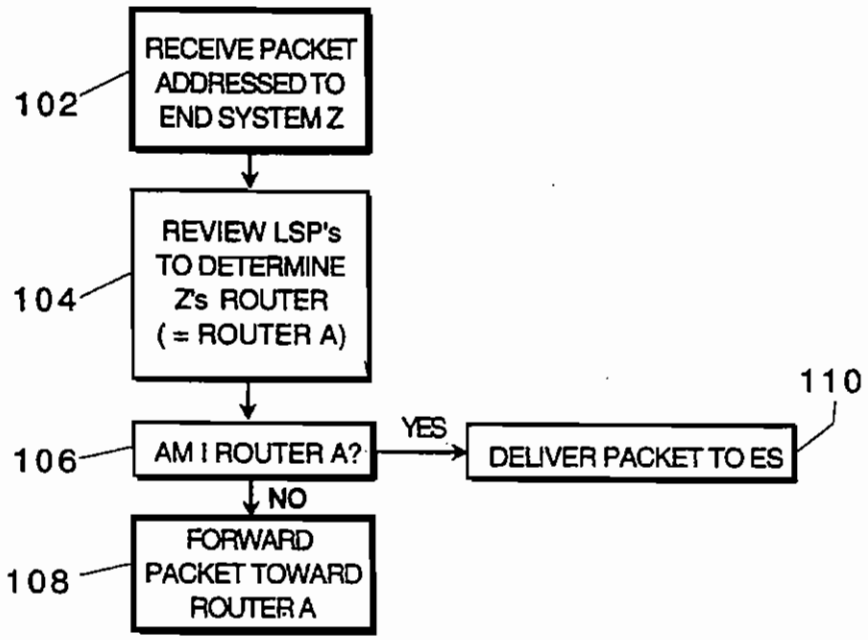

RECEIVE PACKET ADDRESSED TO END SYSTEM Z
102
REVIEW LSP's TO DETERMINE Z's ROUTER (= ROUTER A)
104
AM I ROUTER A?
106
YES
DELIVER PACKET TO ES
110
NO
FORWARD PACKET TOWARD ROUTER A
108

FIG. 1B

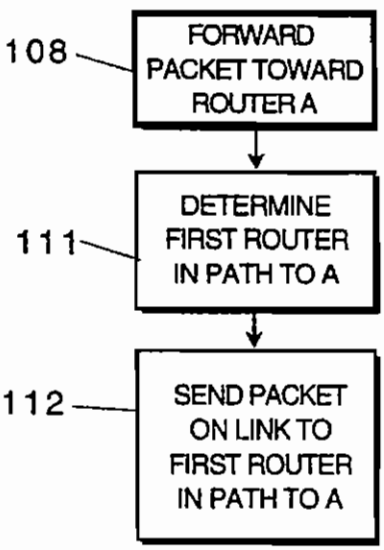

108
FORWARD PACKET TOWARD ROUTER A
DETERMINE FIRST ROUTER IN PATH TO A
111
SEND PACKET ON LINK TO FIRST ROUTER IN PATH TO A
112

FIG. 1C

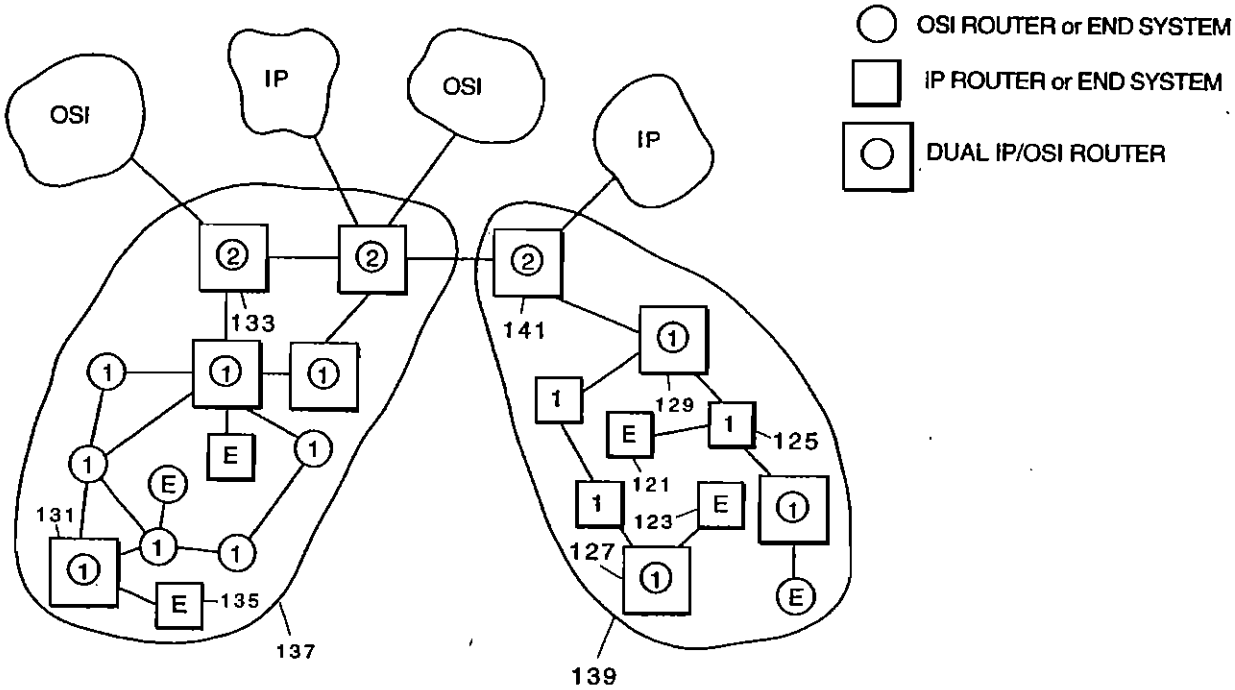

OSI ROUTER or END SYSTEM
IP ROUTER or END SYSTEM
DUAL IP/OSI ROUTER
OSI
IP
OSI
IP
133
141
129
125
121
123
127
131
135
137
139

FIG. 2A

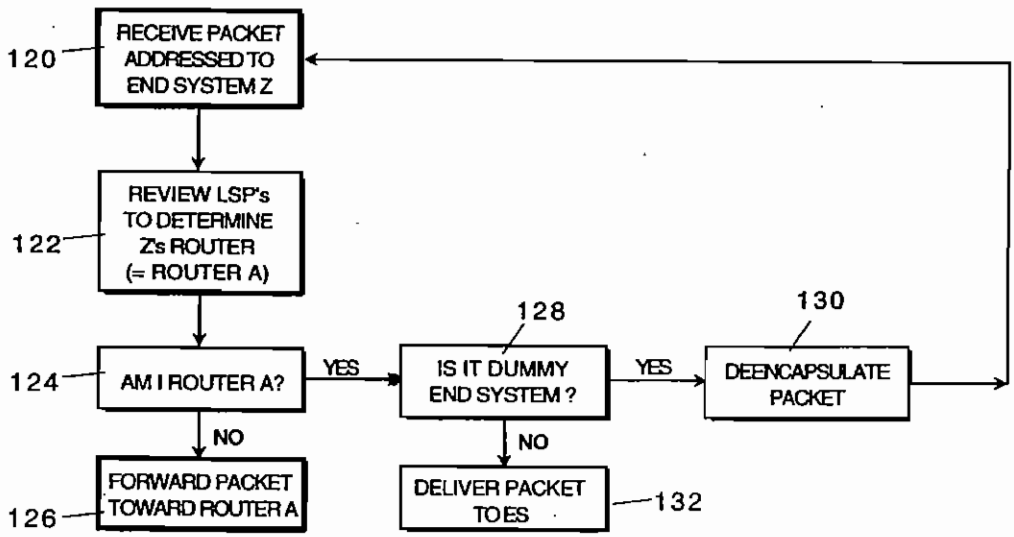

120
RECEIVE PACKET ADDRESSED TO END SYSTEM Z
122
REVIEW LSP's TO DETERMINE Zs ROUTER (= ROUTER A)
124
AM I ROUTER A?
YES
128
IS IT DUMMY END SYSTEM ?
YES
130
DEENCAPSULATE PACKET
NO
126
FORWARD PACKET TOWARD ROUTER A
NO
DELIVER PACKET TO ES
132

FIG. 2B

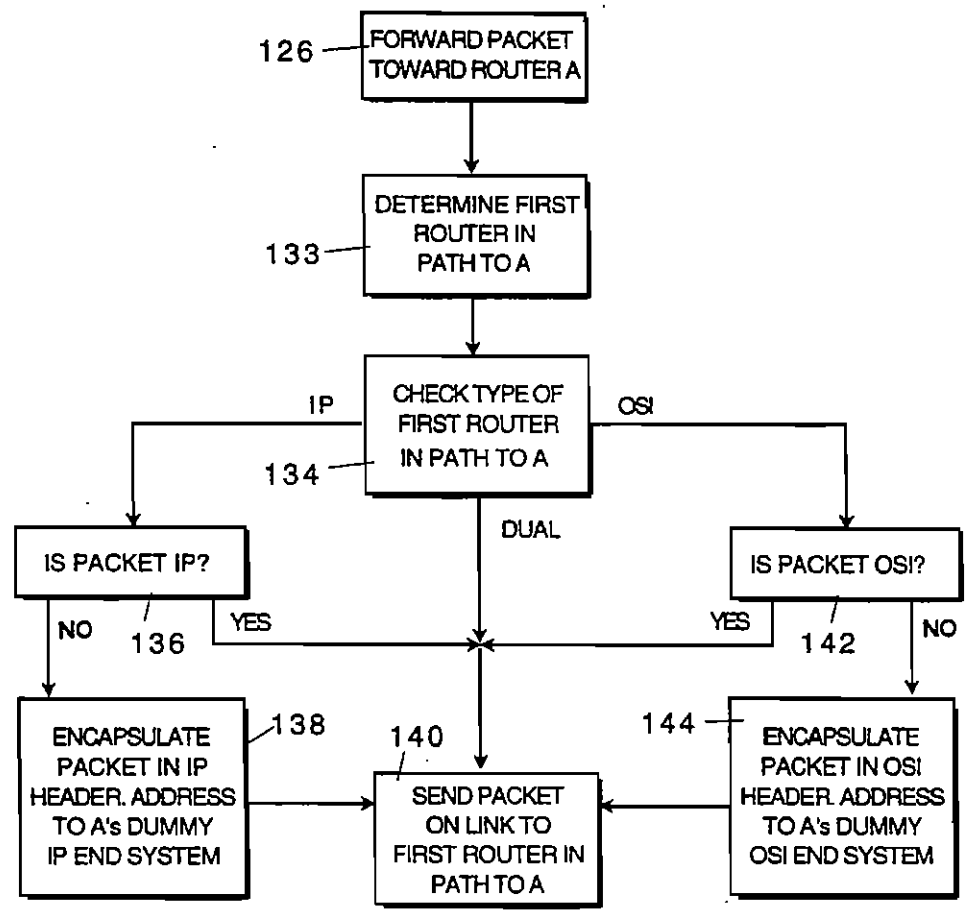

126
FORWARD PACKET TOWARD ROUTER A
DETERMINE FIRST ROUTER IN PATH TO A
133
CHECK TYPE OF FIRST ROUTER IN PATH TO A
134
IP
OSI
DUAL
IS PACKET IP?
IS PACKET OSI?
NO
136
YES
YES
142
NO
ENCAPSULATE PACKET IN IP HEADER. ADDRESS TO A's DUMMY IP END SYSTEM
138
140
144
SEND PACKET ON LINK TO FIRST ROUTER IN PATH TO A
ENCAPSULATE PACKET IN OSI HEADER. ADDRESS TO A's DUMMY OSI END SYSTEM

FIG. 2C

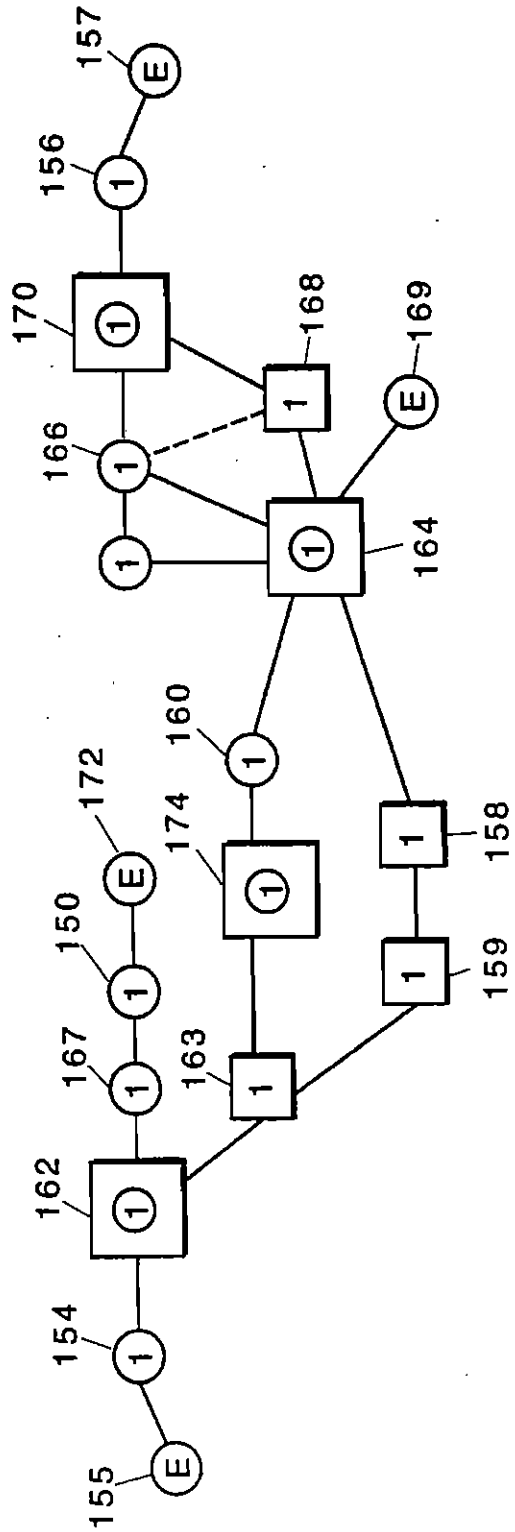

157
E
156
1
170
1
168
1
169
E
166
1
1
1
164
160
1
172
E
174
1
158
1
150
1
159
1
167
1
163
1
162
1
154
1
155
E

FIG. 3

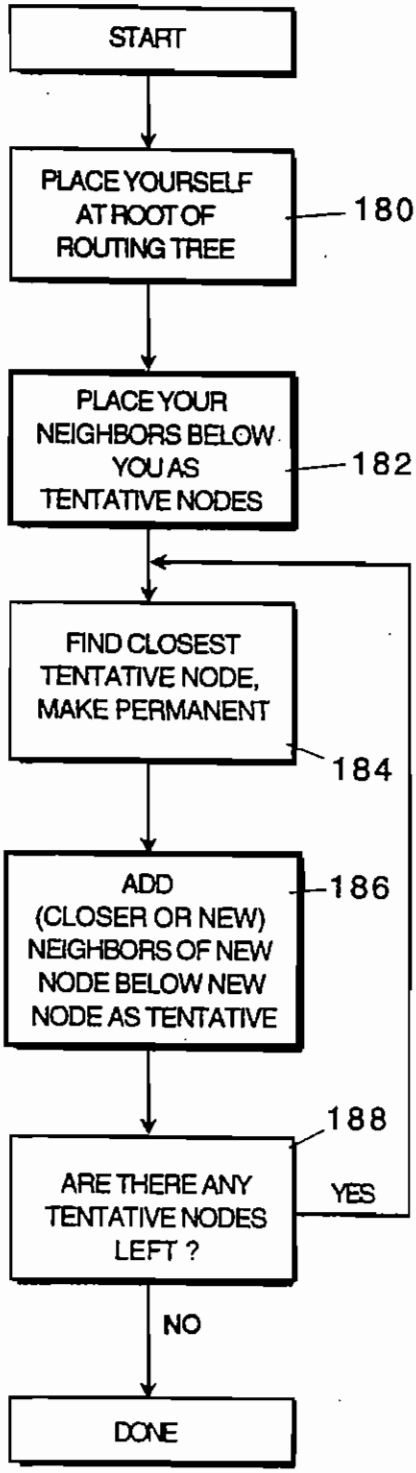

START
PLACE YOURSELF AT ROOT OF ROUTING TREE
180
PLACE YOUR NEIGHBORS BELOW YOU AS TENTATIVE NODES
182
FIND CLOSEST TENTATIVE NODE, MAKE PERMANENT
184
ADD (CLOSER OR NEW) NEIGHBORS OF NEW NODE BELOW NEW NODE AS TENTATIVE
186
188
ARE THERE ANY TENTATIVE NODES LEFT ?
YES
NO
DONE

FIG. 4A

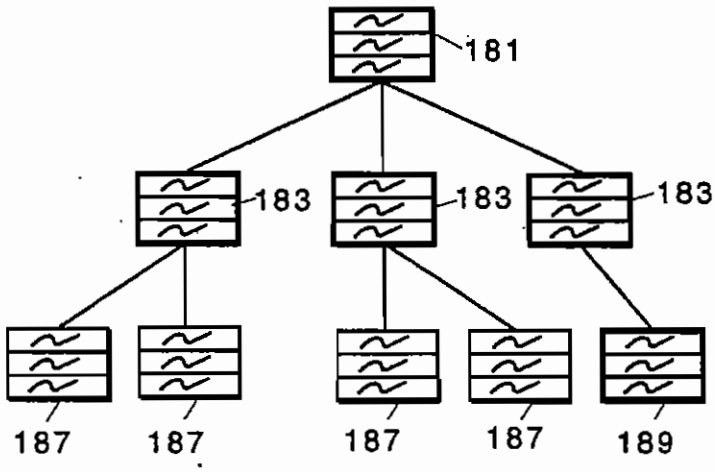

181
183
183
183
187
187
187
187
189

FIG. 4B

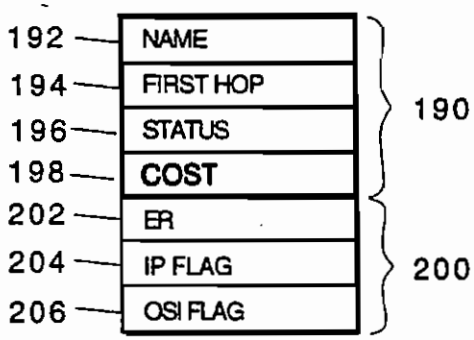

192
194
196
198
202
204
206
NAME
FIRST HOP
STATUS
COST
ER
IP FLAG
OSI FLAG
190
200

FIG. 5A

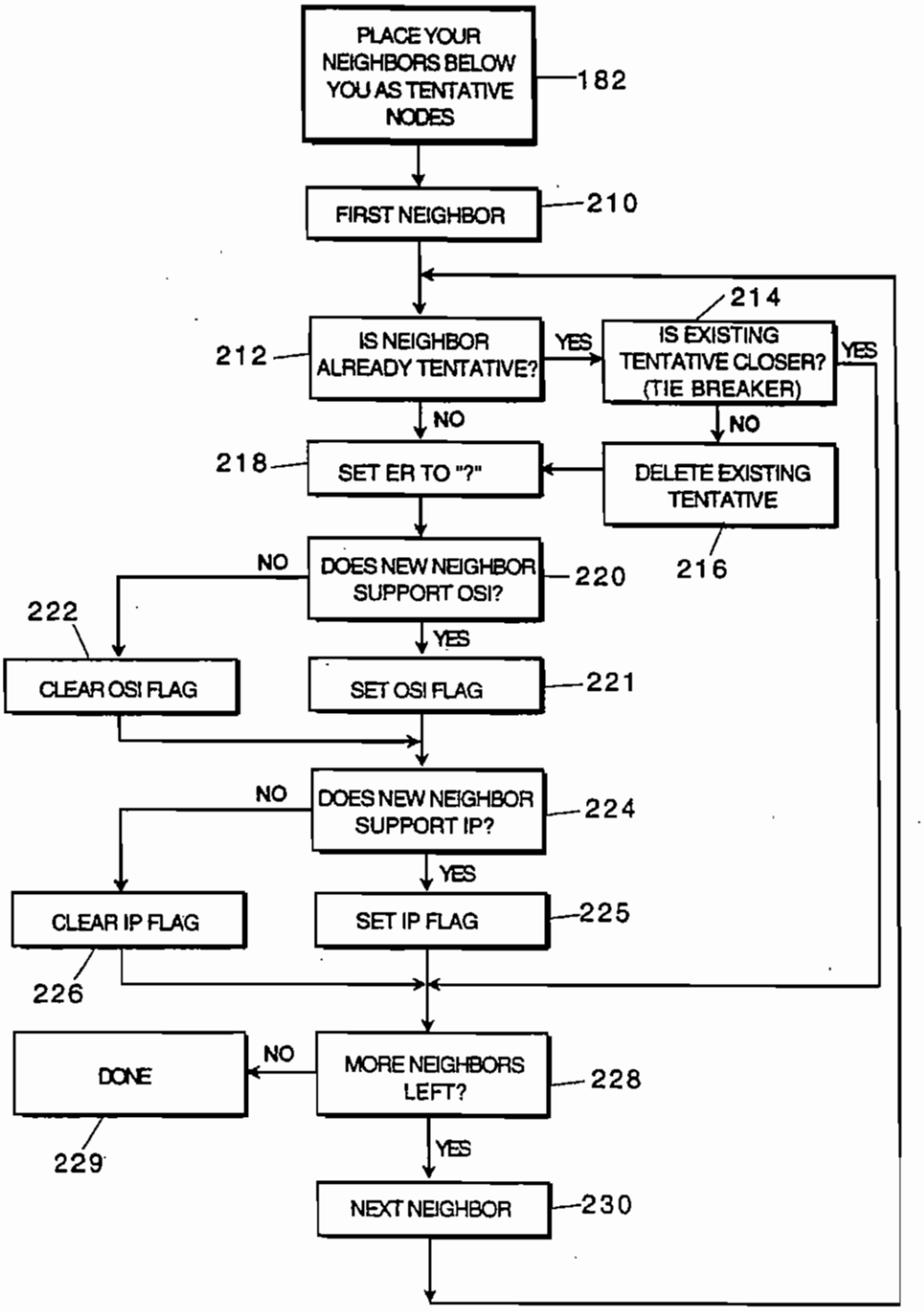

PLACE YOUR NEIGHBORS BELOW YOU AS TENTATIVE NODES
182
FIRST NEIGHBOR
210
214
212
IS NEIGHBOR ALREADY TENTATIVE?
YES
IS EXISTING TENTATIVE CLOSER? (TIE BREAKER)
YES
NO
NO
218
SET ER TO "?"
DELETE EXISTING TENTATIVE
216
NO
DOES NEW NEIGHBOR SUPPORT OSI?
220
222
YES
CLEAR OSI FLAG
SET OSI FLAG
221
NO
DOES NEW NEIGHBOR SUPPORT IP?
224
YES
CLEAR IP FLAG
SET IP FLAG
225
226
DONE
NO
MORE NEIGHBORS LEFT?
228
YES
229
NEXT NEIGHBOR
230

FIG. 5B

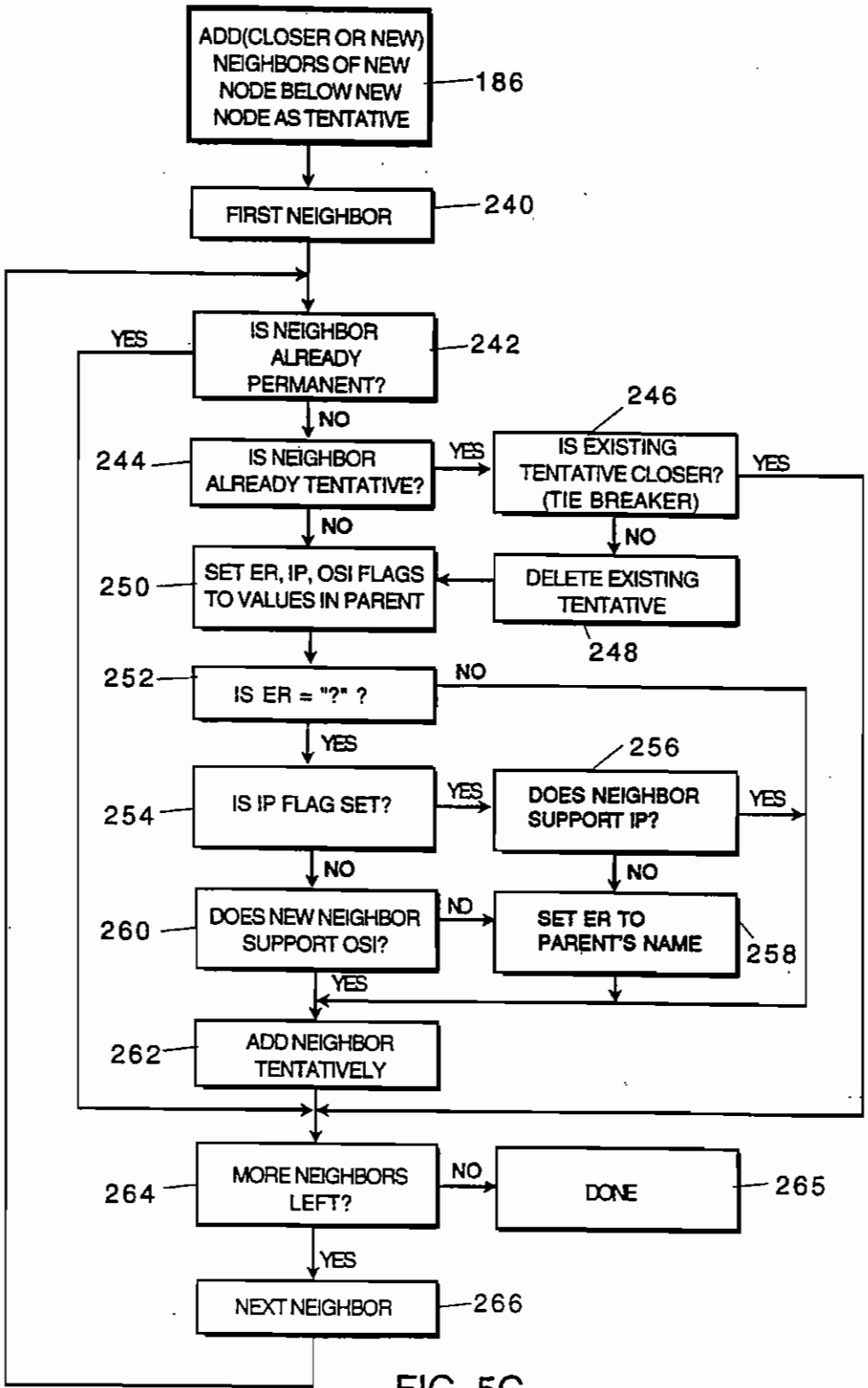

ADD(CLOSER OR NEW) NEIGHBORS OF NEW NODE BELOW NEW NODE AS TENTATIVE
186
FIRST NEIGHBOR
240
IS NEIGHBOR ALREADY PERMANENT?
242
YES
NO
IS NEIGHBOR ALREADY TENTATIVE?
244
YES
NO
IS EXISTING TENTATIVE CLOSER? (TIE BREAKER)
246
YES
NO
DELETE EXISTING TENTATIVE
248
SET ER, IP, OSI FLAGS TO VALUES IN PARENT
250
IS ER = "?" ?
252
NO
YES
IS IP FLAG SET?
254
YES
NO
DOES NEIGHBOR SUPPORT IP?
256
YES
NO
SET ER TO PARENT'S NAME
258
DOES NEW NEIGHBOR SUPPORT OSI?
260
NO
YES
ADD NEIGHBOR TENTATIVELY
262
MORE NEIGHBORS LEFT?
264
NO
DONE
265
YES
NEXT NEIGHBOR
266

FIG. 5C

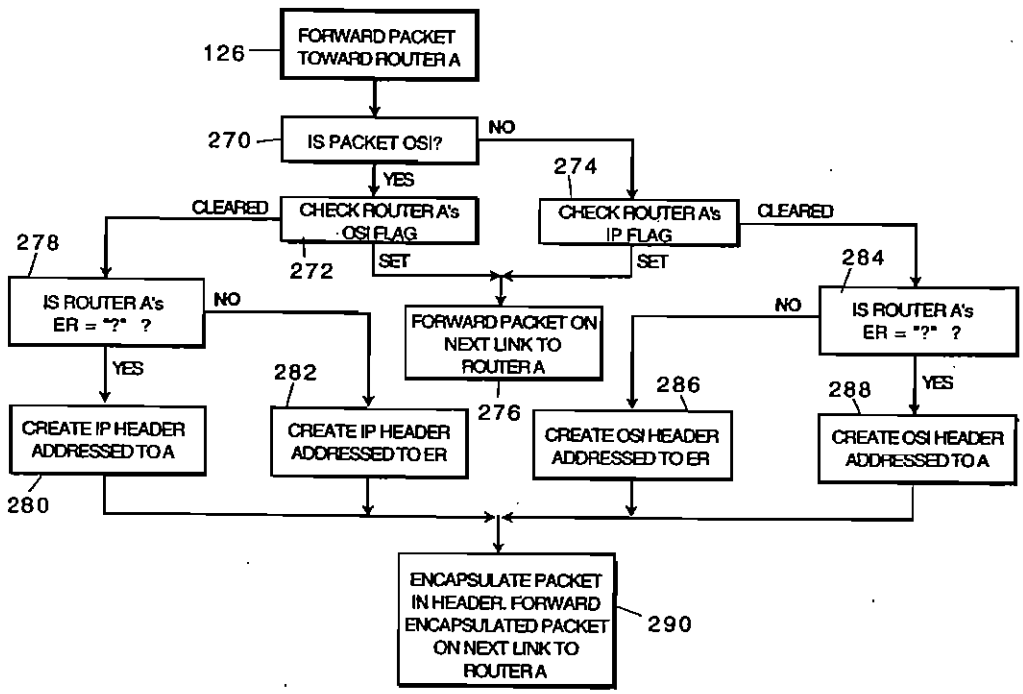

126
FORWARD PACKET TOWARD ROUTER A
270
IS PACKET OSI?
NO
YES
274
CLEARED
CHECK ROUTER A's OSI FLAG
CHECK ROUTER A's IP FLAG
CLEARED
278
272
SET
SET
284
IS ROUTER A's ER = "?" ?
NO
FORWARD PACKET ON NEXT LINK TO ROUTER A
NO
IS ROUTER A's ER = "?" ?
YES
282
286
288
YES
CREATE IP HEADER ADDRESSED TO A
CREATE IP HEADER ADDRESSED TO ER
276
CREATE OSI HEADER ADDRESSED TO ER
CREATE OSI HEADER ADDRESSED TO A
280
ENCAPSULATE PACKET IN HEADER. FORWARD ENCAPSULATED PACKET ON NEXT LINK TO ROUTER A
290

FIG. 6

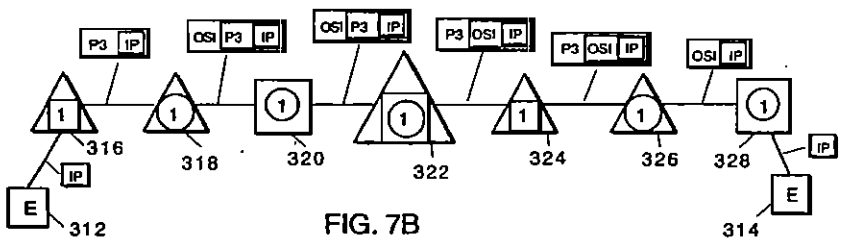
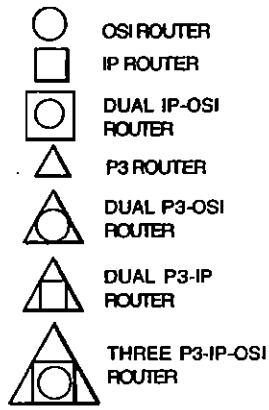


FIG. 7B



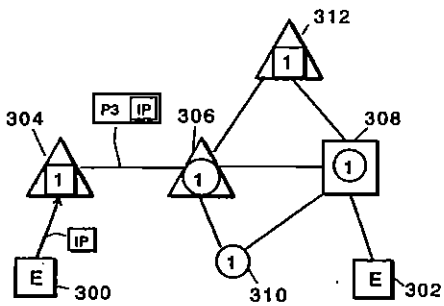


FIG. 7A

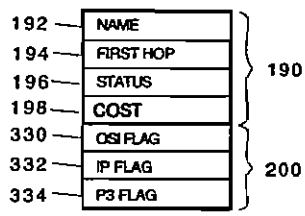


FIG. 8A

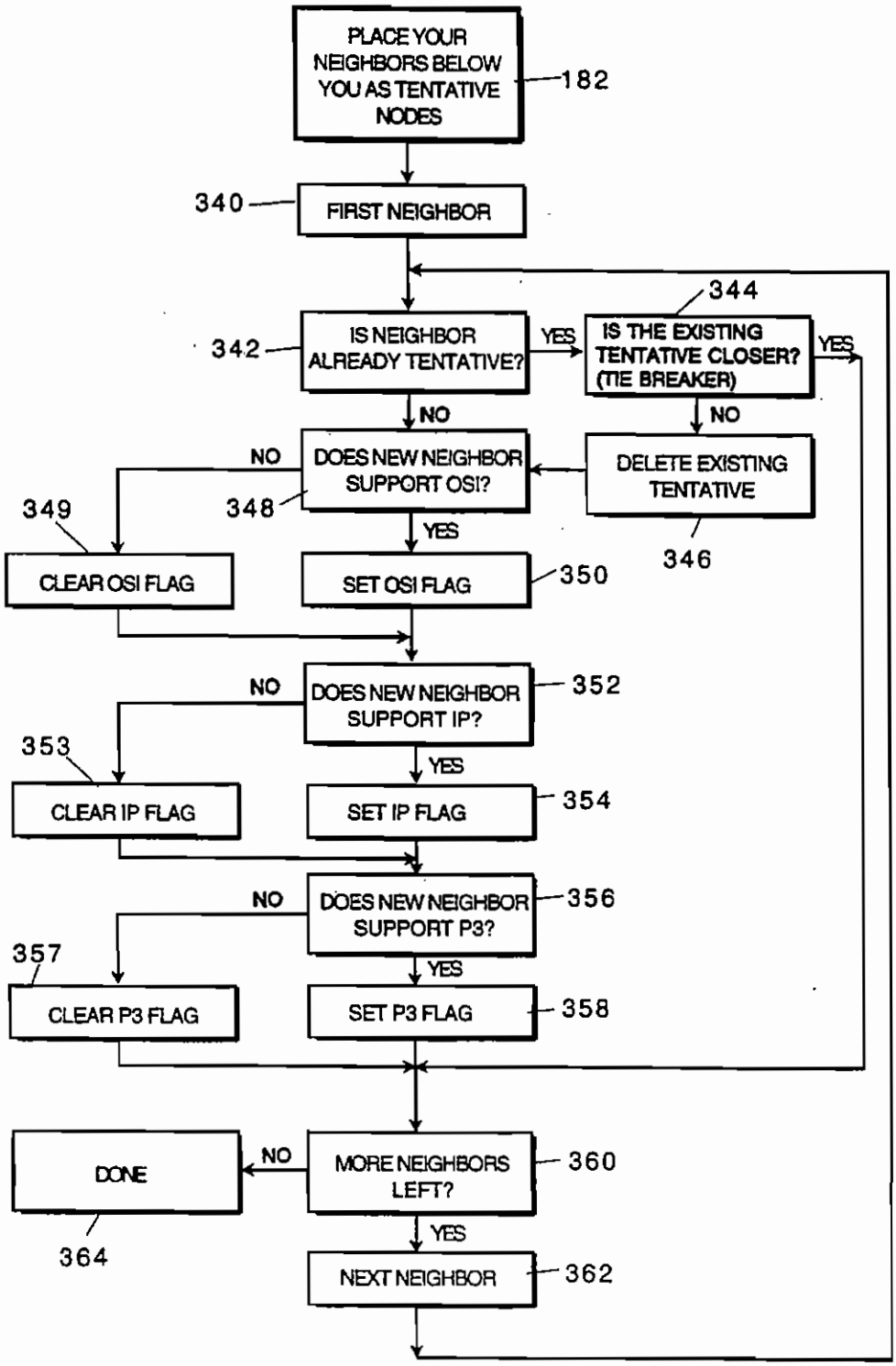

PLACE YOUR NEIGHBORS BELOW YOU AS TENTATIVE NODES
182
340
FIRST NEIGHBOR
342
IS NEIGHBOR ALREADY TENTATIVE?
YES
344
IS THE EXISTING TENTATIVE CLOSER? (TIE BREAKER)
YES
NO
NO
DOES NEW NEIGHBOR SUPPORT OSI?
348
DELETE EXISTING TENTATIVE
346
349
NO
YES
CLEAR OSI FLAG
SET OSI FLAG
350
NO
DOES NEW NEIGHBOR SUPPORT IP?
352
353
YES
CLEAR IP FLAG
SET IP FLAG
354
NO
DOES NEW NEIGHBOR SUPPORT P3?
356
357
YES
CLEAR P3 FLAG
SET P3 FLAG
358
DONE
NO
MORE NEIGHBORS LEFT?
360
364
YES
NEXT NEIGHBOR
362

FIG. 8B

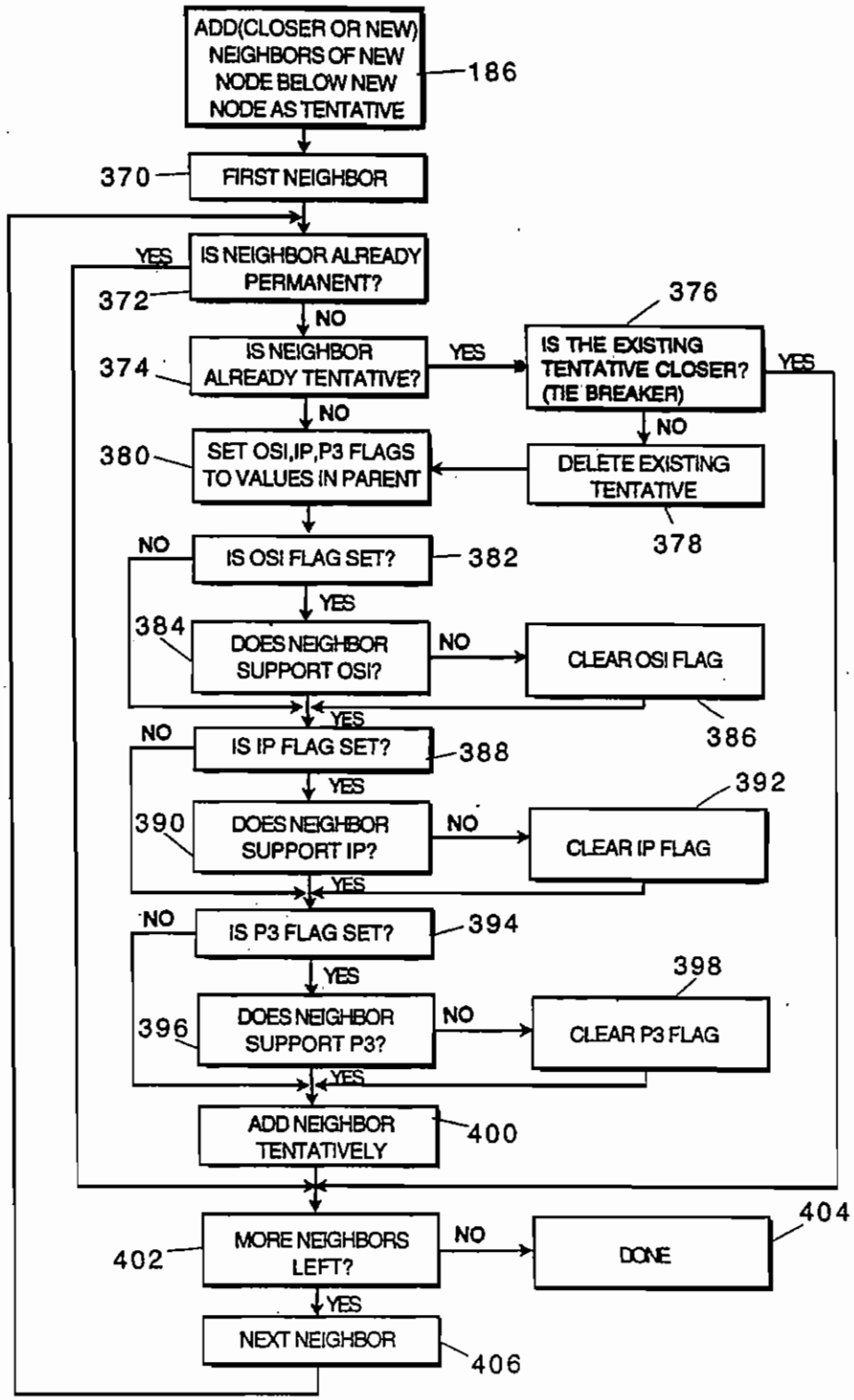

ADD(CLOSER OR NEW) NEIGHBORS OF NEW NODE BELOW NEW NODE AS TENTATIVE
186
370
FIRST NEIGHBOR
YES
IS NEIGHBOR ALREADY PERMANENT?
372
NO
374
IS NEIGHBOR ALREADY TENTATIVE?
YES
376
IS THE EXISTING TENTATIVE CLOSER? (TIE BREAKER)
YES
NO
DELETE EXISTING TENTATIVE
378
NO
380
SET OSI,IP,P3 FLAGS TO VALUES IN PARENT
NO
IS OSI FLAG SET?
382
YES
384
DOES NEIGHBOR SUPPORT OSI?
NO
CLEAR OSI FLAG
386
YES
NO
IS IP FLAG SET?
388
YES
390
DOES NEIGHBOR SUPPORT IP?
NO
392
CLEAR IP FLAG
YES
NO
IS P3 FLAG SET?
394
YES
396
DOES NEIGHBOR SUPPORT P3?
NO
398
CLEAR P3 FLAG
YES
ADD NEIGHBOR TENTATIVELY
400
402
MORE NEIGHBORS LEFT?
NO
404
DONE
YES
NEXT NEIGHBOR
406

FIG. 8C

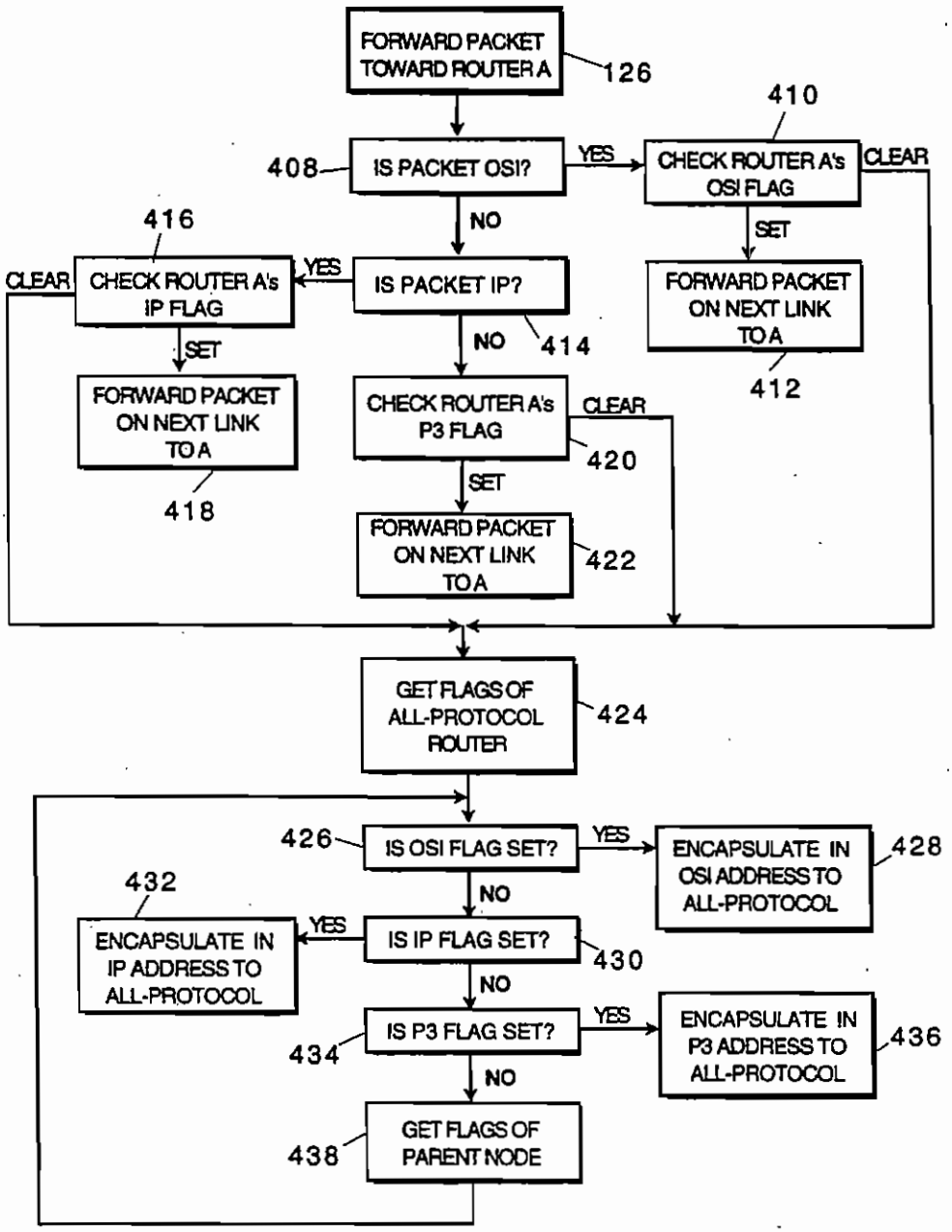

FORWARD PACKET TOWARD ROUTER A
126
410
408
IS PACKET OSI?
YES
CHECK ROUTER A's OSI FLAG
CLEAR
NO
SET
416
CLEAR
CHECK ROUTER A's IP FLAG
YES
IS PACKET IP?
FORWARD PACKET ON NEXT LINK TO A
NO
414
412
SET
FORWARD PACKET ON NEXT LINK TO A
CHECK ROUTER A's P3 FLAG
CLEAR
420
418
SET
FORWARD PACKET ON NEXT LINK TO A
422
GET FLAGS OF ALL-PROTOCOL ROUTER
424
426
IS OSI FLAG SET?
YES
ENCAPSULATE IN OSI ADDRESS TO ALL-PROTOCOL
428
NO
432
ENCAPSULATE IN IP ADDRESS TO ALL-PROTOCOL
YES
IS IP FLAG SET?
430
NO
IS P3 FLAG SET?
YES
ENCAPSULATE IN P3 ADDRESS TO ALL-PROTOCOL
434
436
NO
GET FLAGS OF PARENT NODE
438

FIG. 9

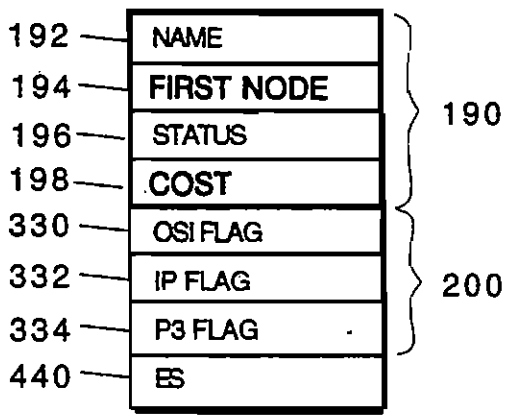

192
NAME
194
FIRST NODE
196
STATUS
198
COST
330
OSI FLAG
332
IP FLAG
334
P3 FLAG
440
ES
190
200

FIG. 10A

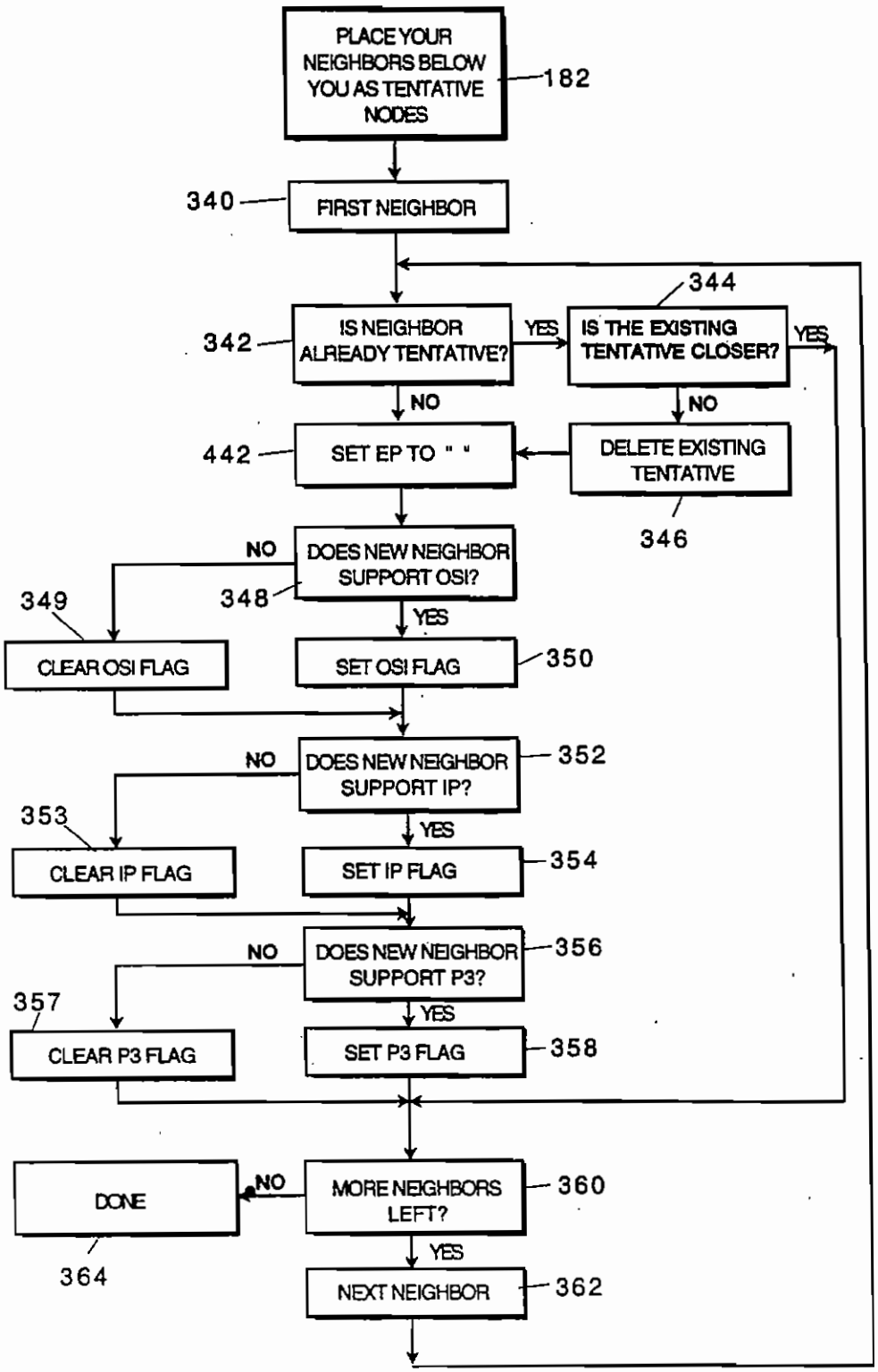

PLACE YOUR NEIGHBORS BELOW YOU AS TENTATIVE NODES
182
340
FIRST NEIGHBOR
342
IS NEIGHBOR ALREADY TENTATIVE?
YES
344
IS THE EXISTING TENTATIVE CLOSER?
YES
NO
NO
442
SET EP TO " "
DELETE EXISTING TENTATIVE
346
NO
DOES NEW NEIGHBOR SUPPORT OSI?
348
349
YES
CLEAR OSI FLAG
SET OSI FLAG
350
NO
DOES NEW NEIGHBOR SUPPORT IP?
352
353
YES
CLEAR IP FLAG
SET IP FLAG
354
NO
DOES NEW NEIGHBOR SUPPORT P3?
356
357
YES
CLEAR P3 FLAG
SET P3 FLAG
358
DONE
NO
MORE NEIGHBORS LEFT?
360
YES
364
NEXT NEIGHBOR
362

FIG. 10B

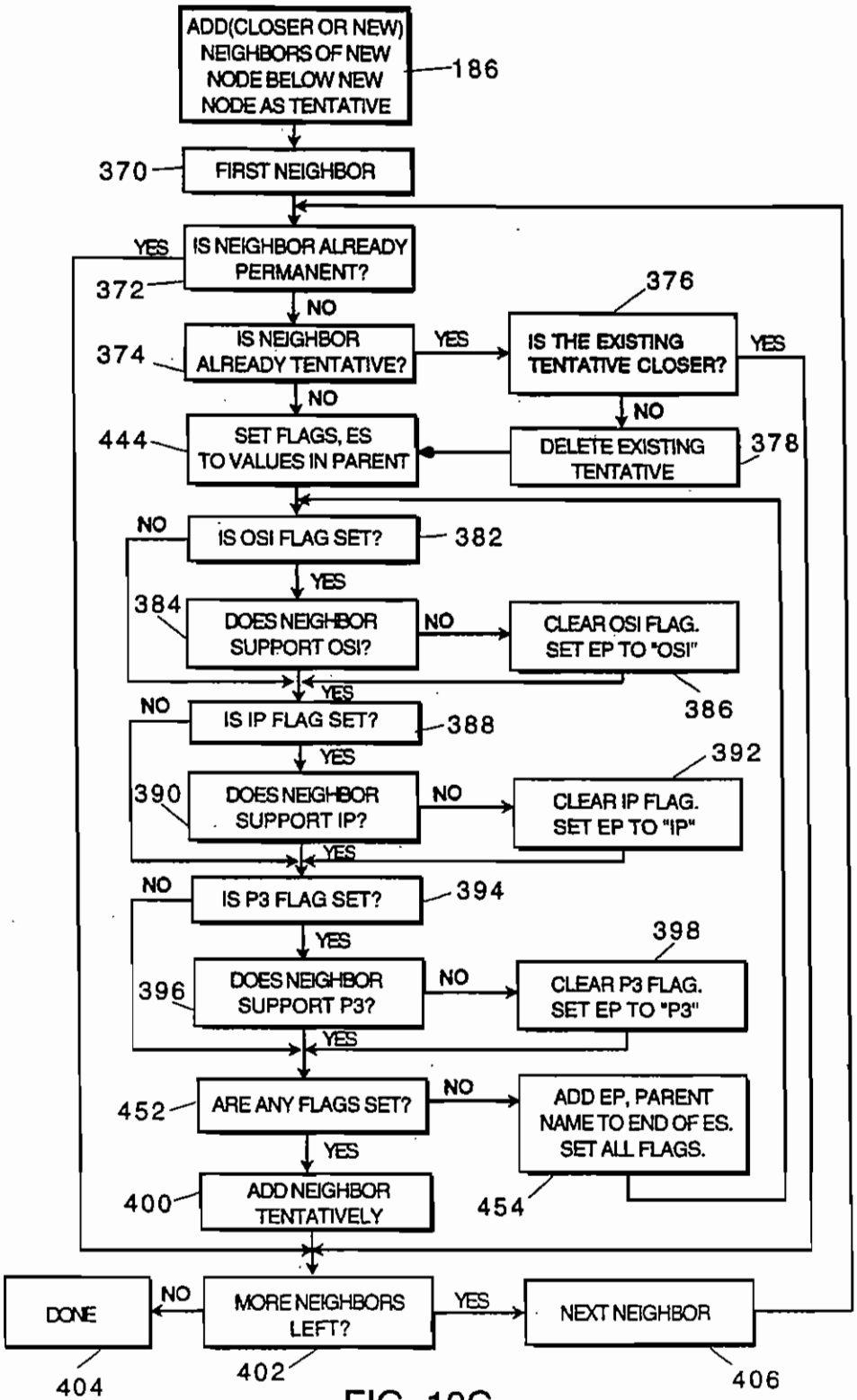

ADD(CLOSER OR NEW) NEIGHBORS OF NEW NODE BELOW NEW NODE AS TENTATIVE
186
370
FIRST NEIGHBOR
YES
IS NEIGHBOR ALREADY PERMANENT?
372
NO
374
IS NEIGHBOR ALREADY TENTATIVE?
YES
376
IS THE EXISTING TENTATIVE CLOSER?
YES
NO
444
SET FLAGS, ES TO VALUES IN PARENT
DELETE EXISTING TENTATIVE
378
NO
IS OSI FLAG SET?
382
YES
384
DOES NEIGHBOR SUPPORT OSI?
NO
CLEAR OSI FLAG. SET EP TO "OSI"
386
YES
NO
IS IP FLAG SET?
388
YES
390
DOES NEIGHBOR SUPPORT IP?
NO
392
CLEAR IP FLAG. SET EP TO "IP"
YES
NO
IS P3 FLAG SET?
394
YES
396
DOES NEIGHBOR SUPPORT P3?
NO
398
CLEAR P3 FLAG. SET EP TO "P3"
YES
452
ARE ANY FLAGS SET?
NO
ADD EP, PARENT NAME TO END OF ES. SET ALL FLAGS.
454
YES
400
ADD NEIGHBOR TENTATIVELY
DONE
NO
MORE NEIGHBORS LEFT?
YES
NEXT NEIGHBOR
404
402
406

FIG. 10C

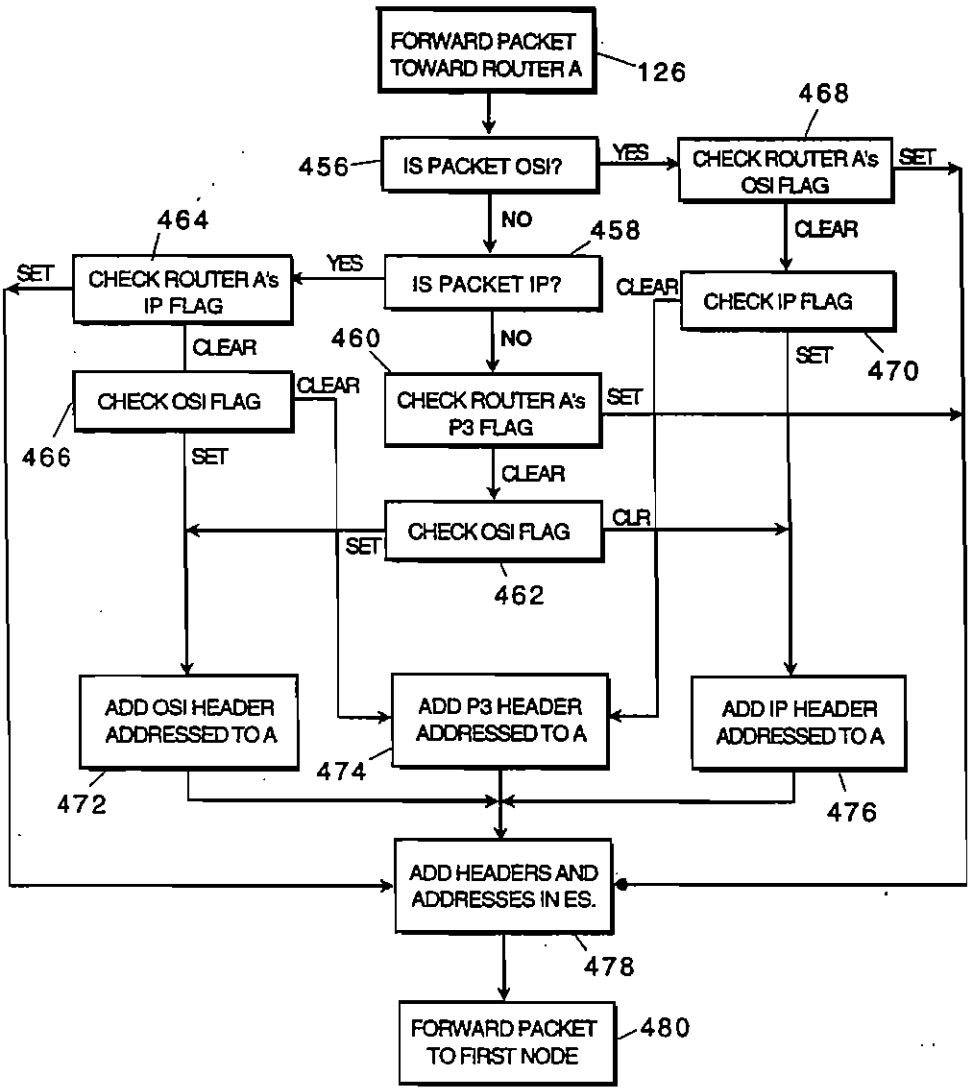

FORWARD PACKET TOWARD ROUTER A
126
468
456
IS PACKET OSI?
YES
CHECK ROUTER A's OSI FLAG
SET
NO
CLEAR
464
458
SET
CHECK ROUTER A's IP FLAG
YES
IS PACKET IP?
CLEAR
CHECK IP FLAG
CLEAR
460
NO
SET
470
CHECK OSI FLAG
CLEAR
CHECK ROUTER A's P3 FLAG
SET
466
SET
CLEAR
CLR
SET
CHECK OSI FLAG
462
ADD OSI HEADER ADDRESSED TO A
ADD P3 HEADER ADDRESSED TO A
ADD IP HEADER ADDRESSED TO A
474
472
476
ADD HEADERS AND ADDRESSES IN ES.
478
FORWARD PACKET TO FIRST NODE
480

FIG. 11

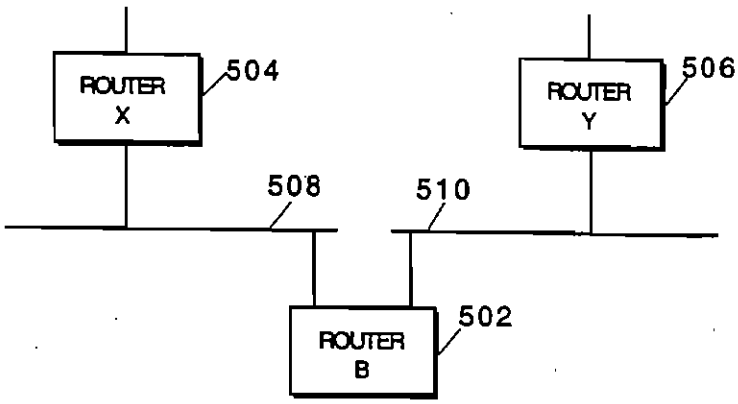

ROUTER X
504
ROUTER Y
506
508
510
ROUTER B
502

FIG. 13

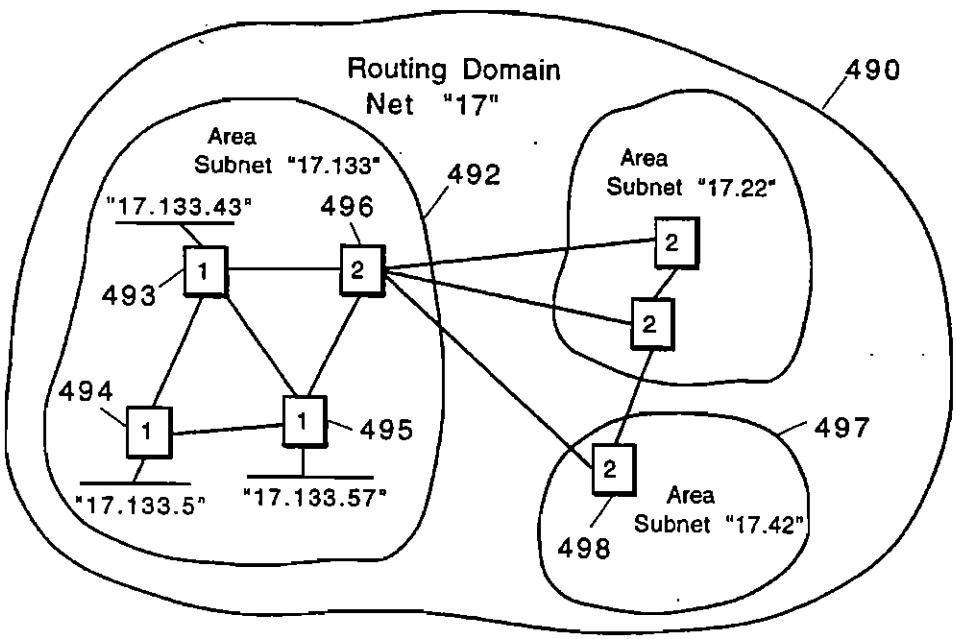

Routing Domain
Net "17"
490
Area
Subnet "17.133"
492
"17.133.43"
496
493
494
495
"17.133.5"
"17.133.57"
Area
Subnet "17.22"
Area
Subnet "17.42"
497
498

FIG. 12

MULTIPLE PROTOCOL ROUTING

BACKGROUND OF THE INVENTION

This invention relates to routing algorithms which support multiple protocols.

The end systems (e.g., computers or printers) which form a computer network are interconnected by devices known as routers. Each end system is attached to one of the network's routers and each router is responsible for forwarding communications to and from its attached end systems.

The routers transfer information to and from the end systems and among themselves along communications links in formatted packets. For example, when an originating end system wishes to transmit information to a destination end system, it generates a packet header in an appropriate format (this header includes, among other information, the address of the destination end system), and then fills the remainder of the packet with the information to be transmitted. (In the following description, the term "user data packet" will refer to the entire packet, i.e. the formatted header and the information that is to be transmitted from end system to end system.) The completed user data packet is then provided to the router attached to (and responsible for) the originating end system, which forwards it toward the destination end system. Packets transmitted among the routers themselves (which will be referred to as "control packets") are similarly formatted and forwarded.

When a router receives a user data packet, it reads the user data packet's destination address from the user data packet header, and then transmits the user data packet on the link leading most directly to the user data packet's destination. Along the path from source to destination, a user data packet may be transmitted along several links and pass through several routers, each router on the path reading the user data packet header and forwarding the user data packet accordingly.

To determine how user data packets should be forwarded, each router typically knows the locations of the network's end systems (i.e., which routers are responsible for which end systems), the nature of the connections between the routers, and the states (e.g., operative or inoperative) of the links forming those connections. Using this information, each router can compute effective routes through the network and avoid, for example, faulty links or routers. A procedure for performing these tasks is generally known as a "routing algorithm".

In popular routing algorithms such as that described in "The New Routing Algorithm for the ARPANET" by McQuillan et al., *IEEE Transactions on Communications*, May, 1980, each router determines which end systems are attached to it, what links to other routers are available, the states of those links, and the identities of the routers on the other ends of those links. To initialize the network, each router places this information in a control packet known as a Link State packet (LSP), and transmits this LSP to all of the other routers in the network (i.e., "advertises" its neighbors and end systems to the network). Later, when changes in the network occur (e.g., a link fails), one or more routers may generate new LSPs which supersede previously generated LSPs.

As long as the most recent LSPs are propagated reliably to all of the routers, each router will have complete information about the topology of the network

and can generate a routing database describing routes through the network (for example, using the algorithm described in "A Note on Two Problems in Connexion with Graphs" by Edsgar Dijkstra, *Numerische Mathematik*, Vol. 1, 1959, pages 269–271).

In order for user data packets to be delivered to their destinations, each end system on the network must have an unambiguous address. There are several independent standards organizations which document and promulgate address allocation schemes, as well as control and user data packet formats which may be used for communicating under these schemes. Several networks of routers have been configured according to these addressing schemes and formats.

In what follows, the term "routing protocol" will be used to describe the combination of an address allocation scheme and a format (or a group of related formats) that describes control packets and other information to be exchanged among routers, and which is used to calculate the paths which the user data packets will take between routers. Routing protocols are often associated with their own routing algorithms, each routing algorithm being documented and promulgated by the standards organization responsible for the associated routing protocol.

Further, the term "protocol suite" will be used to describe the comprehensive set of protocols that is designed to work together to coherently provide complete communication capabilities. Two examples of protocol suites are the TCP/IP protocol suite and the OSI protocol suite. Each of these protocol suites includes one or more network layer protocols which define the format used for control and user data packets that are to be transferred by the network routers. For example, the IP protocol of the TCP/IP protocol suite defines the network layer protocol which makes up the format of the user data packets that are forwarded by the IP routers. Similarly, the OSI routers forward user data packets following the ISO 8473 network layer protocol.

The Transmission Control Protocol (TCP) RFC 793, Internetwork Protocol (IP) RFC 791, the Internet Control Message Protocol RFC 792, and the TCP/IP protocol suite which is described in RFC 1122, "Requirements for Internet Hosts—Communication Layers," RFC 1123, "Requirements for Internet Hosts—Application and Support," RFC 1100, "IAB Official Protocol Standards," and RFC 1009, "Requirements for Internet Gateways," and associated other RFCs, all available from SRI International, DDN Network Information Center, Room EJ291, 333 Ravenswood Avenue, Menlo Park, Calif. 94025, have recently been growing in importance as a multi-vendor communications architecture, and many networks are based on them. (TCP/IP terminology refers to routers as "gateways", and end systems as "hosts".) At the same time, however, existing networks also use the Open Systems Interconnection (OSI) protocols such as described in International Organization for Standardization (ISO) 7498, "Information Processing Systems—Open Systems Interconnection—Basic Reference Model," ISO 8473, "Protocol for Providing the Connectionless-mode Network Service," ISO 9542, "End System to Intermediate System Routing Exchange Protocol," and ISO DP 10589 "Intermediate System to Intermediate System Intradomain Routing Exchange Protocol," all available from BSi Standards, 2 Park Street, London England,

W1A 2BS, and more are expected to be created as OSI is further developed.

Because the several existing protocols (in particular TCP/IP and OSI) have been developed independently, they are largely incompatible.

Networks having incompatible protocols cannot make their links and routers available to each other, which may result in excess redundancy and reduced efficiency. Also, each of the networks must be maintained independently, increasing the total management effort over what would be required by a single, universally compatible network.

One known way to share routing resources among networks having incompatible protocols is known as gateway encapsulation. In gateway encapsulation, a network using protocol A is provided with a path through a second network that uses protocol B. To form the path, two routers are manually configured to provide gateways from protocol A to protocol B, and back.

Each gateway router is configured to be fluent in both protocol A and protocol B. When a user data packet formatted in protocol A is received, at a gateway router, the gateway router "encapsulates" the user data packet in a protocol B header (i.e., generates a protocol B header and places the protocol A user data packet, including the protocol A header, into the data area of the protocol B user data packet). The encapsulated protocol A user data packet is then addressed to the second gateway router, and transmitted through the protocol B network.

When the encapsulated protocol A user data packet is received at the second gateway router, the protocol B encapsulation header is removed, and the protocol A user data packet is forwarded to its destination through the protocol A network.

The routing algorithm used in the protocol A network is suitably modified to make the protocol A routers aware of the gateway, so that user data packets destined to routers at the other end of the gateway are forwarded to one of the gateway routers.

One disadvantage of gateway encapsulation is that the gateway must be manually configured, and thus must also be manually maintained. If a change to the gateway path is desired, or if an additional gateway path is added, the gateway routers must be manually adjusted to effect the desired changes. Furthermore, the gateway path is only available to the protocol A network as long as the gateway routers themselves are operational. If a gateway router fails, the pre-established path through the protocol B network is severed, and communications must be re-routed through the protocol A network.

Another way to route multiple incompatible protocols is known as "ships in the night" (SIN). In the SIN approach, the same physical resources (e.g., routers and links) are used to route completely separate protocols. The shared resources multiplex between the supported protocols, and the protocols themselves operate more or less independently.

With SIN, there is some degree of competition for resources, because implementing two protocols requires additional programming time as well as additional CPU and memory resources in the routers. Furthermore, SIN requires that two complete sets of control packets be distributed through the two independent networks, which doubles the control traffic overhead over that of a single integrated network. Finally, configuring and managing multiple independent networks is less efficient than configuring and managing a single integrated network of the same total size: multiple networks must be configured independently, increasing overhead; and multiple networks cannot be managed as efficiently as a single network because the implicit interactions between the networks (such as the shared loads placed on routers and links) cannot be well characterized and controlled.

SUMMARY OF THE INVENTION

In one aspect, the invention features a method of calculating routes for sending user data packets through an interconnected network of information handling devices. Each of the user data packets includes destination address information conforming to an addressing convention of any one of two or more different independent protocol suites. The routes for user data packets are always calculated using a single route calculation algorithm corresponding to the same routing protocol (which is chosen from an arbitrary protocol suite) regardless of the addressing convention to which the user data packet conforms. This calculation does not involve converting the destination information from one addressing convention to another.

Preferred embodiments of this aspect include the following features.

There are exactly two protocol suites.

Link state packets conforming to the routing protocol used to compute routes are sent to the information handling devices, and routes are calculated based on information contained in the link state packets.

The OSI IS-IS routing protocol is used to compute routes.

In another aspect, the invention features a method for calculating routes for sending user data packets via information handling devices which are interconnected in a communications network. The information handling devices include (a) single-protocol information handling devices capable of recognizing and forwarding only user data packets which conform to a single protocol suite, and (b) multi-protocol information handling devices capable of recognizing and forwarding user data packets which conform to any one of two or more protocol suites. A routing protocol is used to automatically predetermine at which information handling devices to encapsulate and to decapsulate a given packet.

Preferred embodiments of this aspect include the following features.

The information handling devices are organized in areas and all of the information handling devices within a single area are at least capable of recognizing and forwarding user data packets which conform to the same one of the protocol suites. This common protocol suite may be different for different areas.

A given area may include (a) single-protocol information handling devices capable of recognizing and forwarding only user data packets which conform to a first protocol suite, (b) single-protocol information handling devices capable of recognizing and forwarding only user data packets which conform to a second different protocol suite, and (c) at least one multi-protocol information handling device capable of recognizing and forwarding user data packets which conform to the first and second protocol suite (and/or every other protocol suite that is used by any other information handling device in the area).

There are exactly two protocol suites, one which is the TCP/IP protocol suite and another which is the OSI protocol suite.

Link state packets conforming to the routing protocol used to compute routes are sent to the information handling devices, and routes are calculated based on information contained in the link state packets.

The OSI IS-IS routing protocol is used to compute routes and determine at which information handling devices to encapsulate and to decapsulate.

In another aspect, the invention features a method of enabling user data packets to be forwarded from one local area network to another by a device which is capable of acting as a router to recognize and forward user data packets which conform to a first protocol suite and is capable of acting as a bridge to recognize and forward user data packets which conform to at least a second protocol suite. For a user data packet which conforms to the first protocol suite and is addressed to a single address which is not an address of the device, the devices acts as a bridge rather than as a router.

Other features and advantages of the invention will be apparent from the following description of the preferred embodiments and from the claims.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

We first briefly describe the drawings.

FIG. 1A is a diagram of a dual-protocol network segregated on a per-area basis which does not require encapsulation.

FIG. 1B is a flow diagram of the user data packet receiving algorithm to be followed by the routers of FIG. 1A.

FIG. 1C is a flow diagram of the user data packet forwarding algorithm to be followed by the routers of FIG. 1A.

FIG. 2A is a diagram of a dual-protocol network segregated on a per-area basis which requires encapsulation.

FIG. 2B is a flow diagram of the user data packet receiving algorithm to be followed by the dual-protocol routers of FIG. 2A.

FIG. 2C is a flow diagram of the user data packet forwarding algorithm to be followed by the dual-protocol routers of FIG. 2A.

FIG. 3 is a diagram of a dual-protocol network which is not segregated on a per-area basis.

FIG. 4A is a flow diagram of an algorithm used to build routing trees.

FIG. 4B is a diagram of a routing tree.

FIG. 5A is a diagram of a data structure of the routing tree of FIG. 4B as used in a dual-protocol router of FIG. 3.

FIG. 5B is a flow diagram of an algorithm for initializing the routing trees in the dual-protocol routers of FIG. 3.

FIG. 5C is a flow diagram of an algorithm for building the routing trees in the dual-protocol routers of FIG. 3.

FIG. 6 is a flow diagram of the user data packet forwarding algorithm to be followed by the dual-protocol routers of FIG. 3.

FIG. 7A illustrates a three-protocol network which does not have an all-protocol router.

FIG. 7B illustrates a three-protocol network having an all-protocol router.

FIG. 8A is a diagram of a data structure of the routing tree of FIG. 4B as used in a dual-protocol router of FIG. 7B.

FIG. 8B is a flow diagram of an algorithm for initializing the routing trees in the dual-protocol routers of FIG. 7B.

FIG. 8C is a flow diagram of an algorithm for building the routing trees in the dual-protocol routers of FIG. 7B.

FIG. 9 is a flow diagram of the user data packet forwarding algorithm to be followed by the dual-protocol routers of FIG. 7B.

FIG. 10A is a diagram of a data structure of the routing tree of FIG. 4B as used in the all-protocol router of FIG. 7B.

FIG. 10B is a flow diagram of an algorithm for initializing the routing tree in the all-protocol router of FIG. 7B.

FIG. 10C is a flow diagram of an algorithm for building the routing tree in the all-protocol router of FIG. 7B.

FIG. 11 is a flow diagram of the user data packet forwarding algorithm to be followed by the all-protocol router of FIG. 7B.

FIG. 12 illustrates a common area structure.

FIG. 13 illustrates a brouter.

GENERAL DESCRIPTION

Before discussing the invention, it will be useful to define some terminology.

A router that is fluent in only one of the protocols supported by the multi-protocol network will be generally referred to as a "single-protocol" router. Where necessary, the protocol supported by a single-protocol router may be indicated by explicit reference to the protocol. For example, a single-protocol router which is fluent in only protocol P1 may be referred to as a "P1-only" router.

Routers that are fluent in more than one of the supported protocols interconnect the dissimilar single-protocol routers. A router that is fluent in more than one of the supported protocols will be generally referred to as a "multi-protocol" router. Routers that are fluent in two protocols will be referred to as "dual-protocol" routers; routers that are fluent in three protocols will be referred to as "three-protocol" routers, and so on. Multi-protocol routers that are fluent in all of the protocols supported by a given network will be referred to as "all-protocol" routers. (Note that where only two protocols are supported, the terms multi-protocol, dual-protocol, and all-protocol are synonymous; however, this synonymity does not apply when more than two protocols are supported.)

There are two types of multi-protocol routers. A "simple multi-protocol router", will route user data packets conforming to two or more protocols. However, a simple multi-protocol router is only able to forward the user data packets; it is not able to perform encapsulation or decapsulation to make user data packets in one protocol compatible with another protocol. An "enhanced multi-protocol router" will not only route user data packets conforming to two or more protocols, but is also able to perform encapsulation or decapsulation to make user data packets in one protocol compatible with another protocol. The extent to which enhanced multi-protocol routers may or may not support complex combinations of the other three types of routers is described in detail below.

Note that there is a distinction between the capabilities of a particular router, and its operation at any one time. For example, a router may be a multi-protocol router, but may be configured to operate as a single-protocol router. This would allow it to interoperate with other single-protocol routers in a network which only supports one protocol. The term P1-capable will be used to refer to any router which is able to forward user data packets formatted (and addressed) in accordance with protocol P1. Thus the set of P1-capable routers includes P1-only routers and any multi-protocol routers which include P1 among the set of protocols that they are capable of handling.

Also, note that an end system that communicates with routers and other end systems in accordance with a given protocol can only be connected to a router that is capable in that protocol. For example, an end system designed to communicate in protocol P1 can only connect to routers that are P1-capable. This has implications on the network architectures, as will be seen more fully below.

In one particular embodiment, the invention is applied to a dual-protocol network supporting the TCP/IP protocol suite defined by the Internet Engineering Task Force (IETF) as well as the OSI protocol suite defined by ISO. The following discussion will be directed to this particular set of protocols, although the invention is not limited to these protocols, and the inventive concepts set forth below may be applied to other groups of protocols.

To apply the invention to TCP/IP and OSI, these protocols are slightly modified. Primarily, the format of various control packets such as the link state packets and the router to router "hello" packets are normalized so that all of the routers can connect to each other and become aware of the full topology of the multi-protocol network. In one embodiment, the common format comprises the current ISO DP 10589 format, with additional fields appended to the format, as allowed by the current format. In particular, additional optional fields are added to the hello packets, link state packets, and sequence number packets. The other packet formats (in particular user data packet formats defined by ISO 8473 and IP, as well as control packet formats for router to end system communications), and the addressing schemes of the various different protocols, are not modified. ISO 9542 defines the control packet formats that are used for communications between routers and end systems in an OSI environment. Since one particular type of ISO 9542 packet format (the Intermediate System Hello packet) performs "double duty", and is also used for communications between routers over point-to-point links, this packet type will be modified by the addition of information in a manner that routers can interpret correctly, and that standard OSI end systems will ignore.

Particular modifications to the protocols suitable for one embodiment of the invention will be discussed under the heading "Detailed Description", below. The following general discussion of the invention will assume that the modifications described above have been made to the underlying protocols. Therefore, any reference to a particular protocol should be understood as a reference to the protocol as modified to conform with the goals of the preceding paragraph.

In an IP-OSI dual-protocol network, the types of routers discussed above can be further defined as follows:

(1) An "IP-only router" can understand and forward only user data packets conforming to RFC 791. Such a router conforms to the requirements for Internet gateways (as currently defined in RFC 1009, available from SRI International at the above address). However, an IP-only router will not be able to forward 8473 packets, and may operate without regard for ISO packet and addressing standards, except that IP-only routers may use OSI 10589 as their routing protocol.

(2) An "OSI-only" router can understand and forward only user data packets conforming to OSI 8473. Such a router conforms to the other requirements for OSI routers (including the end system-intermediate system protocol, ISO 9542, available from BSi Standards at the above address). However, an OSI-only router will not forward TCP/IP packets, and may operate without regard for IETF packet and addressing standards.

(3) A "simple dual-protocol router" can understand and forward both 8473 and IP user data packets simultaneously. However, a simple dual-protocol router is not be able to encapsulate IP user data packets inside OSI user data packets for transmission through an OSI-only router, or alternatively, encapsulate OSI user data packets inside IP user data packets for transmission through an IP-only router.

(4) An "enhanced dual-protocol router" can understand and forward both 8473 and IP user data packets simultaneously. In addition, an enhanced dual-protocol router can encapsulate IP user data packets inside OSI user data packets for transmission through an OSI-only router, and similarly can encapsulate OSI user data packets inside IP user data packets for transmission through an IP-only router. The way in which the encapsulation is performed will be discussed below.

Note that, of the four types of routers described above, types 1, 3, and 4 are all IP-capable. Similarly, types 2, 3, and 4 are all OSI-capable.

Even though dual-protocol routers are more flexible than IP-only or OSI-only routers, some vendors may still wish to implement single-protocol routers. Thus, even in a dual-protocol network, there may be IP-only and OSI-only routers, which will have to connect to and coexist with dual-protocol routers, and with each other. A main issue that must be confronted is how to control this interaction to provide for the correct forwarding of user data packets. Several options are proposed below.

The most direct way to allow interaction between the protocols is on a "per-area" basis. Referring to FIGS. 1A and 2A, large networks are often organized hierarchically; that is, partitioned into smaller networks 90, 92, 94, 137, 139. For the purposes of this discussion, these smaller partitions of the large network will be referred to as "areas"; routing within an area will be referred to as Level-1 routing, and routing between areas will be referred to as Level-2 routing.

TCP/IP routers and OSI routers are arranged into a common area structure. (Maintenance of an independent two level hierarchy for the two protocols is possible, but is complex, and does not benefit from the advantages of a common area structure discussed below.) The detailed description, which follows later, discusses methods for creating a common area structure while allowing independent use of the existing addressing schemes and packet formats, e.g., OSI and TCP/IP addressing schemes and packet formats, without requiring modification of those addressing schemes and

packet formats. In the following general description, it will be assumed that a suitable method has been employed to create such a common area structure.

In FIGS. 1A and 2A, those routers that are OSI-only, and the end systems designed to communicate with the OSI protocols, are indicated by circles. Those routers that are IP-only, and the end systems designed to communicate with the TCP/IP protocols, are indicated by squares. Those routers that are dual-protocol (either simple dual or enhanced dual) are indicated by circles within squares. The Level-1 routers (which reside within the areas 100) are marked with a "1". The Level-2 routers which connect the areas 100 are marked with a "2". The end systems are marked with an "E".

In a hierarchical network such as in FIGS. 1A and 2A, the Level-1 routers need not share link state information with all of the other Level-1 routers, rather, this information is only shared with those Level-1 routers in the same area. To permit routing of user data packets between areas, end system addresses are assigned such that, by observing an end system address, the routers in an area can determine whether the end system having that address is in the local area. (Typically this observation involves reading the more significant bits of the address and/or applying bit masks to the address). When a user data packet is sent from an end system in a given area to another end system in the same area, the user data packet is forwarded via Level-1 routing. Otherwise, if the user data packet is sent to an end system in another area, Level-1 routers in the originating area forward the user data packet to an appropriate Level-2 router in their area. In the most common embodiment, this involves forwarding to the nearest Level-2 router in their area. Level-2 routers then forward the user data packet to the nearest Level-2 router in the destination area. Then the user data packet again travels via Level-1 routing to the destination end system.

There are several embodiments of the invention which organize the dual-protocol network on a per-area basis. Two of these embodiments are illustrated in FIGS. 1A and 2A.

In the embodiment illustrated in FIG. 1A, the topology of the network is restricted such that no user data packet encapsulation is required. In each area, one of the following is true: (1) every router in the area operates as OSI-only (e.g., the area 90 on the left of FIG. 1A); (2) every router in the area operates as IP-only (e.g., the area in the center of FIG. 1A); or (3) every router in the area operates as dual-protocol (e.g., the area 94 on the right of FIG. 1A). Similarly, the Level-2 backbone operates entirely as IP-only, entirely as OSI-only, or entirely as dual-protocol.

In some embodiments, all areas operate as dual-protocol. In other embodiments, some areas operate as OSI-only, and some as dual-protocol. In other embodiments, some areas operate as IP-only, and others as dual-protocol. As illustrated in FIG. 1A, in some embodiments, some areas operate as IP-only, some areas as OSI-only, and some areas as dual-protocol. Any of these combinations are possible as long as (i) the Level-2 backbone operates in dual-protocol mode (as in FIG. 1A) or (ii) the protocol which is not supported by the Level-2 backbone is prohibited from operating outside of the individual areas.

In the architecture of FIG. 1A, there is no need for encapsulation, and thus all of the dual-protocol routers are "simple dual" routers. It may not be immediately evident that encapsulation is not necessary in the architecture of FIG. 1A; therefore, some discussion will be dedicated to this point.

First, note that encapsulation is only necessary if some of the paths that may be followed by a user data packet pass through routers that are not capable in the protocol of the user data packet. If every router in the path is capable in the protocol of the user data packet, then every router can forward the user data packet in accordance with that protocol, and the user data packet will successfully reach its destination without being encapsulated.

Given the above paragraph, it is clear by inspecting FIG. 1A that any path that may be taken by a TCP/IP user data packet or an OSI user data packet can only pass through IP-capable or OSI-capable routers, respectively. Therefore, there will never be any need to encapsulate user data packets.

To be more rigorous with the above proof (this discussion will be of use in analyzing more complex multi-protocol networks to be presented below) note that, in any dual-protocol network, the first router on any path followed by an OSI-formatted user data packet must be OSI-capable. To prove this assertion, remember that OSI-formatted user data packets can only be generated by OSI end systems or (in some special cases) by OSI-capable routers. Because all OSI end systems must be connected to OSI-capable routers, the first router on any path followed by an OSI-formatted user data packet must be an OSI-capable router. For similar reasons, the last router on any path followed by an OSI-formatted user data packet must be OSI-capable. This is true because an OSI-formatted user data packet must be addressed to an OSI end system or (in some special cases) to an OSI-capable router. (A non-OSI end system or router could not interpret or use the contents of an OSI-formatted user data packet, and even if it could, the address of a non-OSI end system or router would not be an OSI address and could not be placed in an OSI user data packet header). Again, because all OSI end systems must be connected to OSI-capable routers, the last router on a path followed by any OSI-formatted user data packet must be OSI-capable.

For the reasons given above, the first and last routers along any path followed by an OSI-formatted user data packet must be OSI-capable. For similar reasons, the first and last routers along any path followed by an IP-formatted user data packet must be IP-capable.

Now note that, within any given area in FIG. 1A, all of the routers have the same capabilities. As stated above, the first router in the path of a user data packet must be capable in that user data packet's protocol: because all of the routers in an area have the same capabilities, this means that all of the routers in the area originating a user data packet must be capable in that user data packet's protocol. Similarly, the last router in the path of a user data packet must be capable in that user data packet's protocol; because all of the routers in an area have the same capabilities, this means that all of the routers in the area to which a user data packet is addressed must be capable in that user data packet's protocol. Thus all of the routers in the areas where a user data packet starts and ends (which may be the same) are capable in that user data packet's protocol. The only other routers through which the user data packet travels are the Level-2 routers, and the Level-2 routers have sufficient capability to forward any user data packets that may be presented to them: either the Level-2 routers are dual-protocol, or, if they are not

dual-protocol, the protocol which is not supported by the Level-2 routers is prohibited from operating outside of the individual areas. Thus, in the FIG. 1A architecture, all of the routers along any user data packet's path must be capable in that user data packet's protocol, and no encapsulation is required.

One algorithm that may be used by the routers of FIG. 1A when receiving user data packets is illustrated in FIG. 1B. Referring to FIG. 1B, when a router of FIG. 1A receives 102 (either from another router or from an attached end system) a user data packet addressed to an end system Z, the receiving router first reviews 104 the LSPs it has stored to determine if there is a router that has "advertised" a connection to end system Z. The router connected to end system Z will be called router A (note that, if end system Z is not in the same area as the receiving router, router A is the nearest level-2 router in the local area). Next, the receiving router checks 106 if router A is itself, i.e., if end system Z is connected to the receiving router. If not, the user data packet is forwarded 108 to the appropriate next router on the path to router A. However, if end system Z is connected to the receiving dual-protocol router, then the receiving dual-protocol router delivers 110 the user data packet to the appropriate end system.

Referring to FIG. 1C, when a dual-protocol router forwards 108 a user data packet toward router A, it determines 111 the first router in the path to router A and then sends 112 the user data packet on the link to that first router.

The operation of the algorithms discussed above can best be shown by example. Two examples of routing in the network of FIG. 1A are provided below. The first example does not involve Level-2 routing; the second example does.

In the first example, OSI end system 101 wishes to send a user data packet to OSI end system 103. To this end, it generates an OSI header addressed to end system 103, and forwards the user data packet to OSI-only router 105. Following the flow charts of FIGS. 1B and 1C, router 105 determines that the user data packet should be forwarded to OSI-only router 107, because router 107 has advertised a connection to end system 103 in its most recent LSP. Therefore, router 105 forwards the user data packet to router 107. Following the flow chart of FIG. 1B, router 107 determines that the user data packet is addressed to one of the end systems connected to it, and thus delivers the user data packet to end system 103.

In the second example, OSI end system 103 wishes to forward a user data packet to OSI end system 109. To this end, end system 103 generates a user data packet in OSI format, which it forwards to OSI-only router 107. Following the flow charts of FIGS. 1B and 1C, router 107 determines that the user data packet is not addressed to an end system in the local area, and thus it determines that the user data packet should be forwarded to dual-protocol router 111, which is the nearest Level-2 router in router 107's local area. Thus, router 107 forwards the user data packet on the link to OSI-only router 105 (which leads most directly to router 111). Following the flow charts of FIGS. 1B and 1C, router 105 then independently determines that the user data packet is not addressed to an end system in the local area, and thus determines that the user data packet should be forwarded to dual-protocol router 111. Thus router 105 forwards the user data packet on the link to router 111. Router 111, following the flow charts of FIGS. 2B and 2C, recognizes that the OSI user data packet should be forwarded toward the area router 115. The user data packet is then

forwarded through the Level-2 dual-protocol routers 113 and 115, eventually arriving at dual-protocol router 117. Following the flow charts of FIGS. 1B and 1C, router 117 determines that the user data packet is addressed to an OSI end system which is attached to it. Therefore, it delivers the user data packet to end system 109.

Before discussing other possible embodiments of the invention which involve encapsulation, it is useful to make the following observation: most of the complications associated with single-protocol routers occur when more than one type of single-protocol router appears in the same area. If there is only one type of single-protocol router in an area, all user data packets can be placed in the protocol of the single-protocol routers (by encapsulation if necessary), and then easily routed within the area.

Therefore, referring to FIG. 2A, a second embodiment of the invention explicitly prohibits the use of IP-only and OSI-only routers in the same area. In each area, one of the following is true: (1) the area is an "OSI-preferred area", and every router in the area operates as OSI-only or dual-protocol (e.g., the area 137 on the left of FIG. 2A); or (2) the area is an "IP-preferred area", and every router in the area operates as IP-only or dual-protocol (e.g., the area 139 in the right of FIG. 2A). Although the areas are designated as OSI-preferred or IP-preferred, because both types of areas include dual-protocol routers, there may be an IP end system in an OSI area (e.g., IP end system 135), and vice-versa. As a result, IP user data packets will be transmitted through OSI areas, and vice-versa. These packets may need to be encapsulated to traverse the single-protocol routers within the areas. For this reason, in the embodiment of FIG. 2A, all of the dual-protocol routers are "enhanced dual" routers.

The Level-2 backbone may be composed entirely of dual-protocol routers, as illustrated in FIG. 2A. Alternatively, the Level-2 router in an area may be single-protocol if the protocol not supported by the Level-2 router will be not be routed outside the area. One restriction that ensures that this will be the case is as follows: the Level-2 routers for an area must support the protocol used by any end system in the area. For example, for an area to have an IP-only Level-2 router, all of the end systems in that area must be IP. This restriction ensures that the single-protocol Level-2 routers will never be expected to perform Level-2 routing on a user data packet having an incompatible protocol (because all user data packets travelling into or out of an area must be to or from and end system in the area). However, it may be desireable to allow an area to use a protocol which is not supported by its Level-2 router(s), for example if the protocol is still being tested and has not been universally adopted. In the embodiments illustrated in FIG. 2A, there must be one common protocol which is supported by all Level-2 routers. For example, it is not permitted to have an OSI-only Level-2 router and an IP-only Level-2 router in the same routing domain (this topological restriction is removed in other embodiments of the invention).

Encapsulation is necessary within areas of the architecture of FIG. 2A because, within an area, all of the routers do not have the same capabilities. In addition, if there are single-protocol Level-2 routers, encapsulation

may also be necessary at Level-2, because not all of the Level-2 routers would have the same capabilities. (Compare this situation to the discussion of FIG. 1A above, where all of the routers within an area and at Level-2 have the same capabilities, and because of this, encapsulation is not required.)

The following discussion of encapsulation will be directed to encapsulation at Level-1. However, for embodiments which have single-protocol routers at Level-2, the same techniques may be used to encapsulate at Level-2. To implement encapsulation, several issues must be confronted. First, there must be a way for routers to decide when encapsulation is necessary.

In known routing algorithms, when a router seeks to forward a user data packet, it consults a network map (generated from the link state packets) to determine the "best" path for the user data packet to follow to its destination. The user data packet is then forwarded on the link connecting to the first router on this path. (Note that the first router on the path must be on the other end of a link connected to the forwarding router.) Here, the user data packet's path is mapped in a similar manner; however, the mapping is done without regard for the protocols supported by the routers on the path.

Once the user data packet's path is chosen, the forwarding router attempts to forward the user data packet to the first router on the path. If the first router on the path does not support the user data packet's protocol, then encapsulation is required, and the forwarding router encloses the user data packet in a suitable encapsulation header before the user data packet is forwarded.

Note that only dual-protocol routers need to perform the analysis of the previous paragraph. The analysis of the previous paragraph is not necessary in single-protocol routers because they cannot encapsulate user data packets. Furthermore, the restrictions on the topology of FIG. 2A prevent any circumstances in which a single-protocol router would need to perform encapsulation (these circumstances can also be prevented in less restricted architectures, as discussed below), as follows: (1) any user data packet being forwarded by a single-protocol router is by definition formatted in the protocol supported by the single protocol router; (2) the topology of FIG. 2A prevents single-protocol routers from being connected to other routers which do not support their protocol. Thus, there are no circumstances in which a single-protocol router would need to encapsulate a user data packet.

Once a dual-protocol router decides that a user data packet needs to be encapsulated, it must have some way to address the encapsulation header so that the user data packet will arrive at the router which is expected to decapsulate the user data packet, and this destination router will know to do so. Such a mechanism prevents the user data packet from being lost in the network, and prevents the destination router from treating an encapsulated user data packet as an ordinary, non-encapsulated user data packet, and accidentally misinterpreting its contents or forwarding it without decapsulating it. To achieve these goals, encapsulated user data packets are uniquely addressed, in a manner consistent with the encapsulating protocol.

One possible way to properly address encapsulated user data packets is to assign a "dummy" end system address to each dual-protocol router (only dual-protocol routers are capable of decapsulating user data packets, therefore, only dual-protocol routers need to have such addresses). The link state packets of the dual-protocol routers "advertise" their dummy end system addresses along with other "real" end system addresses. Thus, within the LSPs, the dummy addresses are treated the same way as the addresses of real end systems connected to the dual-protocol router. When a dual-protocol router wishes to encapsulate a user data packet (only dual-protocol routers encapsulate user data packets) and send it to another dual-protocol router, it generates an encapsulation header, which it addresses to the dummy address for the destination router (the dummy address is placed in a predetermined location in the dual-protocol routers' LSPs, for example, it can be added to the beginning or end of the list of real end system addresses).

Between the encapsulating router and the decapsulating router, the encapsulated user data packet may pass through many other single- or dual-protocol routers. Each of these routers simply reads the user data packet's encapsulation header and, because the LSP of the destination router advertised the dummy address as an end system to which it is connected, the intermediate routers simply forward the encapsulated user data packet toward the destination router, in the same manner that ordinary, unencapsulated user data packets are forwarded.

When the encapsulated user data packet arrives at the destination router, the destination router recognizes the dummy address as indicating that the user data packet is encapsulating another user data packet. The destination router then removes the encapsulation header, and interprets the contents of the user data packet as a newly received user data packet.

Although the above methods determine when to encapsulate, and illustrate how to address user data packets so that they can be recognized and decapsulated, the dual-protocol router encapsulating a user data packet must also know where to send the encapsulated user data packet, i.e., which of the other dual-protocol routers to which the encapsulated user data packet should be addressed. Remember that the encapsulating router maps out the path the user data packet should follow to its destination before it decides if the user data packet should be encapsulated. Once a decision to encapsulate has been made, to minimize the number of encapsulations and decapsulations, the encapsulated user data packet should be addressed so that it passes through as many routers as possible before it needs to be decapsulated. However, the user data packet cannot travel through a router that does not support the encapsulating protocol. Therefore, the user data packet must be encapsulated before the first router on the path which does not support the encapsulating protocol. In the following, the router on the path just prior to the first router which does not support the encapsulating protocol will called the "last capable router on the path" (because this router is the last router in the chain of routers, starting at the router encapsulating the user data packet, which support the encapsulating protocol).

In some architectures (in particular, the architecture of FIG. 2A) the above-described "last capable router along the path" will always be the router to which the destination end system is connected. This is an advantage of the FIG. 2A architecture because, as discussed above, even in known networks which do not support dual-protocol protocols, routers forwarding user data packets must first determine the router to which the

destination end system is connected. Thus, in the FIG. 2A architecture, this computation not only determines how to forward the user data packet, it also determines how to address the encapsulation header. (In other multi-protocol architectures—see the discussion of FIG. 3 below—the "last capable router along the path" will not always be the router to which the destination end system is connected. In these architectures, extra computation is required to determine how to address the encapsulation header).

Because the single-protocol routers in FIG. 2A do not perform encapsulation and therefore need not determine when to encapsulate, the algorithm used by the routers in FIG. 1A, i.e., that illustrated in FIGS. 1B and 1C, can be used by the single-protocol routers in FIG. 2A.

However, the dual-protocol routers of FIG. 2A must use a more complex algorithm for receiving and forwarding user data packets. Referring to FIG. 2B, when a dual-protocol router of FIG. 2A receives 120 (either from another router or from an attached end system) a user data packet addressed to an end system Z, the receiving dual-protocol router first reviews 122 the LSPs it has stored to determine if there is a router that has "advertised" a connection to end system Z. The router connected to end system Z will be called router A. If end system Z is not in the same area as the receiving dual-protocol router, router A is the nearest Level-2 router in the local area. Next, the receiving dual-protocol router checks 124 if router A is itself, i.e., if end system Z is connected to the receiving dual-protocol router. If not, the user data packet is forwarded 126 to the appropriate next router on the path to router A. However, if end system Z is connected to the receiving dual-protocol router, then the receiving dual-protocol router checks 128 if end system Z is the "dummy" end system, i.e., if the user data packet is an encapsulated user data packet. If not, the user data packet is delivered 132 to the appropriate end system. However, if the user data packet is an encapsulated user data packet, the encapsulation header is removed 130, and the user data packet contained therein is treated as a newly received user data packet by returning to step 120.

Referring to FIG. 2C, when a dual-protocol router forwards 126 a user data packet toward router A, it first determines 133 the first router in the path to router A. The dual-protocol router then checks 134 whether the first router in the path to router A is an OSI router, an IP router, or a dual IP-OSI router.

If the first router is a dual IP-OSI router, the forwarding router directly sends 140 the user data packet on the link to that first router without the need for any encapsulation.

If the first router is IP-only, the forwarding router checks 136 if the user data packet is formatted in IP. (The detailed description below discusses how the user data packet protocols may be properly recognized.) If not, the user data packet is encapsulated 138 in an IP header, which is addressed to router A. Note that, if the destination end system is in the local area, then router A is the "last capable router on the path", as discussed above. However, if the destination end system is not in the local area, then router A is the nearest Level-2 router in the local area, which will decapsulate the user data packet (the Level-2 router must be dual-protocol if OSI and IP traffic is being routed outside the local area) and forward it via Level-2 routing to the correct area (which may involve additional encapsulations and decapsulation) where the Level-1 routing will deliver the packet (possibly after further encapsulation and decapsulation) to the destination end system. Regardless of whether the user data packet is encapsulated, it is sent 140 on the link to the first router in the path to router A.

Similarly, if the first router is OSI-only, the forwarding router checks 142 if the user data packet is formatted in OSI. If not, the user data packet is encapsulated 144 in an OSI header, which is addressed to router A. Regardless of whether the user data packet is encapsulated, it is sent 140 on the link to the first router in the path to router A.

The use of the above algorithms can be best shown by example. Two examples of routing in this architecture are provided below; the first example does not involve encapsulation, whereas the second requires encapsulation.

In the first example, TCP/IP end system 121 wishes to forward a user data packet to TCP/IP end system 123. To this end, end system 121 forms an IP user data packet addressed to end system 123. This user data packet is forwarded to IP-only router 125. Router 125, following the flow charts of FIGS. 1B and 1C, determines that the user data packet should be forwarded toward dual-protocol router 127. The user data packet is then forwarded to dual-protocol router 129, which is the first router on the path to router 127. Router 129 receives the user data packet, and following the flow charts of FIGS. 2B and 2C, also determines that the user data packet should be forwarded toward dual-protocol router 127, and that it need not be encapsulated because the next route is IP-capable (router 125 did not need to decide whether to encapsulate the user data packet because it is an IP-only router). In this way, the user data packet eventually arrives at dual-protocol router 127. Router 127 then follows the flow chart of FIG. 2B and determines that Ip end system 123 is attached to it, and that the address of end system 123 is not its dummy address. Therefore, it delivers the user data packet to end system 123.

In the second example, TCP/IP end system 135 wishes to forward a user data packet to TCP/IP end system 123. To this end, end system 135 generates a user data packet in IP format, which it forwards to dual-protocol router 131. Following the flow chart of FIG. 2B, dual-protocol router 131 determines that the user data packet is not addressed to an end system in the local area, and thus it determines that the user data packet should be forwarded to dual-protocol router 133, which is the nearest Level-2 router in router 131's local area. Also, router 131 determines that the user data packet should be encapsulated because the next router is OSI-only and the user data packet is formatted for IP. Thus, router 131 generates an encapsulation header addressed to router 133's dummy address, and forwards the user data packet on one of the links leading to router 133. The OSI-only and dual-protocol routers in router 133's area then forward the user data packet as any other OSI user data packet until it reaches dual-protocol router 133. Router 133, following the flow chart of FIG. 2B, determines that the user data packet is addressed to its dummy address, and therefore is an encapsulated user data packet. The encapsulation header is removed, and router 133 analyzes the internal user data packet. Following the flow charts of FIGS. 2B and 2C, router 133 then recognizes that the IP user data packet should be forwarded toward the nearest

Level-2 router in the destination area (i.e., router 141). The user data packet is then forwarded, without encapsulation, through the Level-2 dual-protocol routers, eventually arriving at router 141 (alternatively, where there are single-protocol routers at Level-2, there may also be encapsulation at Level-2). Following the flow charts of FIGS. 2B and 2C, router 141 determines that the user data packet should be forwarded toward router 127, and that it need not be encapsulated because the next router can understand IP. Therefore, router 141 forwards the user data packet toward router 127. When the user data packet arrives at router 127, it follows the flow chart of FIG. 2B and determines that TCP/IP end system 123 is attached to it, and that the address of end system 123 is not its dummy address. Therefore, it delivers the user data packet to end system 123.

In two simplified forms of the embodiment illustrated in FIG. 2A, either OSI-only or IP-only routers are prohibited from the network. In these embodiments, FIG. 2C can be simplified by simplifying the check at step 134, and removing either steps 142 and 144, or steps 136 and 138. In either embodiment, the Level-2 backbone may operate as all dual-protocol, or may be a mixture of dual-protocol and single-protocol routers, as long as the restriction discussed above is met (i.e., that any protocol not supported by an area's Level-2 router cannot be routed outside that area).

One advantage of the "per-area" embodiments discussed above is that they allow for a graceful evolution of the network from one protocol to another. Currently, TCP/IP is the dominant protocol suite (at least in most environments in the United States). Thus it is likely that, initially, users would choose to implement a network of IP-only and dual-protocol routers. These will operate together in a graceful manner (for example, where necessary, encapsulation will occur automatically). As OSI becomes more important, the IP-only routers would be upgraded to dual-protocol routers, probably eventually resulting in an all-dual routing domain. Assuming that OSI will eventually supersede TCP/IP, then during the last step of migration, users will gracefully replace some of the dual-protocol routers with OSI-only routers, and/or add additional OSI-only routers. All of this will be quite graceful, as long as OSI-only and IP-only routers are never placed in the same area.

The per-area embodiments of the invention described above are quite flexible and have several advantages which have been disclosed. One drawback of these embodiments is that they have topological restrictions, namely, OSI-only and IP-only routers cannot be placed in the same area. However, if there are three types of routers (IP-only, OSI-only, and dual-protocol), some users may desire (and can be expected) to connect the routers together in the same area regardless of the restrictions imposed by the routing algorithm. Therefore, in another embodiment of the invention, these per-area topological restrictions are eliminated.

The embodiment of the invention illustrated in FIG. 3 has fewer topological restrictions than the preceding embodiments, in particular, OSI-only and IP-only routers are allowed to coexist in a given area. The network of FIG. 3 may represent one area of a larger network, or it may represent the Level-2 backbone of a network, where the depicted routers reside in local areas. (Note that if FIG. 3 represents an area, the Level-2 router in the area must be dual-protocol if both OSI and IP traffic is to be routed outside of the local area.)

The topology of FIG. 3 is more flexible than the topologies of FIGS. 1A or 2A, but routing user data packets through this topology is more complicated. Specifically, the following complications arise when OSI-only and IP-only routers are mixed within an area.

A first complication is that users may accidentally connect links between routers that do not support a common protocol. There must be a mechanism to ensure that such links are never recognized or used by the routers, because all user data packets sent along such links will be lost. For example, the erroneously connected link between OSI-only router 166 and IP-only router 168 (indicated by a dotted line) must never be used. There are several possible ways to prevent such links from being used.

One solution is to configure the OSI-only and IP-only routers to use different values in the protocol data unit (PDU) type field (see Section 9 of the above-referenced 10589 specification) of their hello packets. Therefore, OSI-only routers would think that the hello packets sent by the IP-only routers are invalid, and vice-versa.

Another solution is to encapsulate the hello packets sent by IP-only routers inside other IP packets, so that they are ignored by OSI-only routers.

A third solution is to alter the sub-network dependent link initialization procedures in IP-only routers or OSI-only routers (or both), so as to prevent the link from becoming valid. It may be more convenient to perform these modifications in the IP-only routers because, currently, the IP specification is less rigid.

A second complication with the FIG. 3 embodiment is that IP-only and OSI-only routers can occur on the same LAN. (Note that if a LAN spans two areas, ISO 10589 ensures than the Level-1 routers from different areas on the LAN will ignore each other. This ensures that this issue does not arise in the previous "per-area" embodiments.) Providing dual protocols on LANs raises several issues that must be confronted.

Each LAN has an elected "designated router". The designated router is responsible for updating the LAN routers about the LAN topology. Although the actual topology is a single link connecting all of the routers, the topology reported by the designated router is a star, where the center of the star is a "pseudonode". (The designated router generates a link state packet for the pseudonode indicating a star connection to all of the other routers on the LAN.) As a result, the first router on the path to the user data packet's destination is always the pseudonode, and the second router on the path is the router to which the user data packet should be directly transmitted. The pseudonode is a notational convention—when transmitting a user data packet over a LAN, the user data packet is typically forwarded to the second router on the path to its destination, rather than to the first router.

The above mechanism works properly when the routers on the LAN all support one of the protocols. For example, if there are no IP-only routers on the LAN, then OSI-only routers may forward user data packets on the LAN in the normal fashion (because any user data packets forwarded by an OSI-only router must be formatted for OSI, and all routers on the LAN support OSI). Also, dual-protocol router may forward user data packets on the LAN in the manner described above (i.e., OSI user data packets are forwarded in the normal fashion, and IP user data packets are encapsulated if the next router is OSI-only). The designated router may be chosen to be an OSI-only or dual-

protocol router, because either will be able to exchange control packets with the remainder of the routers on the LAN, and thus learn the full topology of the LAN. The situation is similar where there are no OSI-only routers on the LAN.

If there are OSI-only and IP-only routers on the LAN, then several difficulties arise. First, an appropriate "designated router" must be elected for the LAN. IP-only routers cannot correctly interpret fields in control packets exchanged with OSI-only routers (and vice-versa); therefore, the designated router cannot be either IP-only or OSI-only, because neither can correctly learn the entire LAN topology and generate the appropriate link state packet for the pseudonode. There are two possible solutions.

The first solution is to elect a dual-protocol router as the designated router. The dual-protocol router can correctly exchange control packets with all of the other routers, and thus can learn the entire LAN topology. To do this, the dual-protocol designated router send two sets of hello packets, one of which is interpreted by dual-protocol and IP-only routers and one of which is interpreted by dual-protocol and OSI-only routers. Once it has learned the LAN topology, the dual-protocol router then generates a link state packet for the pseudonode, which lists all of the routers on the LAN, regardless of their protocol capabilities. Note that if there are no single-protocol routers on the LAN, the dual-protocol routers send hello packets in a "preferred" format, for example the OSI format, elect one dual-protocol router as designated router, and use the "preferred" format for control and user data packets on the LAN.

The first solution can only be used if there is a dual-protocol router on the LAN. If there are no dual-protocol routers on the LAN, the second solution must be used. In the second solution, two designated routers are elected, one which is IP-capable and one which is OSI-capable. To do this, the IP-capable routers on the LAN exchange control packets with other IP-capable routers, and elect one as an IP-capable designated router, and similarly, the OSI-capable routers exchange control packets with other OSI-capable routers, and elect one as an OSI-capable designated router. (If there is a dual-protocol router on the LAN and the second solution is chosen, then the dual-protocol router may be elected as both pseudonodes.) The IP-capable and OSI-capable pseudonodes then exchange control packets with the other IP-capable or OSI-capable routers, respectively, and learn that portion of the LAN topology which is capable in their protocol. Next, the two pseudonodes generate two link state packets, one for an "IP pseudonode" and one for an "OSI pseudonode". The IP pseudonode link state packet lists only the IP-capable routers on the LAN, and the OSI pseudonode link state packet lists only the OSI-capable routers on the LAN. (If there is one or more dual-protocol router on the LAN and the second solution is chosen, then the dual-protocol router(s) will be listed in both pseudonode link state packets.)

When IP-only and OSI-only routers occur on the same LAN, the forwarding of user data packets on that LAN may also be more complicated.

As discussed above, when forwarding a user data packet on a LAN, typically the user data packet is sent to the second router on the path to its destination (i.e., the packet is sent to the router after the pseudonode). To allow OSI-only and IP-only routers on the same LAN, the following restrictions are added: (i) if the router doing the forwarding is dual-protocol, it first determines whether the user data packet is compatible with the second router—if not, it encapsulates the packet before forwarding it; (ii) if the router doing the forwarding is a single-protocol router, and there are two pseudonodes, then all routers that the single-protocol router is aware of (i.e., those listed in the appropriate pseudonode LSP) will be capable in the protocol of the single-protocol router, and the user data packet can be forwarded in the normal fashion; however (iii) if the router doing the forwarding is a single-protocol router, and there is only one pseudonode, then it is possible that the second router (i.e., the router after the pseudonode) may be a single-protocol router supporting the wrong protocol suite. Therefore, in this case the single-protocol router forwards the user data packet to a dual-protocol router on the LAN (perhaps to the designated router, which must be dual), rather than to the second router. The dual-protocol router can then encapsulate the packet if required.

A third complication with the FIG. 3 embodiment is that encapsulation is more complex. Note that, in the topology of FIG. 3, user data packets may need to be encapsulated to traverse the single-protocol routers within the areas; therefore, all of the dual-protocol routers must be "enhanced dual" routers. For example, consider an OSI user data packet which needs to traverse a path from OSI end system **155** connected to OSI-only router **154** to OSI end system **157** connected to OSI-only router **156**. The user data packet (at minimum) must be encapsulated by dual-protocol router **162**, so that it can traverse IP-only router **163**. Also, the user data packet must be decapsulated by router **164** before it can be sent to router **166**, however, it must remain encapsulated if it is to be sent through router **168** and then decapsulated at router **170**.

In the above discussion of FIG. 2A, it was asserted that, to minimize the number of encapsulations and decapsulations, an encapsulated user data packet should be addressed so that it passes through as many routers as possible before it needs to be decapsulated, i.e., to the last router along the path that supports the encapsulating protocol. It was also noted that, in more complex architectures, the "last capable router along the path" will not always be the router to which the destination end system is connected. This is clear from the above example, as the "last capable router on the path" may be router **164** (if it is desirable to route through router **166**), or it may be router **170** (if it is desirable to route through router **168**).

The encapsulation header must be addressed to the "last capable router along the path" for the user data packet to be correctly decapsulated. To allow this, the above-referenced Dijkstra algorithm must be modified so that it not only indicates which neighboring router to forward the user data packet to, but (where necessary) also specifies which destination address needs to be used for the encapsulating header (i.e., which router is the "last capable router along the path").

A routing tree such as illustrated in FIG. 4B is generated by the Dijkstra algorithm of FIG. 4A. Referring to FIGS. 4A and 4B, to initialize the Dijkstra algorithm, the router generating the tree first creates 180 (FIG. 4A) a node **181** (FIG. 4B) representing itself at the root of the tree. Next, nodes **183** representing the neighbors of the router generating the tree are placed **182** below the root of the tree as "tentative". The relationship of

node 181 to 183 is a parent-child relationship. In the following, the node further from the root node 181 may be referred to as the "child", and the node closer to the root node 181 may be referred to as the "parent".

When a node representing a router is placed on the tree tentatively, it indicates that one path through the network to the router has been located (namely, the path through the routers represented by the nodes between the tentative node and the root node 181). As will be seen below, tentative nodes are made permanent once the Dijkstra algorithm verifies that the tentative path to the represented router is the best possible path. To perform this verification, the algorithm performs a loop, locating permanent paths to each router in the network, until paths to all routers have been determined. The routing tree in FIG. 4B represents a tree in the midst of construction. The nodes in FIG. 4B with a dark outline are permanent, whereas the nodes in FIG. 4B with a light outline are tentative—nodes 187 have been added as tentative nodes, and node 189 has been made permanent.

In each pass of the Dijkstra loop, the "closest" tentative node to the root node 181 is located 184 and made permanent (the length, or "cost", of a path from the router represented by the root node to the router represented by another node is calculated using a suitable metric such as link delay times). The closest tentative node may be made permanent because the path from the root to the closest tentative node must be the best path to the router represented by the tentative node; any other possible paths to the represented router would have to pass through one of the other tentative nodes, and thus would by definition be "longer" paths (because links never have negative cost).

Once the closest tentative node is made permanent, the routers which neighbor the router represented by the newly permanent node are added 186 tentatively to the tree as children of the newly permanent node. However, if any of these neighboring routers is already represented by permanent node on the tree, then a tentative node representing the neighboring router is not added to the tree (because a better path to the neighboring router has already been found). Also, if any of the neighboring routers is already represented by a tentative node on the tree, then a comparison is made between the existing path to the neighboring router (i.e., the path represented by the existing tentative node) and the new path to the neighboring router. If the new path is "shorter" (in the sense discussed above) than the existing path, a new tentative node representing the new path is added to the tree, and the existing tentative node is removed. Otherwise, the existing tentative node is retained, and no new tentative node is added.

There is the possibility that two paths to the same router will be equal in "length" or very similar in length. In these cases, it is necessary and/or useful to implement a tie-breaking routine, which breaks such ties or near ties in a well-behaved manner. This can help to share network traffic loads between paths of equal or almost-equal cost.

Finally, 188, the algorithm checks if there are any tentative nodes left (i.e., if there are any routers to which paths have been established, but to which best paths have yet to be determined). If not, the algorithm ends. Otherwise, the algorithm proceeds to step 184, and makes another tentative node permanent.

Referring to FIG. 5A, several fields of data may be contained in each of the node data structures of the tree of FIG. 4B. Fields 190 contain general information useful for performing the Dijkstra algorithm.

The name field 192 contains the name of the router which a node represents. The name of the router represented by a node is placed in the name field when the node is first created.

The first hop field 194 contains the name of the first router on the path from the router generating the routing tree to the router which the node represents. This field is used by the router generating the routing tree to determine where (i.e., to which of its neighboring routers) a user data packet should be forwarded. This information is generated as follows: when a node 183 representing a router which directly neighbors the router generating the routing tree (represented by node 181) is added to the tree (step 182, FIG. 4A), the name of the neighboring router (i.e., the router represented by the node 183) is placed in the name field 192 and in the first hop field 194 of the node 183. Thereafter, when another tentative node is added to the tree (step 186, FIG. 4A) below a parent permanent node, the first hop field 194 in the new node is set to the value of the first hop field 194 in the parent node. In this way, the first hop values are established when the nodes 183 are added to the tree, and these values are maintain throughout the branches of the tree leading from the nodes 183.

The status field 196 indicates whether the router represented by a node is in operation, and may also represent the level of operability of the router or its links. This information may be used to route user data packets around faulty routers, or to wait for a router to become operable before sending user data packets.

The cost field 198 indicates the total cost of the route from the router represented by the root node 181 to the router represented by the node. This information is used when comparing two tentative nodes to determine which is closer to the root node 181. The cost information may be derived from one of many well known metrics. The cost is determined by adding the cost of the link from the parent node to a new child node to the cost of the parent node. The cost of the root node 181 is defined to be zero, or another predetermined value.

Fields 190 contain enough information to generate a routing tree from LSPs, and to forward user data packets along a path having the lowest cost according to the selected cost metric. However, as discussed above, the embodiment illustrated in FIG. 3 is complex because, if encapsulation is required, it is necessary for the Dijkstra algorithm also to specify which destination address needs to be used for the encapsulating header.

To this end, additional fields 200 are added to the node data structures. These fields contain information used in encapsulation, and are filled out as described below. Because single-protocol routers in FIG. 3 do not perform encapsulation, the single-protocol routers generate a routing tree having only the fields 190. However, dual-protocol routers in FIG. 3 also generate fields 200, because they perform encapsulation.

The encapsulation router (ER) field 202 indicates the name of the last router along the path to the router represented by the node which supports the encapsulating protocol. This is the router to which the encapsulating header should be addressed.

To clarify this idea, refer to the example of FIG. 3, where an OSI user data packet was being sent from OSI end system 155 to OSI end system 157. For the purpose of this example, assume that the lowest cost path passes from OSI-only router 154 through routers 162, 163, 159,

158, 164, 166, 170, and 156. Then dual-protocol router 162 must encapsulate the user data packet in an IP header, and address the IP header to dual-protocol router 164, because dual-protocol router 164 is the last router along the path which supports the IP protocol.

Now assume that the lowest cost path passes from OSI-only 154 through routers 162, 163, 159, 158, 164, 168, 170, and 156. In this case, dual-protocol router 162 must still encapsulate the user data packet in an IP header, but the header must be addressed to dual-protocol router 170, because router 170 is now the last router along the path which supports the IP protocol.

Note that alternative encapsulation methods are possible: for example, user data packets can be encapsulated to the first dual-protocol router along the path to the use data packet's destination. In this case, router 162 would address the IP encapsulation header to dual-protocol router 164. However, if the lowest cost path passed through router 168, router 164 would remove the header, determine that the user data packet should be forwarded through router 168, and then add a new encapsulation header addressed to dual-protocol router 170, and finally forward the user data packet to router 168. This method involves an additional encapsulation and decapsulation, which is avoided by sending the encapsulated user data packet to the "last capable router along the path".

Note that the "last capable router along the path" may simply be the last router along the path. For example, consider a user data packet being sent from OSI end system 155 to OSI end system 169. In this case, when dual-protocol router 162 encapsulates the user data packet in an IP header, the last capable router along the path will be dual-protocol router 164. Therefore, the encapsulation header should be addressed to dual-protocol router 164.

Furthermore, note that a dual-protocol router may not always need to encapsulate a user data packet. For example, consider an OSI user data packet being forwarded from OSI end system 155 to OSI end system 172. In this case, when the user data packet arrives at dual-protocol router 162, there is no need for encapsulation, and it can be simply forwarded in OSI format to router 167. Therefore, for some destination routers, there is no "last capable router along the path", simply because every router along the path is capable.

A value of "?" in the ER field indicates that the value for the "last capable router along the path" is not needed. This may be because, even though encapsulation is required, the last router along the path is the last capable router (as in the embodiment of FIG. 2A), or, alternatively, because encapsulation is not required.

Referring to FIG. 5A, as the routing tree is being generated, two flags are used to determine whether encapsulation is required, and which router "is the last capable router along the path". These flags are referred to as the IP flag 204 (which indicates whether the first router along the path is IP-capable) and the OSI flag 206 (which indicates whether the first router along the path is OSI-capable). These flags are used to determine the value of the ER field as the routing tree is built.

Referring to FIG. 5B, during step 182 of the Dijkstra algorithm, nodes for the routers neighboring the router generating the tree are added to the tree. For dual routers, this step involves updating the ER field and the OSI and IP flags. A loop is initialized 210 at the first neighbor, and for each neighbor, the loop analyzes 212 if the neighbor is already tentative. If there is not an existing tentative node for the neighbor, then a new tentative is added 218 to the tree. If there is a tentative node for the neighbor, then the cost of the known path to the neighbor (as represented by the tentative node) is compared 214 to the cost of the new path to the neighbor. If the new path is shorter, the existing tentative is deleted 216, and a new tentative is added 218 to the tree. If the new path is longer, then the new path is ignored by skipping to step 228. A tie-breaking mechanism is used to choose between paths having equal or near equal cost. (See the discussion of FIG. 5C below for a review of the considerations involved in tiebreakers).

When a node for the new neighbor is added 218 to the tree, the ER field for the new node is set to "?". This is because all of the nodes added to the routing tree in FIG. 5B are the immediate neighbors of the router generating the tree. For these nodes, the ER field will always be set to "?" because immediate neighbors are the only routers along the path to themselves, and thus must be the "last capable router along the path". That is, there is never a need to encapsulate user data packets which are destined to an immediately neighboring router, because the end system to which the user data packet is direct must be capable in the user data packet's protocol, and the neighbor must be capable in the protocol of the end system because the end system is connected to it.

After the ER field is set to "?", the IP and OSI flags are set for future use. If 220 the neighbor supports OSI, the OSI flag is set 221, otherwise the OSI flag is cleared 222. If 224 the neighbor supports IP, the IP flag is set 225, otherwise the IP flag is cleared 226.

After the node is created and the ER field and IP flag are initialized (or after it is determined in step 214 that a known path to the neighbor is shorter than the new path to the neighbor), the algorithm of FIG. 5B checks 228 whether there are additional neighbors that may be added to the tree. If not, the algorithm ends 229, otherwise, the algorithm moves 230 to the next neighbor and returns to step 212.

During each loop of the Dijkstra algorithm, the neighbors of a newly permanent node are added as tentative nodes below the newly permanent node (step 186, FIG. 4A). In the dual-protocol routers of FIG. 3, step 186 also involves manipulating the ER field and the IP flag.

Referring to FIG. 5C, an algorithm for adding new tentative nodes forms a loop similar to that of FIG. 5B. The loop is initialized 240 at the first neighbor. For each neighbor, the loop checks 242 if the neighbor is already permanent on the routing tree. If the neighbor is already permanent on the routing tree, then the shortest path to the neighbor has already been located, and the new path is ignored by skipping to step 264. (Note that the algorithm of FIG. 5B does not check if nodes are already permanent because none could be permanent at that point in the Dijkstra algorithm).

However, if the neighbor is not already permanent, the loop checks 244 if a tentative node for the neighbor already exists. As in FIG. 5B, if no tentative node for the neighbor exists, a new tentative node for the neighbor is added 250. If a tentative node for the neighbor already exists, the loop checks 246 if the new path to the neighbor has a lower cost than the known path to the neighbor. If the known path is shorter, then the new path is ignored by skipping to step 264. Otherwise, the existing tentative is removed 248, and a new tentative is added 250.

A tiebreaking mechanism is used in step 246 to choose between paths of equal or near equal cost. However, this tiebreaking mechanism must be carefully selected, to avoid the possibility that routers between the source of an encapsulated user data packet and the destination of the user data packet disagree over the outcome of the tiebreak, and thus route the user data packet in a manner not anticipated by the encapsulating router. If a disagreement occurs, user data packets may accumulate more than one encapsulation header on the way to their destination.

For example, referring to FIG. 3, consider an OSI user data packet that is to be transmitted from OSI-only router 155 to OSI-only router 157. The user data packet is forwarded unencapsulated to dual-protocol router 162, at which point it must be encapsulated to travel through IP-only router 163. There are two paths from IP-only router 163 to dual-protocol router 164. The first path travels through IP-only routers 159 and 158, and the second path through dual-protocol router 174 and OSI-only router 160. Assume that these paths are very close in cost, and that dual-protocol router 162 has performed a tiebreaker to determine that the path through IP-only routers 158 and 159 should be used.

As a result, dual-protocol router 162 generates an IP encapsulation header addressed to dual-protocol router (assume that the shortest path to OSI end system 157 passes through IP-only router 168, and thus dual-protocol router 170 is the last IP-capable router on the path to router 156). This IP header is prepended to the OSI user data packet, and the user data packet is forwarded to IP-only router 163.

On receipt of the IP user data packet addressed to router 170 (i.e., the encapsulated OSI user data packet), IP-only router 163 consults its routing tree. Assume that IP-only router 163 has also performed a tiebreaker to determine which path to use to forward user data packets toward dual-protocol router 164. However, IP-only router 163 has decided that the path through dual-protocol router 174 and OSI-only router 160 should be used. Therefore, on receipt of the IP user data packet addressed to router 170, IP-only router 163 forwards the user data packet to dual-protocol router 174.

Dual-protocol router 174, on receipt of the IP user data packet addressed to router 170, also consults its routing tree. Its routing tree indicates that the best path to router 170 is through OSI-only router 160, dual-protocol router 164, and IP-only router 168. Therefore, not realizing that the received IP user data packet is in fact an encapsulated OSI user data packet, dual-protocol router places an OSI encapsulation header on top of the IP encapsulation header already present on the user data packet. This second encapsulation header is addressed to router 164 because router 164 is the last OSI capable router along the path to router 170. The doubly-encapsulated user data packet is then forwarded to router 160.

OSI-only router 160 then forwards the doubly-encapsulated user data packet to dual-protocol router 164. Dual-protocol router 164, recognizing the user data packet as addressed to its dummy encapsulation address, removes the outer OSI header, and analyzes the inner encapsulation header, which is addressed to router 170. Dual-protocol router 164 then consults its routing tree, and determines that the best route to dual-protocol router 170 is through IP-only router 168. As the user data packet is already formatted in IP, it is forwarded without encapsulation to router 168. Router 168 then forwards the user data packet to router 170, which removes the IP encapsulation header and forwards the OSI user data packet to router 156 and from there to OSI end system 157.

The difficulty with the above scenario is that the OSI user data packet accumulated two encapsulation headers in the course of its transmission. This increases the overhead at the dual-protocol routers, and also increases the load on the links because the user data packet size is larger. The extra headers were caused by the fact that two of the routers in the network (in particular, routers 162 and 163) disagreed on how to break a tie between equal cost paths. Thus, to prevent extra headers, the tiebreaking algorithm must be consistent, i.e., all of the routers in the network must agree on which path wins a tie. (See the discussion of FIG. 6 below for a description of other ways that tiebreak algorithms that are not consistent can also be used.)

One consistent tiebreaking mechanism breaks ties based on the names of the routers involved in the paths. For example, consider the case where two paths through the network join at router A, and the two paths have equal (or near equal) cost. On the first path, the router just before router A is router B, and on the second path, the router just before router A is router C. This tie is broken by comparing the names of routers B and C, for example, the tie can be broken in favor of the path passing through router B because B is alphabetically prior to C. (Router names may be interpreted as multi-bit characters or as binary strings; in either case the names are unique and can be used to break ties.)

In the algorithm of FIG. 5C, when two paths join, one of the paths is chosen at step 246. For example, consider the following sequence of events: first, a node representing router C is made permanent on the routing tree, and a node representing router A is then added as a tentative node below the node representing router C. Later, a node representing router B is made permanent on the routing tree, and the algorithm of FIG. 5C attempts to add a node representing router A below the node representing router B. The algorithm would then notice (at step 244) that there is already a tentative node for router A, and would compare (at step 246) the cost of the known path to the existing tentative (i.e., the path through C) to the cost of the new path (i.e., the path through B). Noting that these paths are equal (or nearly equal), the algorithm then breaks the tie by comparing the names of two parents. Because the parent of the existing tentative is router C, whereas the parent of the new tentative would be router B, the algorithm breaks the tie in favor of the new path, and continues to step 248.

Note that all of the routers in the network will break ties in the same way if the above mechanism is used, and therefore, assuming that the LSPs are reliably propagated to all routers, all of the routing trees will be consistent. Also, note that this mechanism cannot be used in FIG. 5B because, in FIG. 5B, all of the routers added directly neighbor the router generating the routing tree, and thus the "parents" of the two tentative nodes are the same. However, the tiebreaking mechanism used in FIG. 5B may be selected at will (for example, it may be random) because there are no routers between the source and destination that could disagree over the outcome of the FIG. 5B tiebreakers.

Returning to FIG. 5C, when a new tentative node is added 250, the ER field, IP and OSI flags are initialized to their values in the parent of the new tentative node.

This is done so later comparisons can determine whether the "last capable router along the path" has been located. First, the loop checks 252 if the ER field is set to "?". If not, then the "last capable router along the path" has already been located, and there is no need for further analysis, so the loop skips to step 262. However, if the ER field is set to "?", then the loop analyzes if the "last capable router along the path" has now been located.

To determine if the "last capable router along the path" has been located, the loop first checks 254 if the IP flag is set. If the IP flag is set and the ER field is "?", it indicates that all of the routers along the path from the router generating the tree to the router which is currently being added to the tree are IP-capable routers. This continuous chain of IP-capable routers will be broken only if the router currently being added to the tree is not IP-capable. Therefore, if the IP flag is set, the loop checks 256 if the neighbor currently being added to the tree is IP-capable—if it is, then the chain of IP-capable routers remains unbroken, and the loop skips to step 262. However, if the router currently being added to the tree does not support IP, the chain of IP-capable routers has been broken, and the "last capable router along the path" is the parent of the new tentative node (i.e., the node most recently made permanent in step 184, FIG. 4A). Therefore, the ER field is set 258 to the name of the parent of the new tentative node.

Similarly, if the IP flag is cleared, then the OSI flag must be set (because the IP flag will only be cleared if the first router along the path was OSI-only). If the IP flag is cleared and the ER field is "?", it indicates that all of the routers along the path from the router generating the tree to the router which is currently being added to the tree are OSI-capable routers. This continuous chain of OSI-capable routers will be broken only if the router currently being added to the tree is not OSI-capable. Therefore, if the IP flag is cleared, the loop checks 260 if the neighbor currently being added to the tree is OSI-capable—if it is, then the chain of OSI-capable routers remains unbroken, and the loop skips to step 262. However, if the router currently being added to the tree does not support OSI, the chain of OSI-capable routers has been broken, and the "last capable router along the path" is the parent of the new tentative node (i.e., the node most recently made permanent in step 184, FIG. 4A). Therefore, the ER field is set 258 to the name of the parent of the new tentative node.

Note that, if the first router along the path is a dual-protocol router, no encapsulation is necessary, and both the OSI and IP flags will be set. However, the above algorithm will treat this case as if the first router along the path were IP-only; i.e., the algorithm will set the ER field to the last IP-capable router on the path. However, as encapsulation will not be done, this value will never be used, and it is merely an artifact of the algorithm's implementation. Alternative algorithms could avoid determining a value for ER when the first router on the path is dual-protocol. For example, a "dual" flag, indicating whether the first router on the path was dual-protocol, could be added to the node data structure—this flag could then be checked before any other tests were made, and the tests skipped if the flag is set. The same function could be achieved without storing an additional flag by simply checking if both the IP and OSI flags are set, and skipping the rest of the operations if they are. However, these alternative algorithms involve data storage and true/false comparisons not required by the algorithm presented.

After the ER field is modified as appropriate, a new tentative node for the neighbor is added 262 to the routing tree. Then (or after it is determined in either steps 242 or 246 that a better path to the neighbor is already known) the algorithm checks 264 whether there are additional neighbors that may be added to the tree. If not, the algorithm ends 265, otherwise, the algorithm moves 266 to the next neighbor and returns to step 242.

The above algorithms are used by the dual-protocol routers of FIG. 3 to generate their routing trees. The single protocol routers of FIG. 3 do not encapsulate user data packets, and thus do not include the fields 200 (FIG. 5A) in their routing tree, and do not perform the more complex algorithms of FIGS. 5B and 5C to form their routing trees.

The single-protocol routers of FIG. 3 receive and forward user data packets in accordance with the algorithms of FIG. 1B and 1C. The dual-protocol routers of FIG. 3 receive user data packets in accordance with the algorithm of FIG. 2B, and forward user data packets (step 126, FIG. 2B) in accordance with the algorithm of FIG. 6.

Referring to FIG. 6, when a user data packet is to be forwarded from a dual-protocol router, the router first determines 270 if the user data packet is OSI. If the user data packet is OSI, the OSI flag of the node in the routing tree which represents router A is checked 272. If the user data packet is not OSI, the IP flag of the node in the routing tree which represents router A is checked 274. If the respective flag is set at step 272 or step 274, then the first router along the path to router A is capable in the protocol of the user data packet, and that the user data packet is forwarded unencapsulated at step 276.

However, if the respective flag is not set at step 272 or 274, then the first router along the path to router A is not capable in the protocol of the user data packet. Therefore the user data packet must be encapsulated.

If the user data packet is OSI and router A's OSI flag is not set, then the algorithm checks 278 if router A's ER is set to "?". If ER is "?", then there is no encapsulation router (as discussed above), and the user data packet can be encapsulated directly to router A. Therefore, if ER is "?" an IP encapsulation header addressed to router A is generated 280. Otherwise, an IP encapsulation header addressed to the encapsulation router is generated 282.

Similarly, if the user data packet is IP and router A's IP flag is not set, then the algorithm checks 284 if router A's ER is set to "?". If ER is "?", then there is no encapsulation router (as discussed above), and the user data packet can be encapsulated directly to router A. Therefore, if ER is "?" an OSI encapsulation header addressed to router A is generated 280. Otherwise, an OSI encapsulation header addressed to the encapsulation router is generated 282.

After the encapsulation header is generated, the user data packet is encapsulated 290 in the header, and the encapsulated user data packet is forwarded on the next link to router A.

As described above, if an inconsistent tiebreaking mechanism (such as a random mechanism) is used in generating the network's routing trees, there is the possibility that user data packets gain multiple encapsulation headers as they are forwarded through the network.

One way to avoid this situation is to modify the forwarding algorithm of FIG. 6 so that, before user data packets are encapsulated, the algorithm checks whether the user data packet is already encapsulated. If there is already an encapsulation header on the user data packet, and the encapsulation header is incompatible with the next router's protocol, the user data packet can be made compatible by simply removing the previous encapsulating header, instead of adding a second header. In this way, multiple encapsulations can be avoided even where inconsistent tiebreaking mechanisms are used.

For this scheme to be feasible, there must be a mechanism for marking user data packets so that they can be recognized as encapsulated user data packets.

One way this can be done is to check if the user data packet is addressed to a "dummy address". (The dummy address appears in a predetermined position of the LSPs, and thus can be distinguished from real addresses.)

Another way this can be done is by marking all encapsulated user data packets so they may be immediately recognized as such. User data packets can be marked in several ways, depending on the encapsulating protocol. OSI user data packets can be marked by reserving one of the possible values of the 8473 select field ("SEL"). When the SEL field is set to this reserved value, it is an indication that the OSI header is encapsulating another user data packet. TCP/IP user data packets can be marked by reserving one of the possible values of the IP "User Protocol" field, which appears in an IP user data packet header. Again, when the User Protocol field is set to this reserved value, it is an indication that the IP header is encapsulating another user data packet.

The routing trees in the network may disagree even if a consistent tiebreaking algorithm is used. For example, the network topology (particularly the location or cost of links) may change, and while the changes are being propagated (via new LSPs) to all of the routers in the network, the routing trees of the various routers will be inconsistent. As a result, user data packets may temporarily follow loops in the network, and may accumulate extra encapsulation headers. Temporary packet looping is a well-known side effect of topological changes, and will occur regardless of whether the network uses dual protocols. However, the accumulation of extra encapsulation headers is an effect peculiar to dual-protocol networks, and can be avoided by the modification of the FIG. 6 algorithm described above.

The above embodiments, which support two protocols, may be enhanced to support three or more protocols. Because of their topological restrictions and resulting simplicity, it is most straightforward to add a third protocol (which will be referred to as "P3") to the per-area embodiments of FIGS. 1A and 2A. However, it is more complex to add a third protocol to the less restricted embodiment of FIG. 3.

To adapt the FIG. 1A embodiment for a third protocol, areas routing in the third protocol may simply be added to the network, and the Level-2 backbone reconfigured to support all three protocols (if routing in the third protocol is not allowed outside of the local areas, this latter step is not necessary). Note that, with three protocols, there are four possible types of multi-protocol routers: OSI-IP dual-protocol routers, OSI-P3 dual-protocol routers, IP-P3 dual-protocol routers, and OSI-IP-P3 triple-protocol routers. However, as in the two protocol embodiment illustrated in FIG. 1A, all routers in a given area must have the same capabilities—thus an area having an OSI-P3 dual-protocol router cannot also have an IP-P3 dual-protocol router, because their capabilities are not the same, even though both are dual-protocol.

Subject to the above, a third protocol may be easily added to the FIG. 1A architecture. Encapsulation is not required, for the same reasons as set forth in the discussion of FIG. 1A, and thus all of the multi-protocol routers are "simple". The algorithms described by the flow charts of FIGS. 1B and 1C are implemented by each of the routers in the network to receive and forward user data packets.

The embodiment illustrated in FIG. 2A may also be adapted for a third protocol. In FIG. 2A, all routers in a local area must be capable in at least one common protocol. Thus, the FIG. 2A network may be adapted for a third protocol by adding IP-P3 dual-protocol routers to the existing IP areas (e.g., area 139), by adding OSI-P3 dual-protocol routers to the existing OSI areas (e.g., area and/or by adding new P3 areas, which may have P3-only routers and/or any other P3-capable routers. In either embodiment, the Level-2 backbone may support all three protocols, or may support only one or two of the protocols if the unsupported protocols is not allowed outside of the local areas. Furthermore, the Level-2 backbone may be a mixture of triple-protocol, dual-protocol and single-protocol routers, as long as the Level-2 routers can correctly receive user data packets from their local areas and forward them appropriately. Furthermore, as encapsulation is required in the FIG. 2A architecture, all multi-protocol routers are "enhanced".

In a three protocol embodiment of the FIG. 2A network, the single-protocol routers use the routing and forwarding algorithms of FIGS. 1B and 1C, and the multi-protocol routers (e.g., IP-OSI, IP-P3, OSI-P3, and IP-OSI-P3 routers) use the receiving and forwarding algorithms of FIGS. 2B and 2C. If the network has P3 areas (i.e., areas where all routers must support P3), then step 134 of FIG. 2C is modified to check if the area is a P3 area; subsequent additional steps check if the user data packet is a P3 user data packet (by operations analogous to those of steps 136 and 142) and, if not, encapsulate the user data packet in a P3 header (by operations analogous to those of steps 138 and 144).

The topological restrictions of the above three protocol embodiments simplify the receiving and forwarding algorithms. However, they also make the network less flexible. Therefore, in another embodiment, a three protocol network is formed with fewer topological restrictions. The legend of FIG. 7A illustrates the icons used to represent routers in these three protocol networks. Triangles are used to represent P3 routers and end systems. The icons for the 7 different router types are illustrated in the legend, namely: IP-only, OSI-only, P3-only, IP-OSI dual-protocol, P3-OSI dual-protocol, P3-IP dual-protocol, and P3-IP-OSI all-protocol.

In the embodiment illustrated in FIG. 3, routing was performed in a two protocol network having no topological restrictions (with the exception that two incompatible single-protocol routers could not be connected together, a restriction which applies to every multi-protocol network, and is enforced by implementing one of the link initialization processes as described above). However, the algorithms used to route in FIG. 3 do not support routing in three protocol networks. The reasons

are for this are clarified in the discussion of FIG. 7A below.

The embodiment of FIG. 7A is a three protocol network having no topological restrictions. Each router has a common protocol with its neighboring routers, and thus the links between the routers initialize. Routing in this network is quite complex.

Consider an IP user data packet which is to be routed from IP end system 300 to IP end system 302 via the network of routers 304, 306, and 308. The IP user data packet is first forwarded from end system 300 to router 304. Router 304 must then forward the user data packet to router 306. However, router 306 is not IP-capable. The only protocol supported by both routers 304 and 306 is P3, therefore, the user data packet must be encapsulated in a P3 header before it is forwarded to router 306.

The P3 encapsulation header must be addressed to a P3 address. However, only P3-capable routers, i.e., routers 304 and 306, have P3 addresses. Thus, the P3 encapsulation header must be addressed to either router 304 or router 306. Clearly, the header should not be addressed to router 304, as router 306 would then simply return the user data packet to router 304, and no progress will be made. The only other choice is to address the header to router 306.

Therefore, the P3 header is addressed to router 306, and the user data packet is forwarded to router 306. However, on receiving the encapsulated user data packet, router 306 has no information on how to forward the user data packet. When router 306 receives the user data packet, router 306 recognizes it as an encapsulated user data packet, because it is addressed to the dummy P3 address advertised by router 306. Therefore, router 306 removes the user data packet's header and attempts to interpret its contents. However, because router 306 does not support IP, it cannot interpret the encapsulated contents (i.e., the IP user data packet), and therefore has no idea how to forward them.

One possible solution to the above problem is to program all routers so that, if a user data packet is decapsulated and the contents are found to be unintelligible, the contents of are blindly forwarded on one of the links connected to the router, preferably on a link other than the one on which the user data packet was received. If router 306 used such a strategy, it would choose at random between the link to router 308, the link to router 310, and the link to router 312, and forward the contents. Using such a strategy, in the architecture of FIG. 7A, there is a 67% probability that the user data packet will arrive at the IP end system 302, because there is a 33% chance that it will be forwarded to router 308, which can deliver it to end system 302, and there is a 33% chance that it will be forwarded to router 312, which is IP-capable, and can interpret the user data packet and to forward it to router 308. However, there also is a 33% probability that the user data packet would be lost, because if the user data packet is forwarded to OSI-only router 310, it will not be recognized, and will be simply discarded.

Note that, depending on the information available from the common LSP format, router 306 may be able to determine that routers 308 and 312 support protocols not supported by router 306 (and conversely know that router 310 does not support any other protocols). Router 306 can make use of this type of information by choosing to send the unknown encapsulated user data packet to either router 308 or 312, and not to router 312. The logic for this is as follows: routers 308 and 310 at least understand a protocol not understood by router 306, therefore, they have a chance of understanding a user data packet which is unintelligible to router 306. However, router 310 does not understand any protocols not understood by router 306, and thus does not have any chance of understanding the user data packet. Using this type of elimination process, the reliability of a three-protocol networks having no topological restrictions can approach 100%.

The above mechanism is less useful for four- or more protocol networks. In a three-protocol network, there is only one protocol not known by the dual-protocol routers from the perspective of the dual routers, if another router supports an "unknown" protocol, this unknown protocol must be the same protocol as that of an unintelligible user data packet. However, this reasoning only applies to four- or more protocol networks if all of the multi-protocol routers support all but one of the protocols. For example, in a four-protocol network, a user data packet that is unintelligible to a triple-protocol router must be in the one protocol not supported by the triple-protocol router. However, a user data packet that is unintelligible to a dual-protocol router may be in one of two protocols.

Another mechanism for dealing with the user data packet forwarding problems of FIG. 7A is to program all routers so that, if a user data packet is decapsulated and its contents are unintelligible, the contents are blindly forwarded on all of the links connected to the router (except the link on which the user data packet was received). Using this strategy, there is a 100% probability that the user data packet will arrive at its destination. However, there is also a probability that the user data packet will arrive more than once. For example, if the IP user data packet was sent to router 308 and router 312, both would be able to interpret its contents, and both copies would eventually be forwarded to end system 302. Such multiplicity of user data packets is likely to be undesirable.

To avoid the above difficulties, the embodiment of FIG. 7B establishes one topological restriction: there must be at least one all-protocol router in each area. When a user data packet needs to be encapsulated, it is addressed to one of the all-protocol routers in the area (the all-protocol router has an address in all of the protocols).

On receipt of an encapsulated user data packet, the all-protocol router removes the encapsulation header(s) from the user data packet, and determines which encapsulation headers should be added for the user data packet to arrive at its destination.

This procedure may best be explained by example. Consider an IP user data packet sent from IP end system 312 to IP end system 314. The user data packet is first forwarded from IP end system 312 to router 316. Router 316 uses its routing tree to determine that the user data packet should be forwarded to router 318. However, router 318 is not IP-capable. Therefore, router 316 encapsulates the user data packet in a P3 header (P3 being the common protocol between router 316 and router 318), addresses the header to all-protocol router 322, and forwards the encapsulated user data packet to router 318.

When it receives the user data packet, router 318 uses its routing tree to determine that the user data packet (which is addressed to the OSI address of all-protocol router 322) should be forwarded to router 320. How-

ever, router 320 is not P3-capable. Therefore, router 318 encapsulates the user data packet in an OSI header (OSI being the common protocol between router 318 and router 320), addresses the header to all-protocol router 322, and forwards the encapsulated user data packet to router 320.

Similarly, when router 320 receives the user data packet, it uses its routing tree to determine that the user data packet (which is addressed to the OSI address of all-protocol router 322) should be forwarded to router 322. Router 322 is OSI-capable, therefore, router 320 forwards the user data packet to router 322.

When router 322 receives the user data packet, it first interprets the OSI header, and determines that it is addressed to the dummy OSI address of router 322, i.e., to itself. Therefore, it removes the OSI header, and interprets the contents (which is a P3 header). It then determines that this header is similarly addressed to the dummy P3 address of router 322. Therefore, it removes the P3 header, and interprets the contents (which is the original IP user data packet).

The original IP user data packet is addressed to end system 314. Router 322 reviews its recently-received link state packets, and determines that to reach end system 314, the user data packet should be forwarded to router 328. Next, router 322 reviews the path that the user data packet will follow to router 328, and determines that OSI and P3 encapsulation headers need to be added to the user data packet. Therefore, router 322 generates an OSI encapsulation header, addressed to router 328, and adds this to the user data packet. Next, router 322 generates a P3 encapsulation header, addressed to router 326, and adds this to the user data packet. The doubly encapsulated user data packet is then forwarded to router 324.

When it receives the user data packet, router 324 uses its routing tree to determine that the user data packet (which is addressed to the P3 address of router 326) should be forwarded to router 326. Router 326 is P3-capable, therefore, router 324 forwards the user data packet to router 326.

When router 326 receives the user data packet, it first interprets the P3 header, and determines that it is addressed to the dummy P3 address of router 326, i.e., itself. Therefore, it removes the P3 header, and interprets the contents (which is an OSI header). Router 326 then uses its routing tree to determine that the user data packet (which is addressed to the OSI address of router 328) should be forwarded to router 328. Router 328 is OSI-capable, therefore, router 326 forwards the user data packet to router 328.

When router 328 receives the user data packet, it first interprets the OSI header, and determines that it is addressed to the dummy OSI address of router 328, i.e., to itself. Therefore, it removes the OSI header, and interprets the contents (which is the original IP user data packet). Router 328 then determines that the IP user data packet is addressed to end system 314, which is attached to router 328. Therefore, router 328 delivers the user data packet to end system 314.

To implement the above strategy, the information stored by the routers is somewhat different than that stored by the routers of FIG. 3. In particular, referring to FIG. 8A, the additional fields 200 in the nodes of the routing trees in the dual-protocol routers are somewhat different from those of FIG. 4B. In particular, each node includes an OSI flag 330, an IP flag 332, and a P3 flag 334.

If the OSI flag 330 is set in a particular node, it indicates that there is an unbroken chain of OSI-capable routers between the router storing the routing tree and the router represented by the node. If the OSI flag is cleared, this is not the case. The IP flag 332 and P3 flag 334 indicate similar conditions relative to IP-capable and P3-capable routers.

These flags are set or cleared as appropriate as the routing tree is generated. In step 182 of FIG. 4A, the neighbors of the router forming the routing tree are placed below the root node. Referring to FIG. 8B, this operation is performed by a loop, which is initialized at the first neighbor (step 340) and sequences to each new neighbor (step 362) until all of the neighbors have been processed (step 360), at which point the loop terminates (step 364).

Each neighbor is checked to see if a tentative node for the neighbor already exists (step 342). If not, a new node is added by sequencing to step 348. If a tentative node already exists, then the cost of the existing path is compared to the new path (step 344). If the new path is shorter, the old tentative is deleted (step 346) and the new tentative is added by sequencing to step 348. If the new path is longer, the new path is ignored by sequencing to step 360.

As in FIG. 5B, any tiebreaking mechanism may be used in step 344, as all of the nodes added in FIG. 8B represent routers which are the immediate neighbors of the router that is generating the routing tree.

Three tests are made in the main portion of the loop of FIG. 8B. First, if (step 348) the new neighbor supports OSI, the OSI flag in the neighbor's node is set (step 350), otherwise it is cleared (step 349). Second, if (step 352) the new neighbor supports IP, the IP flag in the neighbor's node is set (step 354), otherwise it is cleared (step 353). Third, if (step 356) the new neighbor supports P3, the P3 flag in the neighbor's node is set (step 358), otherwise it is cleared (step 357).

Thus, each neighbor of the router generating the routing tree is added to the routing tree with an indication of the protocols it supports.

Referring to FIG. 5C, as additional neighbors are added to the routing tree in step 186 of FIG. 5C, the OSI, IP, and P3 flags are updated. The operations of step 186 form a loop, which is initialized at the first neighbor (step 370) and sequences to each new neighbor (step 406) until all of the neighbors have been processed (step 402), at which point the loop terminates (step 404).

For each neighbor, the algorithm determines whether a permanent node for the neighbor already exists in the routing tree (step 372); if so, the neighbor is skipped by sequencing to step 402. Otherwise, the algorithm determines whether a tentative node for the neighbor already exists (step 374). If not, a new node is added by sequencing to step 380. If a tentative node already exists, then the cost of the existing path is compared to the new path (step 376). If the new path is shorter, the old tentative is deleted (step 378) and the new tentative is added by sequencing to step 380. If the new path is longer, the new path is ignored by sequencing to step 402.

As in FIG. 5C, a consistent tiebreaking mechanism must be used in step 376.

The flags in the new node for the neighbor are initially set to the values in the node's parent (step 380). However, the values of the flags are adjusted by three tests. In the first test, the algorithm checks if the OSI flag is set (step 382). If the OSI flag is set, it indicates that all of the routers on the path from the router gener-

ating the tree to the new neighbor are OSI-capable. In this case, the algorithm determines whether the chain of OSI-capable routers ends at the new neighbor, by determining whether the new neighbor supports OSI (step 384). If the new neighbor does not support OSI, the chain has broken, and the OSI flag is cleared (step 386). Otherwise, the chain remains intact, and the algorithm performs similar operations to determine the settings of the IP flag (steps 388, 390, 392) and the P3 flag (steps 394, 396, 398).

As a result of the operations described above, the OSI, IP, and P3 flags in each node of the routing tree indicate whether all of the routers leading to that node support OSI, IP, and P3, respectively. This information is used by the routers when forwarding user data packets, as illustrated in FIG. 9. (Note that algorithm of FIG. 9 includes tests which relate to all three of the protocols. However, routers which do not support one of the protocols will never forward user data packets in the unsupported protocol, and thus may eliminate the tests and results that relate to the protocol they do not support.)

When a user data packet is to be forwarded toward a router A (step 126, FIG. 2B), the dual-protocol routers of FIG. 7B first check the user data packet type. If (step 408) the user data packet is OSI, the algorithm checks if the OSI flag of the node for router A is set (step 410). That is, it checks if all of the routers on the path to router A support OSI. If this is the case, the user data packet does not need to be encapsulated, and it may be sent directly on the next link to router A (step 412). However, if the OSI flag is not set, it indicates that the user data packet will have to be encapsulated. Therefore, the algorithm proceeds to send the user data packet to an all-protocol router, by sequencing to step 424.

If the user data packet is not OSI, similar steps are performed to determine whether the user data packet is IP (step 414), and if so, whether (step 416) it is to be encapsulated, or sent directly (step 418). If the user data packet is not OSI or IP, the algorithm assumes that it is P3, and checks if (step 420) it is to be encapsulated, or sent directly (step 422).

As described above, user data packets that must be encapsulated are sent to an all-protocol router. To accomplish this, the user data packet is enclosed in an encapsulation header addressed to one of the all-protocol routers in the network. The encapsulation header must be included even if it is not necessary for protocol compatibility, because the encapsulation header effectively changes the address of the user data packet to that of the all-protocol router.

The protocol of the encapsulation header is chosen so that the encapsulated user data packet travels as far as possible before it needs to be encapsulated again. To accomplish this, the algorithm first retrieves the OSI, IP, and P3 flags of the desired all-protocol router (step 424). (In one embodiment, the closest all-protocol router is used; the closest all-protocol router may be located during the creation of the routing tree.)

If (step 426) the OSI flag of the desired all-protocol router is set (indicating that all of the routers on the path to the desired all-protocol router support OSI), an OSI encapsulation header is used (step 430). However, If the OSI flag is not set, then the IP flag is checked (step 430) If the IP flag is set (indicating that all of the routers on the path to the all-protocol router support IP), an IP encapsulation header is used (step 432). Finally, if (step 434) the P3 flag is set, a P3 encapsulation header is used (step 436).

If none of the flags of the node representing the desired all-protocol router are set, it indicates that no protocol is supported by all of the routers on the path to the all-protocol router. Thus, the user data packet will have to be encapsulated again before it arrives at the all-protocol router. To determine which protocol to use for the encapsulation header, the algorithm retrieves the OSI, IP, and P3 flags of the parent of the node representing the desired all-protocol router, i.e., the flags of the router which is one step closer along the path to the all-protocol router. If any of the flags of this node are set (steps 426, 430, 434), the user data packet is encapsulated in the appropriate protocol. If none of the flags are set, the algorithm steps to the next closer node (step 438). This process continues until a node with at least one flag set is located, at which point the encapsulation protocol is determined, and the user data packet is forwarded.

The data structure of FIG. 8A, and the algorithms of FIGS. 8B, 8C, and 9 are used by the dual-protocol routers of FIG. 7B in generating routing trees and forwarding user data packets. However, at least one all-protocol router in the local area must be capable of performing more complex operations. In particular, the all-protocol router must be capable of adding a sequence of encapsulation headers to a user data packet to allow the user data packet to travel to its destination. The enhancements of the above algorithms necessary to allow this functionality are described below.

In one embodiment, more than one all-protocol router perform the functions described below (i.e., receives user data packets from other routers and adds the appropriate encapsulation headers). In this embodiment, the dual-protocol routers forward user data packets to the nearest all-protocol router, as discussed above.

In another embodiment, some all-protocol routers may choose not to perform the functions described below; those all-protocol routers which do not perform the functions described below use the data structures and algorithms of FIGS. 8A through 9 to forward user data packets. In this embodiment, a notification mechanism (or an election, such as on a LAN) may be used to indicate which of the all-protocol routers are currently "on duty", i.e., providing the functions described below.

Referring to FIG. 10A, to provide additional functionality, the routing database of an "on duty" all-protocol router includes an additional field, and the OSI, IP, and P3 flags are used in a slightly different manner, as discussed below.

The additional field is the encapsulation string field (ES) 440, which stores a list of the protocols and router addresses to be used in encapsulation headers to be added to a user data packet before it is forwarded toward its destination. As illustrated by the above example, these headers are exactly those needed to allow the user data packet to reach its destination without further encapsulation. The algorithm described below attempts to locate the longest possible chains of routers which support one of the protocols, thereby minimizing the number of encapsulating headers to be added to the user data packet.

To obtain the encapsulation string, the algorithms which generate the routing tree are slightly modified. The modifications to the algorithm of FIG. 8B (i.e., the

algorithm of step 182, FIG. 4A, which places the neighbors of the router generating the tree on the tree) are illustrated in FIG. 10B. The flow control of the main loop, and the initialization of the OSI, IP, and P3 flags is identical to that of FIG. 8B, and will not be described again. The only modification is that the ES field is initialized to a null value for each node (step 442).

The algorithm of FIG. 8C (i.e., the algorithm of step 186, FIG. 4A, which places the neighbors of a newly permanent node on the tree) is also slightly modified, as illustrated in FIG. 10C. The flow control of the main loop, and tests performed on the OSI, IP, and P3 flags, are identical to those of FIG. 8C, and will not be described again. However, there are several modifications to the actions performed within the loop.

First, when a node is added to the tree, the values of the flags and of the encapsulation string field are copied from the parent node (step 444).

Second, when the algorithm discovers a break in a continuous chain of routers supporting a given protocol, additional operations are performed. For example, in step 382, the algorithm checks if the OSI flag is set, i.e., if there has been a continuous chain of OSI-capable routers thus far. If this is the case, the algorithm then checks (step 384) if the chain has not been broken, i.e., if the new neighbor supports OSI. If the chain is broken, the OSI flag is cleared (step 446). However, unlike step 386 of FIG. 8C, the algorithm also indicates the identity of the protocol of the broken chain, by setting a variable EP (encapsulation protocol) to the value OSI. This variable is used later in the loop, as discussed below. Similar operations are performed for the other protocols in steps 388, 390, and 448, and steps 394, 396, and 450.

Thus, when the algorithm reaches step 452, the flags have been appropriately modified. At this point, the algorithm checks (step 452) if any of the flags remain set, i.e., if there are any protocols supported by all of the routers thus far. If so, the algorithm proceeds to step 400, adding the node for the new neighbor to the tree.

However, if all of the flags are cleared, then the algorithm sequences to step 454. When the algorithm reaches step 454, the end of a continuous chain of routers which support a given protocol has been reached. In fact, because all of the flags have been cleared, the end of the longest such chain has been reached. The last router in this chain was the router immediately prior to the router whose node is currently being added to the tree, i.e., the parent node on the tree. Furthermore, the protocol of the chain was the protocol of the last flag to be cleared, which is stored in the EP variable.

The goal of the FIG. 10C algorithm is to store in the encapsulation string the list of protocols and addresses which collectively represents the longest chain of routers which lead to the destination. Thus, once the end of such a chain is reached, the name of the last router in the chain, and the protocol of the chain, are stored in the encapsulation string. Therefore, in step 454, the value of the encapsulation protocol variable, and the name of the parent of the node currently being added, are added to the end of the encapsulation string.

At this point, the algorithm begins to search for a second such chain of routers. This is done by setting all of the flags, and returning to step 382. The algorithm then passes through steps 382 through 452 a second time for the same neighbor. When the algorithm reaches step 452 the second time, the flags for the protocols not supported by the new neighbor will be cleared—in this way, the algorithm has re-initialized the flags to search for the next continuous chain of protocols.

The algorithm used by all-protocol routers to forward received user data packets is illustrated in FIG. 11. This operation primarily involves generating encapsulation headers having the protocols and addresses specified in the encapsulation string (step 478), and forwarding the thus encapsulated user data packet to the first router on the route to its destination (step 480).

However, before the headers from the encapsulation string are added, the algorithm determines if an innermost encapsulation header is required. To make this determination, the algorithm first determines if the user data packet is OSI (step 456) or IP (step 458); failing these tests, the user data packet is assumed to be P3.

If the user data packet is OSI, the OSI flag of the destination router is checked (step 468). As described above, the OSI flag of the destination router indicates whether all the routers after the last router specified in the encapsulation string support OSI. Therefore, if the user data packet is OSI, and the OSI flag is set, no encapsulation header is required, and the algorithm sequences directly to step 478.

Similarly, if the user data packet is IP, the IP flag of the destination router is checked (step 464). If the IP flag is set, no encapsulation is required, and the algorithm sequences directly to step 478. Also, if the user data packet is P3, the P3 flag of the destination router is checked (step 460). If the P3 flag is set, the algorithm sequences directly to step 478.

If the appropriate flag is not set in steps 460, 464, or 468, then an innermost encapsulation header is required. The protocol of this header may be OSI (in which case the algorithm sequences to step 472), IP (in which case the algorithm sequences to step 474), or P3 (in which case the algorithm sequences to step 476). To determine which header to use, the other flags are tested. For example, if the user data packet is OSI and the OSI flag is not set, the algorithm checks the IP flag (step 470). If the IP flag is set, then an IP header is used, otherwise, a P3 header is used. Similarly, if the user data packet is IP and the IP flag is not set, the algorithm checks the OSI flag (step 466). If the OSI flag is set, then an OSI header is used, otherwise, a P3 header is used. Finally, if the user data packet is P3 and the P3 flag is not set, the algorithm checks the OSI flag (step 462). If the OSI flag is set, then an OSI header is used, otherwise, an IP header is used.

Using the algorithms and data structures described in FIGS. 8A through 11, the dual-protocol routers can recognize the need for encapsulation, and forward user data packets that require encapsulation to an "on duty" all-protocol router; the "on duty" all-protocol router may then use the algorithms to correctly encapsulate user data packets so that they may be transmitted to their destinations.

It is important to note that, if the tiebreaking mechanism used by different routers is inconsistent, or if changes to LSPs are not propagated to all routers, routing loops may form. This effect also occurs in known networks under these conditions, but is more prevalent using the algorithms described above because user data packets to be encapsulated are always forwarded to the all-protocol router.

As an example, consider the case where the all-protocol router has placed a sequence of encapsulation headers on a user data packet to allow the user data packet to be forwarded along a particular path to its

destination. If one of the routers along the path disagrees on the path, then the user data packet may need to obtain an additional encapsulation header. In this case, the disagreeing router sends the user data packet back to the all-protocol router. The all-protocol router will then re-encapsulate the user data packet for the same path it previously encapsulated it for, and send it back toward the disagreeing router. In this way, the user data packet will loop to and from the all-protocol router and the disagreeing router, until the routers reach a consensus on the correct path, at which time the user data packet will reach its destination.

DETAILED DESCRIPTION

The following describes in detail one method for configuring a single network to run in the OSI and TCP/IP protocol suites. In this proposal, IP-only, OSI-only, and dual-protocol routers exist and can be mixed within a single area.

CONTROL PACKET FORMATS

As discussed above, where the invention is applied to routing in the TCP/IP and OSI protocols, the formats of the hello, LSP, and sequence number packets are normalized. In particular, the formats are substantially similar to the 10589 formats currently in use, but additional optional fields are added.

The approach described below does not change the way that OSI user data packets are handled. There are no changes to the contents or handling of ISO 8473 user data packets and error reports, nor to ISO 9542 Redirects, end system hello packets, and intermediate system hello packets. The approach described below does not change the way that OSI user data packets are handled. Specifically, there are no changes to the contents nor to the handling of ISO 8473 user data packets and error reports. Similarly, ISO 9542 Redirects and End System Hello packets are unchanged. ISO 9542 Intermediate System Hello packets do "double duty" and are used both for communication from routers to end systems, and for communications between routers on point-to-point links. The ISO 9542 Intermediate System Hello packets are modified by the addition of IP information in a way which OSI-only end systems will ignore. Other control packets (those associated with ISO 10589) remain unchanged except for the addition of IP information.

To allow easy parsing of the control packets, the optional fields are encoded in type-length-value (TLV) format. For example, the first octet indicates the type of the field, the second octet indicates the length of the field's value, and the third and following octets contain the value. The first two octets are read by a router before the contents of the field are interpreted. If the field type is recognized by the router, the contents of the fields are interpreted appropriately. However, if the field type is not recognized by a router, the entire field is ignored (by skipping the number of octets indicated in the length field). Regardless of whether the field was interpreted or ignored, if the control packet is forwarded to another router (as is the case with LSPs) it is passed on unchanged, i.e., including any ignored fields.

Hello Packets

Three additional fields are added to the 10589 hello packet format:

A "Protocols Supported" field identifies the protocols which are supported by the router generating the hello packet. This field must be included in all hello packets transmitted by IP-capable routers. If the field is omitted, it is assumed that the router transmitting the hello packet is OSI-only. The data protocols supported can be indicated by one-octet Network Layer Protocol Identifiers (NLPIDs) (as assigned by ISO/TR 9577).

An "IP Interface Address" field is included in all hello packets transmitted by IP-only and dual-protocol routers. This field contains the IP address of the interface on which the hello packet is transmitted. The address is formatted in four octets.

An "Authentication" field may be included to error-check the hello packet.

Link State Packets

Six additional fields are added to the 10589 link state packet format. A "Protocols Supported" field identifies the protocols which are supported by the router generating the LSP. This field must be included in all LSPs transmitted by IP-capable routers. If the field is omitted, it is assumed that the router transmitting the link state packet is OSI-only. The data protocols supported can be indicated by one-octet NLPIDs (as assigned by ISO/TR 9577, available from BSi standards at the above address).

One or more "IP Interface Address" field are included in all LSPs transmitted by IP-only and dual-protocol routers. These fields contain a list of one or more of the IP addresses of the interfaces of the router which originates the LSP. The address is formatted in four octets.

An "IP Internal Reachability Information" field may be included in LSPs transmitted by IP-capable routers. If present, this field identifies a list of zero or more entries of the format [metrics, IP address, subnet mask] which identify addresses reachable by the router which originated the LSP. Each entry must contain a default metric, and may contain delay, expense, and error metrics. If an IP-capable router does not directly reach any IP addresses, then it may omit this field, or may include the field with zero entries. In Level-1 LSPs, this field includes only entries directly reachable (via one of its interfaces) by the router originating the LSP. If included in a Level-2 LSP, this field includes entries reachable by the router which originates the LSP either via one of its interfaces, or indirectly via Level-1 routing. The metrics are preferably formatted as six bit fields, with the most significant bits used to indicate whether the metric is supported by the originating router. The IP address and subnet mask are both formatted as four octets.

An "IP External Reachability Information" field may be included in Level-2 LSPs transmitted by Level-2 IP-capable routers. If present, this field identifies a list of zero or more entries of the format [IP address, subnet mask, metrics] which identify addresses reachable, via a direct link to an external router, by the router which originated the LSP. Each entry must contain a default metric, and may contain delay, expense, and error metrics. If a Level-2 router does not have any external routes (via neighboring routers in other routing domains), it may omit this field, or may include the field with zero entries. The metrics are preferably formatted as six bit fields, with the most significant bits used to indicate whether the metric is supported by the originating router. The IP address and subnet mask are both formatted as four octets.

External routes may make use of "internal" or "external" metrics. Internal metrics are comparable with the metrics used for internal routes. Thus in choosing between and internal route and an external route using internal metrics, the metric values may be directly compared. In contrast, external metrics cannot be directly compared with internal metrics. Any route defined solely using internal metrics is always preferred to any route defined using external metrics, but when an external route using external metrics must be used, the lowest values of the external metric is preferred regardless of the internal cost to reach the appropriate exit point. (To identify whether a metric is internal or external, an extra bit may be added to the fields described in the previous paragraph. In one embodiment, this bit may be placed in one of the reserved bit positions of TCP/IP metric fields.)

It is useful, in the operation of external routing protocols, to provide a mechanism for border routers (i.e., routers in the same routing domain, which have the ability to route externally to other domains) to determine each other's existence, and to exchange external information (in a form understood only by the border gateways themselves). This is made possible by inclusion of an "Inter-Domain routing protocol Information" field, which may be present in Level-2 LSPs transmitted by Level-2 IP-capable routers. This field is transmitted for the convenience of the external routing protocol, and is not used by internal protocols.

An "Authentication" field may be included to error-check the hello packet.

Sequence Number packets

Sequence number packets (which list the names of the network's routers and the sequence numbers of the most recently received LSPs from those routers) are exchanged between routers to verify that all routers have the most recent version of the link state packets from the other routers.

An "Authentication" field may be included in these control packets for the purpose of error-checking.

Distinguishing Protocols

Dual-protocol routers may receive a combination of OSI control and user data packets and IP control and user data packets. It is necessary for the dual-protocol routers to be able to clearly and unambiguously distinguish the two protocol suites. This fact is complicated by the fact that the two protocols were developed independently, with no explicit effort to have the two protocol suites use different protocol type fields, or to otherwise distinguish the two.

There are several different cases which need to be handled separately. The first case is on LANs, or over other underlying sub-networks and links which provide a link level demultiplexing function (such as Ethernet, FDDI, all 802.x sub-networks, and the IETF point-to-point protocol PPP). In these cases, the underlying demultiplexing function is used. OSI and IP packets are encapsulated in underlying subnet headers without any intervening demultiplexing protocol.

On HDLC lines there is no link layer demultiplexing function. Similarly, on X.25 virtual circuits, to mix both OSI and TCP/IP packets on the same virtual circuit, there is no way to demultiplex. Therefore, in these cases the network level demultiplexing is used. The network layer protocol ID field, which comprises the first octet of 8473 packets, is used to perform the demultiplexing. OSI packets are encapsulated in link level packets without additional headers. IP packets are pre-appended with a single octet using a special value chosen from the ISO network protocol ID space, which indicates that the packet is IP. (When dual-protocol routers encapsulate IP user data packets in OSI user data packets, no additional IP demultiplexing field is necessary, as other mechanism are used to recognize encapsulated user data packets.)

Other Issues

Several other issues relating to an implementation of a dual-protocol network are discussed below.

Area Structure

A two level hierarchical approach is used to route OSI and IP traffic. Both protocols make use of a common area structure. Similarly, the amount of knowledge maintained by routers about specific IP destinations is as similar as possible to that for OSI. For example, IP-capable Level-1 routers are not expected to know the location of IP destinations outside of their area.

IP uses a very different addressing scheme from OSI. OSI addressing explicitly identifies the area. This makes it very easy for OSI Level-1 routers to identify user data packets going to destinations outside of their area, which need to be forwarded to Level-2 routers. However, IP addressing does not identify the area. For an IP Level-1 router to determine that given a particular IP destination address, a user data packet must be forwarded to a Level-2 router, the router must determine that the address is not reachable via any Level-1 router in its area.

Generally, the IP addresses that are reachable by a particular interface of an IP-capable router can be described by a single [IP address, subnet mask] pair. Here both the IP address and subnet mask are 32 bits long. Any addresses which match the specified address in the bits specified in the mask must be reachable directly via that interface, i.e., without going through other IP routers.

The OSI addressing scheme is used for OSI traffic, and the area structure is strongly linked to the OSI addresses, as in known OSI networks. In IP, there are several considerations. First, it is important to be flexible with the limited IP address space in order to cope with the anticipated growth of IP environments. Furthermore, dual-protocol routers may be installed in existing environments, which already have assigned IP and/or OSI addresses. Also, even if IP addresses are not already pre-assigned, the address limitations of IP will constrain what addresses may be assigned. Therefore, no specific relationship between IP addresses and the area structure is required.

As will be described below, greater efficiency and scaling of the routing algorithm can be achieved if there is some correspondence between the IP address assignment structure and the area structure.

To allow the area structure to be used for IP traffic, Level-2 routers include in their Level-2 LSPs a list of all [IP address, subnet mask] pairs reachable in their area. In general, this information may be determined from the Level-1 LSPs from all routers in the area.

Ignoring resource constraints, it would be permissible for a Level-2 router to simply duplicate all [IP address, subnet mask] entries from all Level-1 routers in its local area, for inclusion in its Level-2 LSP. However, in order for hierarchical routing to scale to large routing

domain sizes, it is highly desired to abbreviate the reachable address information.

This is accomplished by manual configuration of summary address entries. Each Level-2 router is configured with a set of [IP address, subnet mask] pairs for announcement in its Level-2 LSPs. In one embodiment, Level-2 routers are always required to announce these in Level-2 LSPs; alternatively, Level-2 routers may be allowed to leave out the summary addresses if there are no corresponding reachable Level-1 addresses in the area.

When the Level-2 LSP is generated, the set of reachable addresses for the Level-2 router's area (obtained from Level-1 LSPs from the area) is compared with the configured reachable addresses. Redundant information obtained from Level-1 LSPs is not included in Level-2 LSPs. Generally it is expected that the Level-2 configured information will include more inclusive addresses (corresponding to a subnet mask with less bits set to "1"). This will therefore allow one configured address-/subnet mask pair to hierarchically supersede the information corresponding to multiple entries in the Level-1 LSPs.

However, any addresses obtained from the Level-1 LSPs which is are not superseded by the manually configured information are included in the Level-2 LSPs. In this case, the metric values announced in the Level-2 LSPs are calculated from the sum of the metric values announced in the corresponding Level-1 LSP, plus the distance from the Level-2 router to the appropriate Level-1 router. If this sum results in a metric value greater than the maximum value that can be reported in Level-2 LSPs, then the maximum value is reported.

In some cases, a configured internal reachable address may announce reachability to an address which does not correspond to any Level-1 reachable address information. This may be normal during startup, but if it persists it represents an error condition. This error can only be rectified by human intervention; network management must be aware of the possibility of this error condition and correct it if it occurs.

The reachable IP addresses obtained by a Level-2 router from Level-1 LSPs include addresses that it can reach directly via its own interfaces (i.e., those it would report in its own Level-1 LSPs). Similarly, addresses of Level-1 routers (determined from the IP Interface Address field in the corresponding Level-1 LSPs) also represent addresses which are reachable via Level-1 routing in the area, and therefore need to be announced as Reachable address entries in the Level-2 LSPs.

In general, the same [IP address, subnet mask]entry may be announced in the Level-I LSPs sent by multiple Level-1 routers in the same area. In this case (assuming the entries are not superseded by a manually configured entry), then only one corresponding entry shall be included in the Level-2 LSP. The metric values announced in Level-2 LSPs correspond to the minimum of the metric value(s) that would be calculated for each of the Level-1 LSP entries.

There may be some IP addresses which match the manually configured summary address entries, but which are not reachable via Level-1 routing. If a Level-2 router receives an IP packet whose IP address matches a manually configured address which it is including in its Level-2 LSP, but which is not reachable via Level-1 routing in the area, then the packet must be discarded. In this case, an error report may be returned with the reason for discard specifying destination unreachable.

An example of the above scheme follows:

Referring to FIG. 12, assume that the network number for the entire TCP/IP domain 490 is "17" (a class A network). Also, assume each area is assigned a subnet number consisting of the next 8 bits. Each LAN in an area is assigned a subnet number consisting of 16 bits, giving each a 24 bit subnet mask. Finally 8 bits are left for the host field.

Now assume that area 492 (having subnet number "17.133") has a number of IP capable Level-1 routers 493, 494, 495 announcing (in the special IP entry in their Level-1 LSPs) sub-nets "17.133.5" (for 494), "17.133.43" (for 493) and "17.133.57" (for 495). In this example, in order to save space in its Level-2 LSPs, the Level-2 router 496 of area 492 would be configured to announce subnet 17.133. Only this one address needs to be announced in Level-2 LSPs. Thus any IP user data packet destined for an address in subnet "17.133.5", "17.133.43", or "17.133.57" will be passed to this area by the other Level-2 routers. The inclusion of "17.133" in Level-2 LSPs means that the four subnet addresses starting with "17.133" do not all have to be listed separately in Level-2 LSPs.

If there is any traffic for an unreachable address, for example the address "17.133.124.7", this traffic will also be routed to area 492, because all Level-2 routers in domain 490 will incorrectly believe that area 492 can handle it. (This may happen if management makes a gross configuration error, or a system tries to send traffic to an address which doesn't exist.) Area 492 will have to discard this traffic.

However, suppose that subnet number "17.133.125" was actually reachable via some other area (for example, area 497). In this case, the Level-2 router 498 in area 497 would be announcing (in its Level-2 LSPs according to information taken from its received Level-1 LSPs), reachability to subnet "17.133.125". Due to the use of "best match" routing, this works correctly. All traffic from other areas destined to subnet "17.133.125" would be sent to Level-2 router 498, and all other traffic to subnet "17.133" (i.e., traffic to any IP address starting with "17.133", but not starting with "17.133.125") would be sent to Level-2 router 496.

External Links

External links may be IP-only, or dual-protocol. In addition, in some cases it may be desirable to run encapsulated OSI over IP-only external links (because there may be a remote router which is ready to receive and decapsulate such OSI user data packets).

In general, which user data packets are sent in which manner over external links is highly dependent on both policy issues, and on the specific implementations in the remote external routers. It is therefore necessary to make use of considerable configuration information in controlling the use of such external links.

Encapsulated user data packets, if received at a Level-2 router to be forwarded outside of the domain, are first decapsulated. They are therefore treated just like unencapsulated IP user data packets which arrive at the same Level-2 router. If necessary, encapsulation of user data packets for transmission on links outside of the domain is independent of encapsulation of user data packets within the domain.

Currently there is no OSI standard for inter-domain routing. (However, it is expected that a protocol will be

defined.) At the present time, manual configuration is used. The link is statically configured with the set of address prefixes reachable via that link, and with the method by which they can be reached (such as the DTE address to be dialed to reach that address, or the fact that the DTE address should be extracted from the Initial Domain Part (IDP) of the ISO address). Therefore fixed (manually updated) tables are used for ISO inter-domain routing. This will be done only by Level-2 routers.

External links

IP will be run over external IP links only by Level-2 routers. This will make use normal IP external connectivity functions, for example as explained in RFC 1009 "Requirements for Internet Gateways".

There will be circumstances where it is necessary to interconnect different OSI domains using TCP/IP links. Architecturally this will be modelled as a normal external OSI connectivity, which just happens to use IP as the underlying subnet service. Regular external OSI links are manually configured. Therefore the same approach is used for TCP/IP links. No dynamic routing protocol will be run for the OSI link, and the availability of the remote OSI router will be assumed unless manually changed.

In order to reach the remote OSI router, it is necessary to know the IP address of the decapsulating router (i.e., of the OSI router which will remove the IP header). For this external link, it will therefore be necessary to manually configure the fact that this particular external OSI link makes use of an underlying IP service, and the IP address of the remote OSI router.

Note that, given the IP address to which the (encapsulated OSI) user data packet is to be sent, the resulting IP user data packet is forwarded externally just like any other IP user data packet. Thus, it will make use of the external IP information obtained as explained above.

There may be circumstances where it is necessary to interconnect different IP domains using external OSI links. Architecturally this will be modelled as a normal external IP connectivity, which just happens to use OSI as the underlying subnet service.

External IP reachability is determined by using EGP, RIP, BGP, or some other protocol. The same approach is used here. Part of the manual configuration of how to get to the remote IP gateway will include the underlying subnet address of the remote IP gateway. Since this is done in this case by encapsulating the user data packet in 8473, this address will simply be the OSI address.

Note that, given the OSI address to which the (encapsulated IP) user data packet is to be sent, the resulting 8473 user data packet is forwarded externally just like any other 8473 user data packet. Thus, it will make use of the external OSI information obtained as explained above.

TOS Routing

IP Type of Service (TOS) routing is provided through use of the Quality of Service (QOS) feature of 10589. Routing is done on the basis of throughput (the default metric), delay, expense, or residual error probability. Note that any particular user data packet may be routed on the basis of any one of these four metrics. Routing on the basis of general combinations of metrics is not supported.

The support for TOS is optional. If a particular user data packet calls for a specific TOS, and the correct path from the source to the destination is made up of routers all of which support that particular TOS, then the user data packet will be routed on the optimal path. However, if there is no path from the source to destination made up of routers which support the desired service, then the user data packet is forwarded using the default metric instead. This allows for QOS service in those environments where it is needed, while still providing acceptable service in the case where an unsupported QOS is requested.

The IP TOS field is mapped into the four QOS metrics as follows. The first two ("precedence") bits are not used to choose the route. The value of the last three bits (respectively, the "low delay", "throughput", and "reliability" bits) used to choose routes by mapping the following values to the indicated QOS metrics:

100: The QOS delay metric is used.

010: The default (throughput) QOS metric is used.

001: The QOS reliability metric is used.

All other combinations: The default metric is used.

Note: The IETF Router Requirements Working Group is considering assigning one of the unused bits in the TOS field in the IP header to correspond to minimum cost routing. In this case, the type of service for a packet with the minimum cost bit set to 1, and the other three TOS bits set to 0, will be mapped onto the cost metric of 10589.

Order of Preference of Routes

In order to ensure correct interoperation of routers, it is necessary to clearly and unambiguously specify the order of preference of routes in the routing algorithm. In general, there are several ways in which this order may be specified. What is necessary is that all router implementations agree to use the same precise order. In the preferred embodiment, the following order is used:

Among routes in an area, if the specified destination address matches more than one [IP address, subnet mask] pair, then the more specific address (i.e., the one with more bits in the mask set to "1") is preferred.

Among routes in an area to equally specific address matches, routes on which the requested TOS (if any) is supported are always preferred to routes on which the requested TOS is not supported.

Among routes in the area of the same TOS to equally specific addresses, the shortest route is preferred. For determination of the shortest path, if a route on which the specified TOS is supported is available, then the specified TOS metric is used, otherwise the default metric is used. Among routes of equal cost, loadsplitting may be performed as specified in 10589.

If a given destination is reachable within an area, then the path within the area is preferred to the path outside of the area (via Level-2 routing). If a given destination is not reachable, then the user data packet is routed to a Level-2 router using the following criteria.

Among routes to Level-2 routers, routes on which the requested TOS (if any) is supported are preferred to routes on which the requested TOS is not supported.

Among routes to Level-2 routers having the same TOS, the shortest routes are preferred. For determination of the shortest path, if a route on which the specified TOS is supported is available, then the specified TOS metric is used; otherwise, the default metric is used. Among routes of equal cost, loadsplitting may be performed as specified in 10589.

In Level-2 routers, routes via Level-1 routing are always preferred to routes via Level-2 routing. Level-2 routes are chosen under the following criteria.

Routes using internal metrics are always preferred to routes using external metrics.

Among internal routes, the route with the most specific [IP address, subnet mask] pair is preferred. Among routes with equally specific address matches, routes on which the requested TOS (if any) are preferred. Among routes of the same TOS to equally specific addresses, the shortest route is preferred. The determination of the shortest path uses the requested TOS first (as described above), with loadsplitting among equal cost routes in accordance with 10589. (Note that internal routes and external routes are treated identically for the purposes of the Dijkstra calculation).

Among external routes, the route with the most specific matching [IP address, subnet mask] pair is preferred. (Note that, for external routes, the subnet mask will normally correspond precisely to the network number. This implies that there will always be equal length matching strings. However, this test is included to allow future, more general handling of external addresses.) Among routes with equally specific address matches, routes on which the requested TOS (if any) are preferred. (External routes are considered to support the requested TOS only if the internal route to the appropriate border gateway supports the requested TOS, and the external route reported by the border gateway also supports the requested TOS.) Among routes of the same TOS to equally specific addresses, the shortest route is preferred. In determining the shortest path, routes with a smaller announced external metric are preferred, and among routes with equal external metrics, routes with a shorter internal metric are preferred. Among routes of equal cost, loadsplitting is performed in accordance with 10589.

Where Level-2 routers are announcing manually configured summary addresses in the Level-2 LSPs, there will be IP addresses which match the manually configured addresses, but which do not match any addresses which are actually reachable via Level-1 routing in the area. The above scheme will handle user data packets addressed to such addresses as follows. If the specified destination is reachable via Level-1 routing, then in accordance with the above criteria, the packet will be delivered via Level-1 routing. If the specified destination is not reachable via Level-1 routing, but is reachable via Level-2 routing, and thera are other Level-2 routers which offer more desireable routes according to the above rules (e.g., a route with a more specific match, or a route which has an equally specific match but supports the correct TOS), then the user data packet will be forwarded along the more desireable route. However, if the address cannot be reached via Level-1 routing, and the manually configured summary address is the most desireable Level-2 route, the address is unreachable and the user data packet must be discarded.

Encapsulation

IP user data packets are encapsulated directly in 8473 user data packets with a one-octet prepended NLPID value identifying that the encapsulated packet is IP. Inclusion of the NLPID allows encapsulated IP packets to be distinguished from encapsulated 8473 packets. The 8473 destination address indicates the address of the router to which the user data packet is being forwarded. The address to use (including SEL value) is included in the LSPs sent by dual-protocol routers.

The 8473 source address indicates the address of the router which does the encapsulation. When the destination 8473 address needs to be changed (such as when a Level-1 dual-protocol router encapsulates the user data packet with the address of a Level-2 router in its area, and the Level-2 router then needs to re-encapsulate with the address of a Level-2 router in the destination area), this is considered to be a new encapsulation and the source address is changed.

The SEL value used for the 8473 source address is left to each individual router. However, it is useful to use a different value from that to be used for the destination address. Use of a different SEL value here assures that, if an 8473 error report is returned to a router by a remote OSI router, such error reports will be very easy to distinguish from encapsulated IP user data packets.

An OSI user data packet is encapsulated directly in an IP user data packet with no intervening protocol. The IP destination address indicates the address of the router to which the user data packet is being forwarded. In this case, the user protocol field specifies that the contents of the data portion of the IP user data packet contains an encapsulated OSI user data packet. Here the IP source address indicates the address of the router which does the encapsulation.

Fragmentation, Segmentation, and Reassembly

Generally, data communications links have a "Maximum Transmission Unit" (MTU) which specifies the largest packet that can be transmitted on that link. In some cases, it may be necessary to transmit over a link a packet which is too large for that link (i.e., the length of the packet is larger than the MTU for that link). Many user data packet formats (including IP and 8473) provide for user data packets to be split into multiple packets for transmission over such links. DoD IP calls this function "fragmentation". OSI 8473 offers the same function, but calls it "segmentation". With either protocol, the destination for the user data packet is required to reassemble the packet before performing further processing of it.

When packets are encapsulated, we specify the address of a router as the destination in the encapsulating header. In one embodiment, the specified router is required to reassemble the encapsulated packet before performing decapsulation and forwarding the user data packet contained therein.

However, it is desirable (for performance and resource utilization reasons) to eliminate the need for reassembly of user data packets at routers. In another embodiment (applicable when only one encapsulating header will be required at any one time, such as the environment illustrated in FIG. 2A), the need for possible fragmentation or segmentation of the encapsulating packet is eliminated by performing fragmentation or segmentation of the original user data packet before encapsulation.

Where IP user data packets are to be encapsulated in OSI 8473 packets, manually configured information "MAX_IP_FRAGMENT_SIZE_IN_L1_AREA" for each area, and "MAX_IP_FRAGMENT_SIZE_IN_L2_BACKBONE" for the Level-2 backbone, specify the maximum IP fragment size which can be carried without requiring OSI segmentation. If an IP user data packet larger than this size is to be encapsulated, it must be fragmented before encapsulation,

and the resulting OSI 8473 packets must specify "don't segment".

Similarly, where OSI 8473 user data packets are to be encapsulated in IP packets. Manually configured information "MAX_OSI_SEGMENT_SIZE_IN_L-1_AREA" for each area, and "MAX_OSI_SEGMENT_SIZE_IN_L2_BACKBONE" for the Level-2 backbone, specify the maximum OSI segment size which can be carried without requiring IP fragmentation. If an OSI user data packet larger than this size is to be encapsulated, it must be segmented before encapsulation, and the resulting IP packets must specify "don't fragment".

OSI 8473 and ICMP Error Messages

Dual-protocol routers, when operating as IP gateways, must be able to send ICMP error messages (as specified in RFC 1009 "Requirements for Internet Gateways"; and RFC 792 "Internet Control Message Protocol", available from SRI International at the above address).

In addition, when a dual-protocol router encapsulates an IP user data packet in an 8473 header, it puts its own OSI address (including a special architectural constant SEL value) as the source address. In some error conditions, it therefore may receive an 8473 error report in response to the encapsulated user data packet. In some cases receipt of such 8473 error reports by an encapsulating router may require the sending of ICMP error reports, in other cases receipt of such 8473 error reports may signal the existence of an error condition, which is made available to network management. Similar situations occur where and OSI packet is encapsulated in and IP packet, and an ICMP error report is received.

Lifetime Control

For unencapsulated OSI 8473 and IP user data packets, lifetime is decremented exactly as if the user data packet was being forward by a pure OSI or pure IP router (respectively).

For IP user data packets which are encapsulated in OSI 8473 user data packets, we must consider the relationship between the lifetime field in each of the two packet headers.

The most straightforward approach would be to have the encapsulating router use an 8473 TTL value which is based on its notion of the "distance" to the decapsulating router. This would imply that the two lifetime fields are pretty much independent. However, there are several disadvantages of this approach: (i) this approach makes it possible for the encapsulated user data packet to loop for an extended period of time; (ii) it is not clear how the encapsulating router would pick the value; (iii) this approach defeats the intent that the lifetime field is measured in units of time (seconds or half seconds).

There is an alternate approach: In the encapsulation of an IP user data packet, the router will copy the value from the IP TTL field to the 8473 lifetime field. The field in the 8473 header will then be decremented by OSI routers in the normal manner. When the user data packet is decapsulated, the value of the lifetime field will be copied back from the 8473 header to the IP header. This approach allows the meaning and functionality of the lifetime field to be preserved between protocols.

Note that IP measures lifetime in seconds, where OSI measures lifetime in half seconds. In most cases, each hop will take less than a half second, and the same result (decrementing by one) will occur with either interpretation. In some cases, with encapsulated IP user data packets, this may result in over-decrementing (which is permissible), whereas with encapsulated OSI user data packets, this could result in underdecrementing by a factor of two. However, hops rarely take longer than a half second, this error condition is not likely to cause problems.

Error Reports

It is possible that, due to configuration errors, there may be times when a dual-protocol router forwards an IP packet to an OSI-only router, or forwards an OSI packet to an IP-only router. (Similarly, an IP-only router may forward an IP packet to an OSI-only router, or vice versa.) In this case, the packet must be discarded. An error report may be transmitted, in accordance with the IP or OSI 8473 specification (respectively). The reason for discard specified in the error report should specify "destination host unreachable" (for IP) or "destination unreachable" (for OSI). The number of packets discarded for this reason may be maintained as a network variable.

Congestion

One would like to avoid congesting dual-protocol routers as much as possible. For OSI packets, a congestion avoidance scheme is used to set the congestion indication bit in the network layer header when the router is "congested". The DoD TCP/IP protocol suite has, at this time, weaker congestion avoidance mechanisms (based on source quench, and packet dropping). This leads to the possibility that in the event of congestion, the ("better behaved") OSI end systems could suffer with respect to the ("less well behaved") TCP/IP end systems.

However, this is simply a special case of a more general problem: Since congestion avoidance is optional in 8473, even in a pure OSI system there will be some OSI end systems which implement congestion avoidance (and are therefore well behaved), and others which implement only congestion recovery (and are therefore moderately badly behaved), or implement no congestion backoff at all.

To harmonize these approaches, the following mechanism may be used: each router (i) may ignore the congestion avoidance mechanism altogether, and never set the congestion experienced bit; (ii) may provide a functional equivalent of dual queues, with those OSI packets making use of the congestion experienced field being queued separately from all other packets (i.e., OSI packets without this field, and all IP packets). Whether or not the congestion experienced bit is set on any particular packet depends only upon the level of traffic in the appropriate queue.

Brouters

Referring to FIG. 13, a so-called brouter 502 is a device which operates as both a bridge and a router. Suppose that two routers 504 and 506 are respectively connected to two Ethernets 508, 510, and brouter 502 is connected between the two Ethernets. Assume that the two routers implement the OSI (IS-IS) protocol and are both level 1 routers in the same domain. Also assume that brouter 502 is arranged to treat TCP/IP packets as if the brouter were a normal IP router and to treat all other packets as if the brouter were a bridge.

IS-IS hello packets sent by routers 504, 506 are received and forwarded by brouter 502 using normal bridge functions. Routers 504, 506 also receive each other's link state packets. Thus, it appears to brouter 502 that routers 504, 506 are exchanging OSI packets and they are forwarded using normal bridge functions. It appears to routers 504, that they are on the same local area network so they learn each other's 48-bit Ethernet addresses and exchange routing information.

Now suppose that brouter 504 receives an IP packet which it needs to forward via brouter 506. Because router 504 thinks that it and router 506 are on the same Ethernet, router 504 simply forwards the IP packet directly, using normal Ethernet encapsulation and using the 48-bit Ethernet address of router 506 in the Ethernet header. Then brouter 502, thinking as a bridge, would say "This is an IP packet. I don't forward this as a bridge". Then brouter 502, thinking as a router says, "This is an IP packet. I know how to forward IP packets. However, this packet is sent to an Ethernet address which is not me, thus I will ignore it." The result is that the IP packet does not get forwarded as it should.

One solution to this problem is to have the brouter follow these rules:

1. For packets which are not IP packets, act as a bridge (as before).

2. For IP packets sent to an Ethernet broadcast or multicast address, act as an IP router (as before).

3. For IP packets went to the brouter's Ethernet 48-bit address, act as an IP router (as before)

4. For IP packets sent to a single station 48-bit address which is not the brouter's address, act as a bridge (unlike before).

Other Embodiments

The invention may be practiced with protocol suites other than OSI and TCP/IP.

Other embodiments are within the scope of the following claims.

ANNEX A;DIJKSTRA CALCULATION AND FORWARDING

Annex C.2 of ISO 10589 specifies the SPF (Diksktra) algorithm for calculating routes with the IS-IS routing protocol. This annex specifies modifications to the SPF algorithm for supporting IP and dual routing, and specifies a compatible method for forwarding IP packets. This will result in a suitable order of preference of routes.

A.-1 SPF Algorithm for IP and Dual Use

This section specifies an SPF Algorithm for calculating routes with the IS-IS routing protocol, for support of both TCP/IP and OSI. This is based on an extention to the algorithm specified in annex C.2 of ISO 10589.

An algorithm invented by Dijkstra known as shortest path first (SPF) is used as the basis for the route calculation. It has a computational complexity of the square of the number of nodes, which can be decreased to the number of links in the domain times the log of the number of nodes for sparse networks (networks which are not highly connected).

A number of additional optimizations are possible:

1) If the routing metric is defined over a small finite field (as in this document), the factor of log n may be removed by using data structures which maintain a separate list of systems for each value of the metric rather than sorting the systems by logical distance.

2) Updates can be performed incrementally without requiring a complete recalculation. However, a full update must be done periodically to ensure recovery from data corruption, and studies suggest that with a very small number of link changes (perhaps 2) the expected computation complexity of the incremental update exceeds the complete recalculation. Thus, this annex specifies the algorithm only for the full update.

3) If only End System LSP information has changed, it is not necessary to re-compute the entire Dijkstra tree. If the proper data structures are used, End Systems (including IP reachability entries) may be attached and detached as leaves of the tree and their forwarding information base entries altered as appropriate.

The original SPF algorithm does not support load splitting over multiple paths. The algorithm in this annex does permit load splitting (in 2 protocol environments) by identifying a set of equal cost paths to each destination rather than a single least cost path.

A.1.1 Databases

PATHS—This represents an acyclic directed graph of shortest paths from the system S performing the calculation. It is stored as a set of triples of the form <N,d(N), {Adj(N)}>, where:

N is a system identifier. In the level 1 algorithm, N is a 6 octet ID for OSI end systems, a 7 octet ID for routers, or an 8 octet IP Internal Reachability Information entry. For a router which is not a pseudonode, it is the 6 octet system ID, with a 0 appended octet. For a pseudonode it is a true 7 octet quantity, comprised of the 6 octet Designated Intermediate System ID and the extra octet assigned by the Destinated Router. The IP Internal Reachability Information entries consist of a 4 octet IP address plus a 4 octet subnet mask.

In the level 2 algorithm, is either a 7 octet router or pseudonode ID (as in the level 1 algorithm); a variable length OSI address prefix; an 8 octet IP Internal Reachability Information Entry, or an 8 octet IP External Reachability Information entry. The variable length OSI address prefixes, and 8 octet IP Reachability Information entries will always be a leaf, i.e., "End System" in PATHS. As above, the IP Reachability Information entries consist of an [IP address, subnet mask] combination.

d(N; is N's distance from S (i.e., the total metric value from N to S).

{Adj(N)} is a set of valid adjacencies that S may use for forwarding to N.

When a system is placed on PATHS, the path(s) designated by its position in the graph is guaranteed to be a shortest path.

TENT—This is a list of triples of the form <N,d(N),{Adj(N)}>, where N, d(N), and ({Adj(N)} are as defined above for PATHS.

TENT can intuitively be thought of as a tentative placement of a system in PATHS. In other words, the triple <N,x,{A}> in TENT means that if N were placed in PATHS, d(N) would be x, but N cannot be placed on PATHS until is is guaranteed that no path shorter than x exists.

Similarly, the triple <N,x,{A,B}> in TENT means that if N were placed in PATHS, then d(N) would be x via either adjacency A or B.

Note: It is suggested that the implementation maintain the database TENT as a set of list of triples of the form <*,Dist,*>, sorted by distance Dist. In addition, it is necessary to be able to process those systems which

are pseudonodes before any non-pseudonodes at the same distance Dist.

The 8 octet system identifiers which specify IP reachability entries must always be distinguishable from other system identifiers. As specified above, two IP reachability entries which differ only in the subnet mask are still considered to be separate, and will therefore have distinct system identifiers N. The SPF algorithm will therefore calculate routes to each such entry, and the correct entry will be selected in the forwarding process.

A.1.2 Use of Metrics in the SPF Algorithm

Internal metrics are not comparable to external metrics. For external routes (routes to destinations outside of the routing domain), the cost d(N) of the path from N to S may include both internal and external metrics. d(N) may therefore be maintained as a two-dimensioned vector quantity (specifying internal and external metric values).

d(N) is initialized to [internal metric=0, external metric=0].

In incrementing d(N) by 1, if the internal metric value is less than the maximum value MaxPathMetric, then the internal metric value is incremented by one and the external metric value left unchanged; if the internal metric value is equal to the maximum value MaxPathMetric, then the internal metric value is set to 0 and the external metric value is incremented by 1. Note that this can be implemented in a straightforward manner by maintaining the external metric as the high order bits of the distance.

In the code of the algorithm below, the current path length is held in the variable "tentlength". This variable is a two-dimensional quantity tentlength=[internal metric, external metric], and is used for comparing the current path length with d(N) as described above. Tentlength is incremented in the same manner as d(N).

A.1.3 Overview of the Algorithm

The basic algorithm, which builds PATHS from scratch, starts out by putting the system doing the computation on PATHS (no shorter path to SELF can possibly exist). TENT is then pre-loaded from the local adjacency database.

Note that a system is not placed on PATHS unless no shorter path to that system exists. When a system N is placed on PATHS, the path to each neighbor N of N, through N, is examined, as the path to N plus the link from N to M. If <M,*,*> is in PATHS, this new path will be longer, and thus ignored.

If <M,*,*> is in TENT, and the new path is shorter, the old entry is removed from TENT and the new path is placed in TENT. If the new path is the same length as the one in TENT, then the set of potential adjacencies {Adj(M)} is set to the union of the old set (in TENT) and the new set {Adj(N)}. If M is not in TENT, then the path is added to TENT.

Next the algorithm finds the triple <N,x,{Adj(N)}> in TENT, with minimal x. Note: This is done efficiently because of the optimization described above. When the list of triples for distance Dist is exhausted, the algorithm then increments Dist until it finds a list with a triple of the form <*,Dist,*>.

N is placed in PATHS. We know that no path to N can be shorter than x at this point because all paths through systems already in PATHS have already been considered, and paths through systems in TENT still have to be greater than x because x is minimal in TENT.

When TENT is empty, PATHS is complete.

Note that external metrics can only occur in "IP External Reachability Information" entries, which correspond to a leaf (i.e., End System in PATHS). Any route utilizing an entry with an external metric will always be considered to be less desireable than any entry which uses an internal metric. This implies that in the addition of systems to PATHS, all systems reachable via internal routes are always added before any system reachable via external routes.

A.1.4 The Algorithm

The Decision Process Algorithm must be run once for each supported routing metric (i.e., for each supported Type of Service). A level 1 router runs the algorithm using the level 1 LSP database to compute level 1 paths (for those level 1 routers which are not level 2 routers, this includes the path to the nearest attached level 2 router). Level 2 routers also separately run the algorithm using the level 2 LSP database to compute level 2 paths. IP-capable level 2 routers must keep level 2 internal IP routes separate from level 2 external IP routes.

Note that this implies that routers which are both level 1 and level 2 routers, and which support all four routing metrics, must run the SPF algorithm 8 times (assuming partition repair is not implemented).

If this system is a Level 2 Router which supports the partition repair optional function the Decision Process algorithm for computing Level 1 paths must be run twice for the default metric. This first execution is done to determine which of the area's manualAreaAddresses are reachable in this partition, and to elect a Partition Designated Level 2 Router for the partition. The partition Designated Level 2 Router will determine if the area is partitioned and will create virtual Level 1 links to the other Partition Designated Level 2 Routers in the area in order to repair the Level 1 partition. This is further described in section 7.2.10 of 10589.

The SPF algorithm specified here will calculate routes for both OSI and IP. In particular, routes are calculated to all system identifiers N, where N may specify an OSI End System, the OSI address of a router, or an IP reachability entry (including the "IP interface address" advertised by IP-capable routers in their LSPs). In computing the forwarding database, it is an implementation specific issue whether the IP forwarding database is kept separately from the OSI forwarding database. Where appropriate, this annex will refer separately to entries in these two forwarding databases. This is not meant to preclude any specific implementation method.

OSI and IP use separate mechanisms to determine whether a packet is in the area (in particular, OSI makes use of area addresses, and IP determines that a destination is not in an area by looking in the level 1 forwarding data-base and determining that no entry exists for that destination within the area). The route to the nearest level 2 router will result in separate entries in the forwarding database for OSI and IP. For IP, the route to the nearest attached level 2 router may be entered in the forwarding database as a default route (i.e., a route with a subnet mask of all 0).

One approach would be to put the results of each Dijkstra algorithm in a separate forwarding database. For a router which supports both level 1 and level 2 routing (including level 2 internal and level 2 external routes), and which supports all four types of service, this would result in twelve separate forwarding databases for IP. Implementations may choose to minimize

the number of forwarding data-bases by combining the information from the multiple Dijkstra calculations into a single database per supported TOS. This is discussed in section A.2 below.

The SPF algorithm specified in section C.2.3 of 10589 is amended to appear as follows:

Step 0: Initialize TENT and PATHS to empty. Initialize tentlength to [internalmetric=0, externalmetric=0] (tentlength is the pathlength of elements in TENT that we are examining).

1) Add <SELF,O,W> to PATHS, where W is a special value indicating traffic to SELF is passed up to internal processes (rather than forwarded).
2) Now pre-load TENT with the local adjacency database (Each entry made to TENT must be marked as being either an End System or a router to enable the check at the end of Step 2 to be made correctly—Note that each local IP reachability entry is included as an adjacency, and is marked as being an End System). For each adjacency Adj(N) (including level 1 OSI Manual Adjacencies, or level 2 OSI enabled reachable addresses, and IP reachability entries) on enabled circuits, to system N of SELF in state "Up" compute:
   p(N)=cost of the parent circuit of the adjacency (N), obtained from $metric_k$, where k=one of {default metric, delay metric, monetary metric, error metric}
   Adj(N)=the adjacency number of the adjacency to N
3) If a triple <N,x,{Adj(M)}> is in TENT, then:
   If x=d(N;, then {Adj(M)}←{Adj(M)}∪{Adj(N)}.
4) If N is a router or an OSI End System entry, and there are now more adjacencies in maximumPath-Splits, then remove excess adjacencies as described in Clause 7.2.7 of 10589. If N is an IP Reachability Entry, then excess adjacencies may be removed as desired. This will not effect the correctness of routing, but may eliminate the determinism for IP routes (i.e., IP packets still follow optimal routes within an area, but where multiple equally good routes exist, will not necessarily follow precisely the route that any one particular router would have anticipated).
5) If x <d(N), do nothing.
6) If x>N,d(N), remove <N,x,{Adj(M)}> from TENT and add the triple <N,d(N),Adj(N)>.
7) If no triple >N,x,{Adj(M)}> is in TENT, then add <N,d(N),Adj(N)> to TENT.
8) Now add systems to which the local router does not have adjacencies, but which are mentioned in neighboring pseudonode LSPs. The adjacency for such systems is set to that of the designated router. Note that this does not include IP reachability entries from neighboring pseudonode LSPs. In particular, the pseudonode LSPs do not include IP reachability entries.
9) For all broadcast circuits in state "On", find the pseudonode LSP for that circuit (specifically, the LSP with number zero and with the first 7 octets of LSPID equal to L_nCircuitID for that circuit, where n is 1 (for level 1 routing) or 2 (level 2 routing)). If it is present, for all the neighbors reported in all the LSPs of this pseudonode which do not exist in TENT add an entry <N,d(N),Adj(N)> to TENT, where:
   d(N)=$metric_k$ of the circuit.
   Adj(N)=the adjacency number of the adjacency to the DR.
10) Go to Step 2.

Step 1: Examine the zeroeth link state PDU of P, the system just placed on PATHS (i.e., the LSP with the same first 7 octets of LSPID as P, and LSP number zero).

1) If this LSP is present, and the "Infinite Hippity Cost" bit is clear, then for each LSP of P (i.e., all LSPs with the same first 7 octets of LSPID and P, irrespective of the value of LSP number) compute:

$$dist(P,N)=d(P)+metric_k(P,N)$$

For each neighbor N (both End System and router) of the system P. If the "Infinite Hippity Cost" bit is set, only consider the End System neighbors of the system P. Note that the End Systems neighbors of the system P includes IP reachable address entries included in the LSPs from system P. Here, d(P) is the second element of the triple <P,d(P),{Adj(P)}>, and $metric_k$(P,N) is the cost of the link from P to N as reported in P's link state PDU.

2) If dist(P,N)>MaxPathMetric, then do nothing.
3) If <N,d(N),{Adj(N)}> is in PATHS, then do nothing. Note: d(N) must be less than dist(P,N), or else N would not have been put into PATHS. An additional sanity check may be done here to ensure that d(N) is in fact less than dist(P,N)
4) If a triple <N,x{Adj(N)}> is in TENT, then:
   a) If x=dist(P,N), then {Adj(N)}←(Adj(N)}∪{Adj(P)}.
   b) If N is a router or an OSI end system, and there are now more adjacencies in {Adj(N)} than maximumPath-Splits, then remove excess adjacencies, as described in clause 7.2.7 of 10589. For IP Reachability Entries, excess adjacencies may be removed as desired. This will not effect the correctness of routing, but may eliminate the determinism for IP routes (i.e., IP packets will still follow optimal routes within an area, but where multiple equally good routes exist, will not necessarily follow precisely the route that any one particular router would have anticipated).
   c) if x<dist(P,N), do nothing.
   d) if x dist(P,N), remove <N,x,Adj(N)> from TENT, and add <N,dist(P,N,{Adj(P)}>
5) if no triple <N,x,{Adj(N)}> is in TENT, then add <N,dist(P,N),{Adj(P)}> to TENT.

Step 2: If TENT is empty, stop. Else find the element <P,x,{Aj(P)}>, with minimal x as follows:

1) If an element <*,tentlength,*> remains in TENT in the list for tentlength, choose that element. If there are more than one elements in the list for tentlength, choose one of the elements (if any) for a system which is a pseudonode in preference to one for a non-pseudonode. If there are no more elements in the list for tentlength, increment tentlength and repeat Step 2.
2) Remove <P,tentlength,{Adj(P)}> from TENT.
3) Add <P,d(P),(Adj(P)}> to PATHS.
4) If this is the Level 2 Decision Process running, and the system just added to PATHS listed itself as Partition Designated Level 2 Intermediate system, then additionally add <AREA.P,d(P),{Adj(P)}> to PATHS, where AREA.P is the Network Entity Title of the other end of the Virtual Link, obtained by taking the first AREA listed in P's LSP and appending P's ID.
5) If the system just added to PATHS was an end system, go to step 2. Else go to Step 1.

NOTE—In the level 2 context, the "End Systems" are the set of Reachable Address Prefixes (for OSI), the set of Area Addresses with zero cost (again, for OSI), plus the set of IP reachability entries (including both internal and external).

A.2 Forwarding of IP packets

The SPF algorithm specified in section A.1 may be used to calculate (logically) separate IP forwarding tables for each type of service, and for level 1, level 2 internal, and level 2 external routes. Section A.2.1 describes how to forward IP packets, based on these multiple forwarding data-bases. Section A.2.2 describes how the multiple forwarding databases can be combined into a single forwarding database per supported TOS.

A.2.1 Basic Method for Forwarding IP packets

For level 1-only routers:

Determine if the IP destination address matches any entry in the level 1 forwarding table for the specified TOS.

Determine if the IP destination address matches any entry in the level 1 forwarding table for the default TOS.

If default TOS resulted in more specific entry, forward according to default TOS.

If equally specific entries found, forward according to specified TOS.

If no entry was found (which includes no default route entry), then destination is unreachable. Note: For level 1 only routers, the route to the nearest attached level 2 router will be entered into the forwarding database as a default route (i.e., a route with a subnet mask which is all 0). Thus this last event (no entry found) can occur only if there is no attached level 2 router reachable in the area.

For routers which are both level 1 and level 2 routers:

Determine if the IP destination address matches any entry in the level 1 forwarding table for the specified TOS.

Determine if the IP destination address matches any entry in the level 1 forwarding table for the default TOS.

If default TOS resulted in more specific entry (i.e., more bits in the subnet mask take the value 1), forward according to default TOS.

If equally specific entries found, forward according to specified TOS.

If no entry found:

Determine if the IP destination address matches any entry in the level 2 internal forwarding table for the specified TOS.

Determine if the IP destination address matches any entry in the level 2 internal forwarding table for the default TOS.

If default TOS resulted in more specific entry, forward according to default TOS.

If equally specific entries found, forward according to specified TOS.

If no entry found:

Determine if the IP destination address matches any entry in the level 2 external forwarding table for the specified TOS.

Determine if the IP destination address matches any entry in the level 2 external forwarding table for the default TOS.

If default TOS resulted in more specific entry, forward according to default TOS.

If equally specific entries found, forward according to specified TOS

If no entry is found, then destination is unreachable

For level 2-only routers, the above algorithm can be used, except since there is no level 1 forwarding database, the corresponding steps can be skipped.

As discussed earlier, for level 2 routers which are announcing manually configured summary addresses in their level 2 LSPs, in some cases there will exist IP addresses which match the manually configured addresses, but which do not match any addresses which are reachable via level 1 routing in the area. Packets to such addresses are handled according to the rules specified above. This may be accomplished by adding the manually configured [IP address, subnet mask] entry to the level 2 forwarding database (for the appropriate TOS), with a special "next hop" address which specifies that packets for which this entry is selected are to be discarded. This will work correctly because more desireable entries (such as delivering the packet via level 1 routing to the correct destination, or a more specific level 2 route) will automatically take precedence according to the forwarding rules specified above. Less desireable routes (such as using a level 2 external route to the "default route" entry) are not possible because other level 2 routers will believe the summary addresses advertised by this router.

A.2.2 Reduction of IP Forwarding Databases

The multiple forwarding databases used in the basic forwarding method in section A.2.1 can be reduced, by combining the multiple databases into one database for each supported TOS.

For reduction of IP forwarding databases, it is assumed that for any two overlapping address entries, either the entries are identical, or one range contains the other. In other words, for any two [IP address, subnet mask] entries A and B, if there is at least one IP address which matches both entries, then either: (i) the two entries are identical; or (ii) entry A contains entry B (i.e., any IP address which matches entry also matches entry A); or (iii) entry B contains entry A (any IP address which matches entry A also matches entry B).

Non-contiguous subnet masks can be configured to violate this assumption. For example, consider the two entries:

A=[address="01010101-00000101-00000000-00000000", mask="11111111-00001111-00000000-00000000"]

B=[address="01010101-01010000-00000000-00000000", mask="11111111-11110000-00000000-00000000"]

In this case neither entry contains the other. Specifically;

there are IP addresses which match both A and B (e.g., "01010101-01010101-xxxxxxxx-xxxxxxxx"),

there are IP addresses which match A but not B (e.g., "01010101-11110101-xxxxxxxx-xxxxxxxx")

there are IP addresses which match B but not A (e.g., "01010101-01011111-xxxxxxxx-xxxxxxxx").

The reduction of the multiple forwarding databases for each TOS to a single database for each TOS is based on the use of "best match" routing, combined with reduction of the entries placed in the forwarding database in order to eliminate entries which are not to be selected (based on the order of preference of routes specified above). The specific algorithm for creation of the IP forwarding database can be described as follows:

1) Make use of the the Dijkstra algorithm described in section A.1 to create separate forwarding databases for each supported TOS for level 1 routes, level 2 internal routes, and level 2 external routes. (Note that each entry in the forwarding database will specify an [IP address, subnet mask] combination, as well as the next hop router for IP packets which match that entry).

2) For each level 1 route entry which has been placed in the level 1 IP forwarding database for a specific TOS, copy that entry into the overall IP forwarding database for that TOS.

2) For each route entry X which has been placed in the level 2 internal IP forwarding database for a specific TOS, search for overlapping entries in the level 1 IP forwarding database for the specific TOS, and also for the default TOS:

a) If there is any overlapping entry Y in the level 1 forwarding database (for the specfic TOS, or for the default TOS) such that either (i) Y contains X; or (ii) Y is identically specific to X; then ignore entry X.

b) Otherwise, copy entry X into the overall IP forwarding database for the specific TOS.

4) For each route entry X which has been placed in the level 2 external IP forwarding database for a specific TOS, search for overlapping entries in the level 1 IP forwarding database for the specific TOS, and for the default TOS, and the level 2 internal IP forwarding database for the specific TOS, and for the default TOS.

a) If there is an overlapping entry Y such that either (i) Y contains X; or (ii) Y is identically specific to X; then ignore entry X.

b) Otherwise, copy entry X into the overall IP forwarding database for the specific TOS.

This method will result in one forwarding database for each supported TOS. The forwarding of packets can then be simplified to be as follows:

1) For IP packets which map to the default TOS metric (or to an unsupported TOS metric), search the default TOS forwarding database and select the entry which has the most specific match. Forward the packet accordingly.

2) For packets which map to a specific (non-default) TOS metric, search the specific TOS forwarding database and select the entry j which has the most specific match. Also search the default TOS forwarding database and select the entry k which has the most specific match. Forward the packet as follows:

a) If k is more specific than j, forward using entry k

b) If j and k are equally specific, forward using entry j

c) If j is more specific than k, forward using entry j

What is claimed is:

1. A method of calculating routes for sending user data packets via information handling devices which forward data packets through a communications network, comprising

including in at least a first and a second user data packet, destination address information conforming to two or more different addressing conventions of two or more different independent protocol suites,

determining routes for said first and said second user data packets using a route calculation algorithm corresponding to a single routing protocol, chosen from an arbitrary protocol suite, regardless of the addressing convention to which the destination address information in said user data packet conforms,

said route being determined using the destination address information in said user data packets without converting said destination address information from the addressing convention to which it forms to another addressing convention.

2. The method of claim 1 wherein there are exactly two said protocol suites.

3. The method of claim 1 or 2 further comprising

sending to said information handling devices link state packets conforming to said routing protocol, and

calculating said route based on information contained in said link state packets.

4. The method of claim 1 or 2 wherein said routing protocol comprises OSI IS-IS rotating protocol.

5. A method for calculating routes for sending user data packets via information handling devices which forward data packets through a communications network, said information handling devices including (a) single-protocol information handling devices capable of recognizing and forwardly only user data packets which conform to a single protocol suite, and (b) multiprotocol information handling devices capable of recognizing and forwarding user data packets which conform to any one of two or more protocol suites, comprising

using a routing protocol to automatically predetermine whether to encapsulate a packet, at which information handling devices to encapsulate and to decapsulate said packet, and which protocol to use to encapsulate said packet, wherein

said information handling devices are organized in areas and all of said information handling devices within a single said area are at least capable of recognizing and forwarding user data packets which conform to the same one of said protocol suites.

6. The method of claim 5 wherein the one of said protocol suites for which the information handling devices of one of said area are capable differs from the one of said protocol suites for which the information handling devices of another one of said areas are capable.

7. A method for calculating routes for sending user data packets via information handling devices which forward data packets through a communications network, said information handling devices including (a) single-protocol information handling devices capable of recognizing and forwarding only user data packets which conform to a single protocol suite, and (b) multiprotocol information handling devices capable of recognizing and forwarding user data packets which conform to any one of two or more protocol suites, comprising

using a routing protocol to automatically predetermine whether to encapsulate a packet, at which information handling devices to encapsulate and to decapsulate said packet, and which protocol to use to encapsulate said packet, wherein

said network includes (a) single-protocol information handling devices capable of recognizing and forwarding only user data packets which conform to a first protocol suite, (b) single-protocol information handling devices capable of recognizing and forwarding only user data packets which conform to a second different protocol suite, and (c) at least one

multi-protocol information handling device capable of recognizing and forwarding user data packets which conform to at least said first and second protocol suites.

8. The method of claim 7 wherein one of said multi-protocol information handling devices within a given area of said network is capable of recognizing and forwarding user data packets which conform to every protocol suite that is used by any other information handling device in said given area.

9. The method of claim 5, 6, 7 or 8 wherein one of said protocol suites comprises the TCP/IP protocol suite and another of said protocol suites comprises the OSI protocol suite.

10. The method of claim 9 further comprising

sending to said information handling devices, link state packets conforming to said one of said routing protocols, and

calculating said route based on information contained in said link state packets.

11. The method of claim 9 wherein said routing protocol used to automatically predetermine at which information handling devices to encapsulate and to decapsulate a given packet comprises OSI IS-IS routing protocol.

12. The method of claim 5, 6, 7 or 8 wherein there are exactly two of said protocol suites.

13. The method of claim 12 wherein one of said protocol suites comprises TCP/IP and the other of said protocol suites comprises OSI.

14. The method of claim 13 further comprising

sending to said information handling devices, link state packets conforming to said one of said routing protocols, and

calculating said route based on information contained in said link state packets.

15. The method of claim 13 wherein said routing protocol used to automatically predetermine at which information handling devices to encapsulate and to decapsulate a given packet comprises OSI IS-IS routing protocol.

16. The method of claim 12 further comprising

sending to said information handling devices, link state packets conforming to said one of said routing protocols, and

calculating said route based on information contained in said link state packets.

17. The method of claim 12 wherein said routing protocol used to automatically predetermine at which information handling devices to encapsulate and to decapsulate a given packet comprises OSI IS-IS routing protocol.

18. The method of claim 5, 6, 7 or 8 further comprising

sending to said information handling devices, link state packets conforming to said one of said routing protocols, and

calculating said route based on information contained in said link state packets.

19. The method of claim 5, 6, 7 or 8 wherein said routing protocol used to automatically predetermine at which information handling devices to encapsulate and to decapsulate a given packet comprises OSI IS-IS routing protocol.

20. A method for calculating routes for sending user data packets via information handling devices which forward data packets through a communications network, said information handling devices including (a) single-protocol information handling devices capable of recognizing and forwarding only user data packets which conform to a single protocol suite, and (b) multi-protocol information handling devices capable of recognizing and forwarding user data packets which conform to any one of two or more protocol suites, comprising

using a routing protocol to automatically predetermine whether to encapsulate a packet, at which information handling devices to encapsulate and to decapsulate said packet, and which protocol to use to encapsulate said packet, wherein

said routing protocol determines multiple routes from a starting information handling device to an ending information handling device, and

different protocols are recognized by corresponding devices on different routes,

such that the automatic predetermination depends upon the route used.

21. A network of information handling devices which forward user data packets through communications links each said packet including an address indicating the packet's destination, said network comprising

a first information handling device capable of interpreting addresses formatted in accordance with a first addressing scheme, said first addressing scheme defining one or more address fields that are assigned values to form an address,

a second information handling device capable of interpreting addresses formatted in accordance with a second addressing scheme, said second addressing scheme defining different address fields than said first addressing scheme, wherein

at least one address formatted in accordance with said first addressing scheme dose not identify the same destination as any address formatted in accordance with said second addressing scheme, and

said first and second devices exchange control packets which specify:

the links connected to the device which originates the control packet,

the addresses, formatted in accordance with said first addressing scheme, of those destinations that are attached to the originating device and have an address corresponding to said first addressing scheme, and

the addresses, formatted in accordance with said second addressing scheme, of those destinations that are attached to the originating device and have an address corresponding to said second addressing scheme.

22. The network of claim 21 wherein

said first device is capable of interpreting addresses formatted in accordance with said second addressing scheme, and

as part of forwarding at least one given packet addressed in accordance with said first addressing scheme, said first device encapsulates said given packet inside a header addressed in accordance with said second addressing scheme, and transmits the encapsulated packet to said second device.

23. The network of claim 22 wherein

said network is organized into areas comprising routers and links which connect said routers, and

every device in a given area is capable of interpreting addresses formatted in accordance with at least one common addressing scheme.

24. The network of claim 21, 22, or 23 wherein

said first and second devices are configured to transmit a hello packet on a link when the link is connected thereto, said hello packet identifying the device originating the hello packet and indicating the addressing schemes in which the originating device is capable of interpreting addresses.

* * * * *

Exhibit 2

US005390173A

United States Patent [19]

Spinney et al.

[11] Patent Number: 5,390,173

[45] Date of Patent: Feb. 14, 1995

[54] **PACKET FORMAT IN HUB FOR PACKET DATA COMMUNICATIONS SYSTEM**

[75] Inventors: **Barry A. Spinney**, Wayland; **Robert J. Simcoe**, Westboro; **Robert E. Thomas**, Hudson; **George Varghese**, Bradford, all of Mass.

[73] Assignee: **Digital Equipment Corporation**, Maynard, Mass.

[21] Appl. No.: **965,651**

[22] Filed: **Oct. 22, 1992**

[51] **Int. Cl.**[6] **H04J 3/26**; H04L 12/46

[52] **U.S. Cl.** **370/60**; 370/85.13; 370/94.1

[58] **Field of Search** 370/60, 60.1, 85.9, 370/85.13, 85.14, 94.1, 94.2, 94.3, 85.6; 340/825.02, 825.03, 825.8, 825.52

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,706,081	11/1987	Hart et al.	370/85.13 X
4,933,937	6/1990	Konishi	370/94.1 X
5,020,020	5/1991	Pomfret et al.	370/94.1 X
5,060,228	10/1991	Tsutsui et al.	370/85.9 X
5,088,091	2/1992	Schroeder et al.	370/85.13 X
5,115,432	5/1992	Haas	370/94.1
5,121,383	6/1992	Golestani	370/85.6 X
5,220,562	6/1993	Takada et al.	370/85.13

FOREIGN PATENT DOCUMENTS

0281785	9/1988	European Pat. Off.	H04L 11/16
0404337	12/1990	European Pat. Off.	H04L 12/28
0473066	3/1992	European Pat. Off.	H04L 12/46
4103888	8/1991	Germany	H04L 12/28

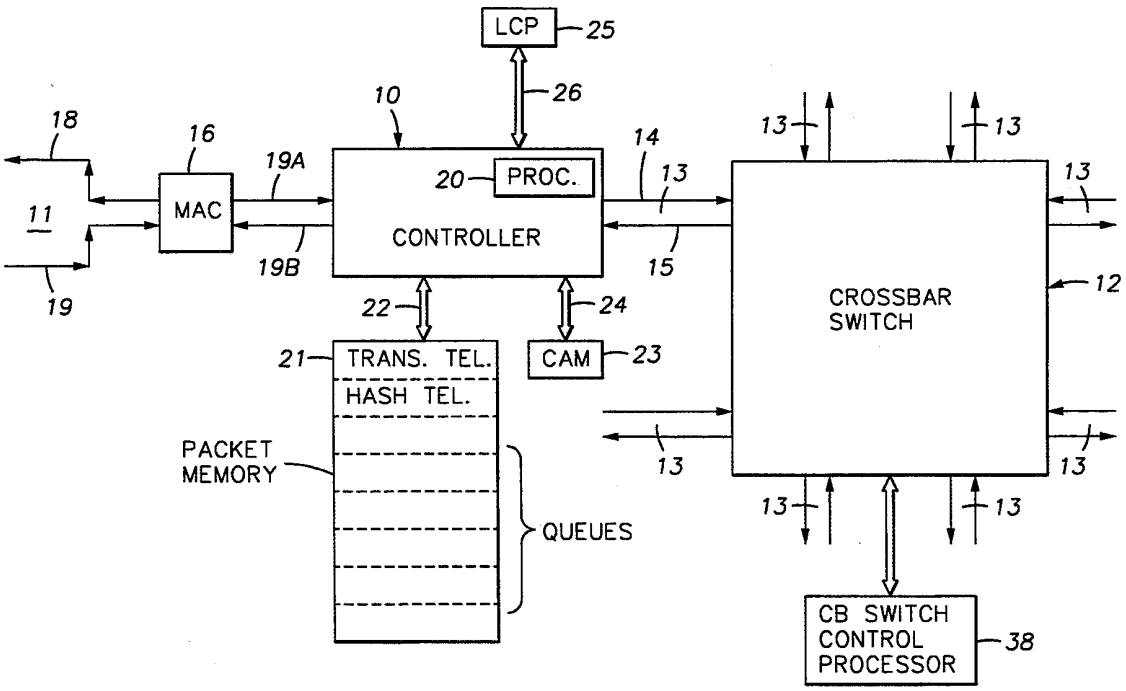
Primary Examiner—Melvin Marcelo

Attorney, Agent, or Firm—Kenneth F. Kozik; A. Sidney Johnston

[57] **ABSTRACT**

A packet data communication network employs a local switch, router or bridge device functioning to transfer packets between segments of a larger network. When packets enter this device, an address translation is performed to generate local source and destination addresses which are much shorter than the globally-unique addresses contained in the packet as dictated by the protocol. These local addresses are inserted in a header that is added to the packet, in addition to any header already contained in the packet. This added header travels with the packet through the local switch, router or bridge device, but then is stripped off before the packet is sent out onto another network segment. The added header may also contain other information, such as a local name for the source and destination segment (link), as well as status information that is locally useful, but not part of the packet protocol and not necessary for transmission with the packet throughout the network. Local congestion information, results of address translations, and end-of-message information, are examples of such status information.

24 Claims, 8 Drawing Sheets



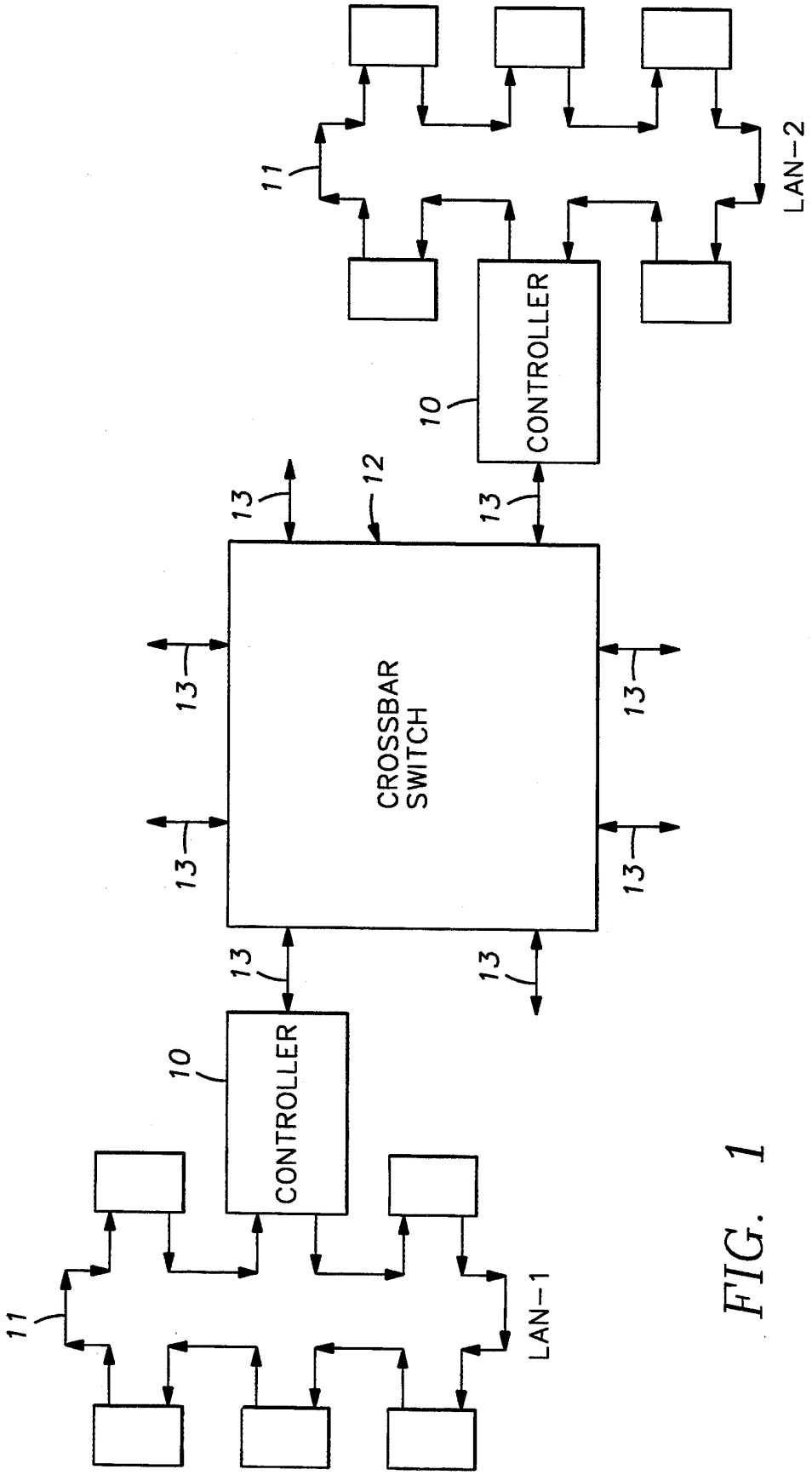

LAN-2
11
CONTROLLER
10
12
13
CROSSBAR SWITCH
CONTROLLER
10
LAN-1
11

FIG. 1

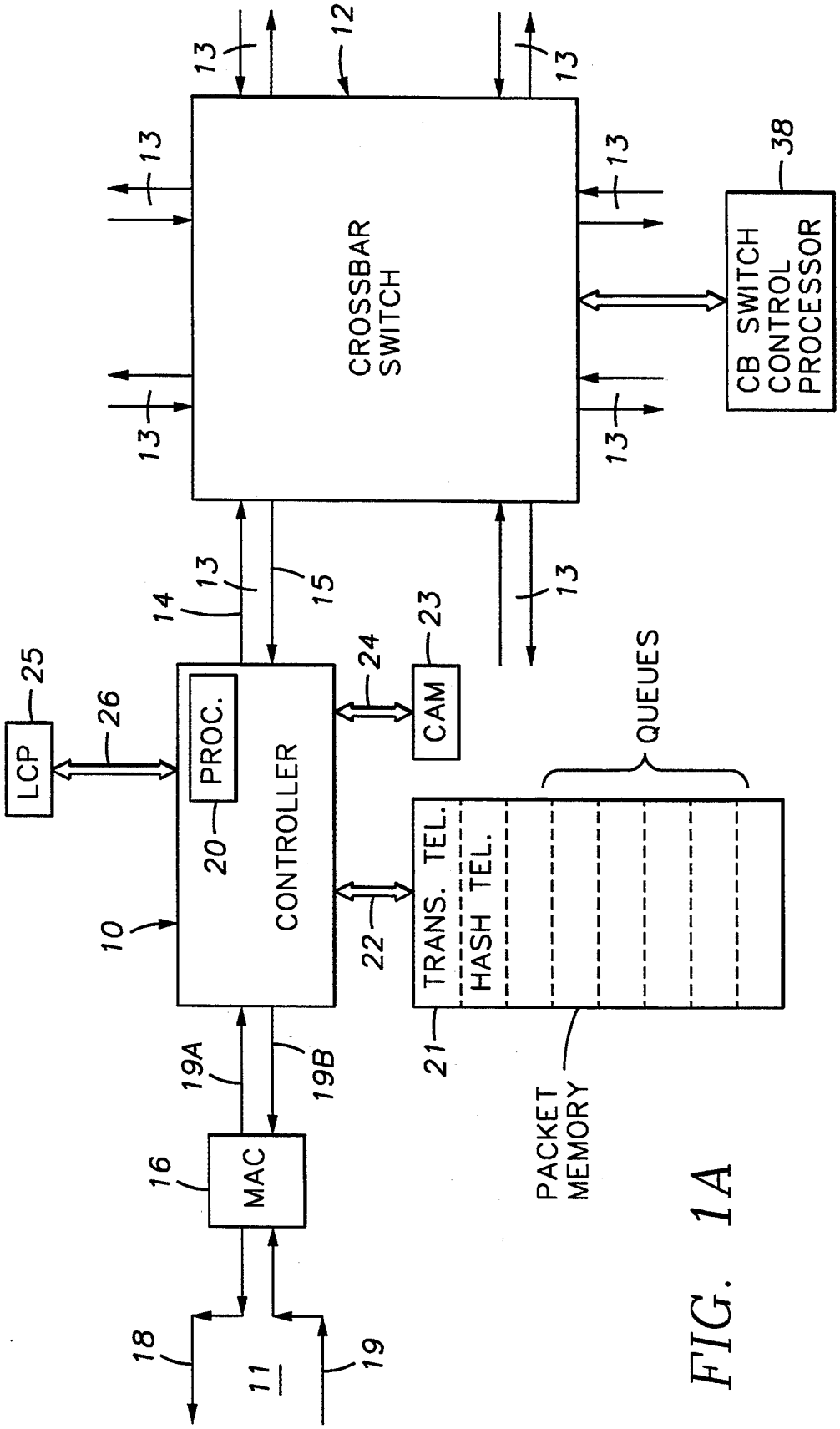

CROSSBAR SWITCH
CB SWITCH CONTROL PROCESSOR
CONTROLLER
PROC.
LCP
CAM
QUEUES
TRANS. TEL.
HASH TEL.
PACKET MEMORY
MAC
10
11
12
13
14
15
16
18
19
19A
19B
20
21
22
23
24
25
26
38

FIG. 1A

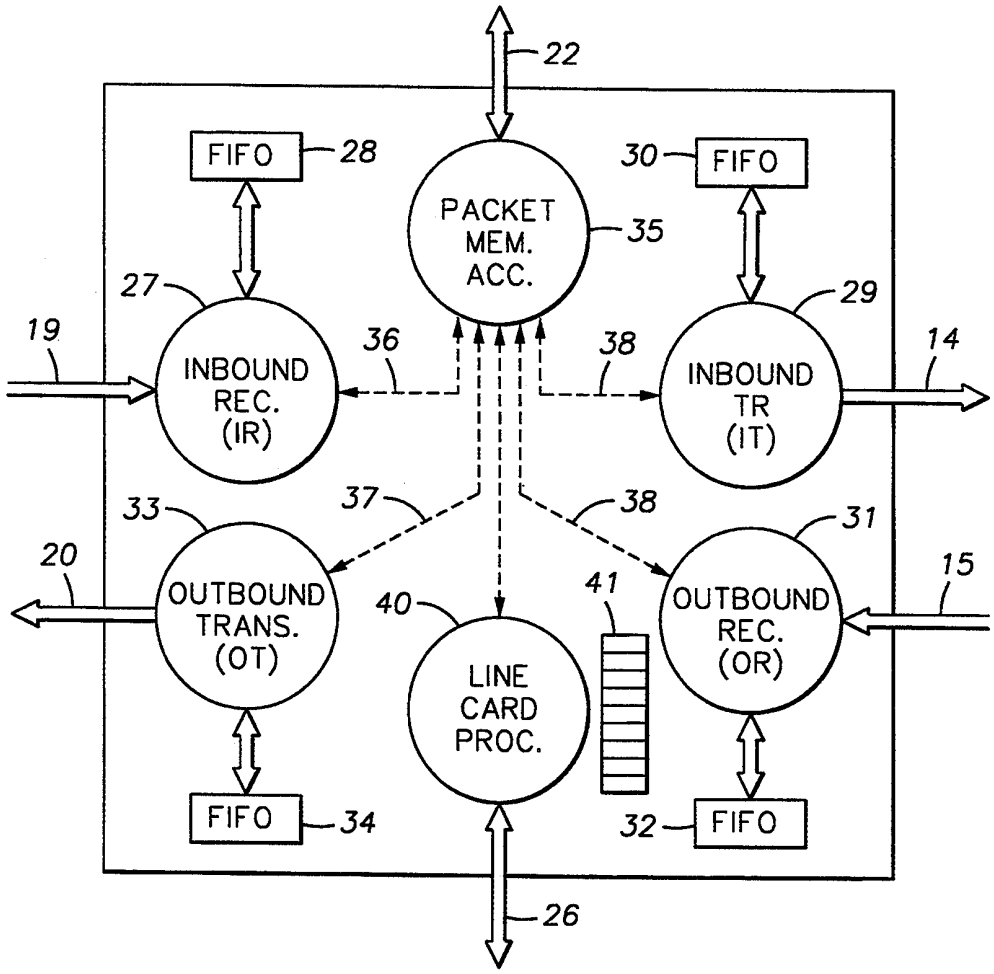

22
FIFO
28
30
FIFO
PACKET
MEM.
ACC.
35
27
29
19
36
38
14
INBOUND
REC.
(IR)
INBOUND
TR
(IT)
37
38
33
31
20
15
OUTBOUND
TRANS.
(OT)
40
41
OUTBOUND
REC.
(OR)
LINE
CARD
PROC.
FIFO
34
32
FIFO
26

FIG. 2

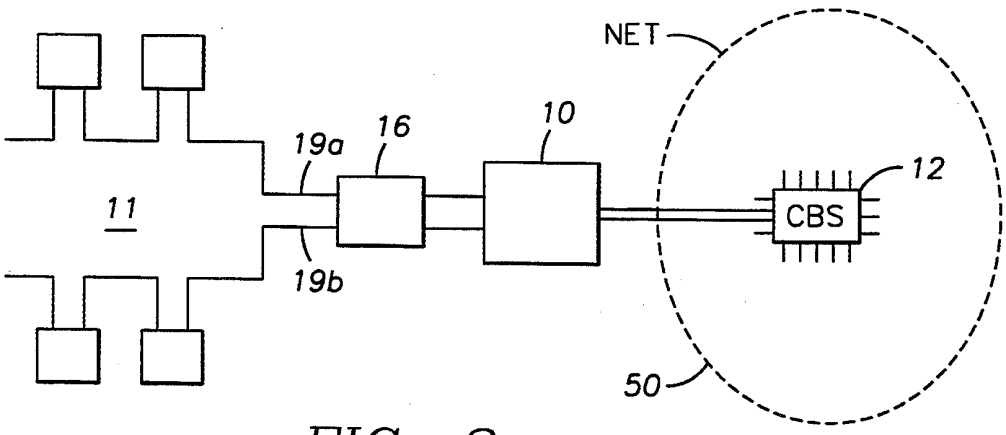

NET
19a
16
10
12
11
CBS
19b
50

FIG. 3

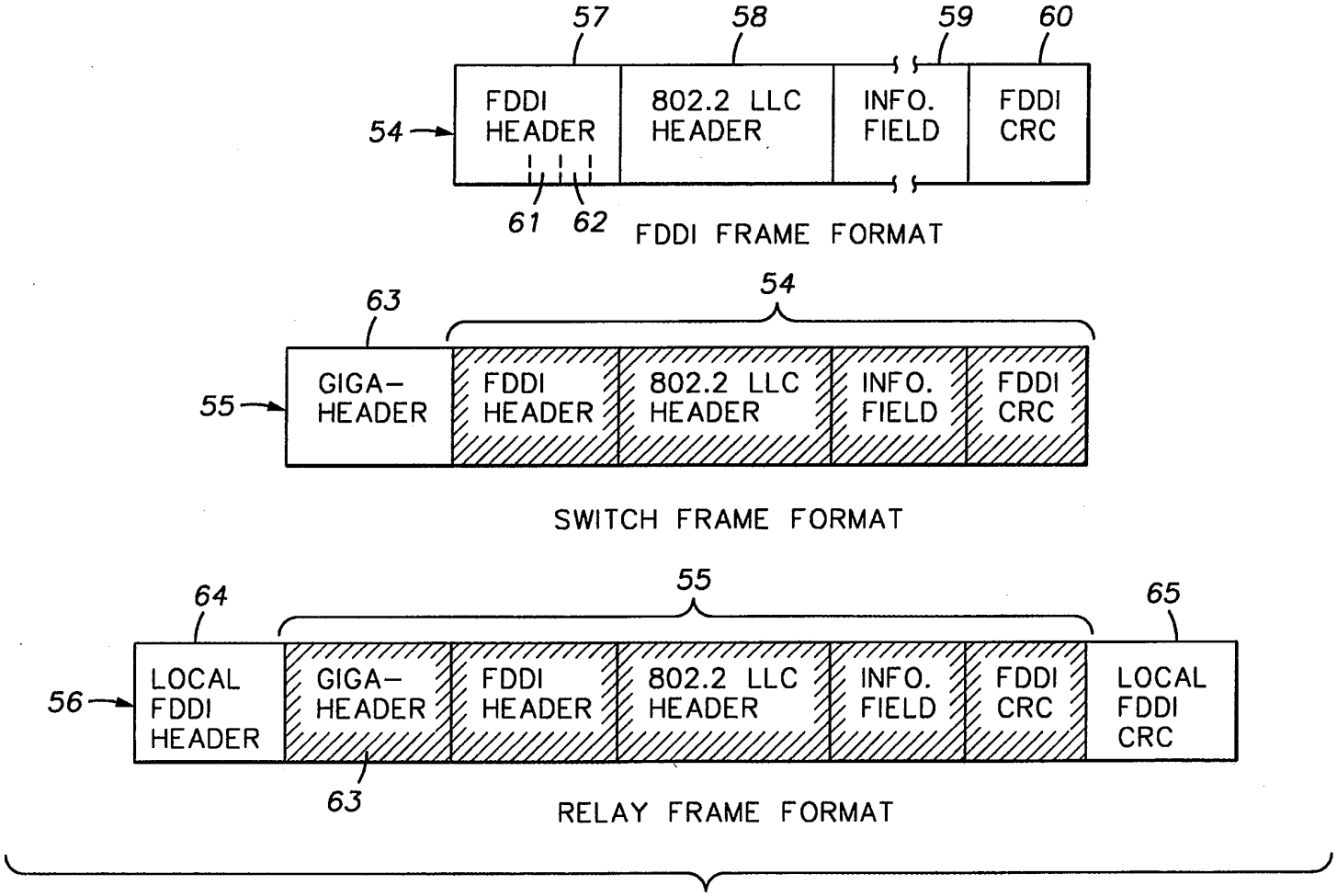

57
58
59
60
54
FDDI HEADER
802.2 LLC HEADER
INFO. FIELD
FDDI CRC
61
62
FDDI FRAME FORMAT
63
54
55
GIGA– HEADER
FDDI HEADER
802.2 LLC HEADER
INFO. FIELD
FDDI CRC
SWITCH FRAME FORMAT
64
55
65
56
LOCAL FDDI HEADER
GIGA– HEADER
FDDI HEADER
802.2 LLC HEADER
INFO. FIELD
FDDI CRC
LOCAL FDDI CRC
63
RELAY FRAME FORMAT

FIG. 4

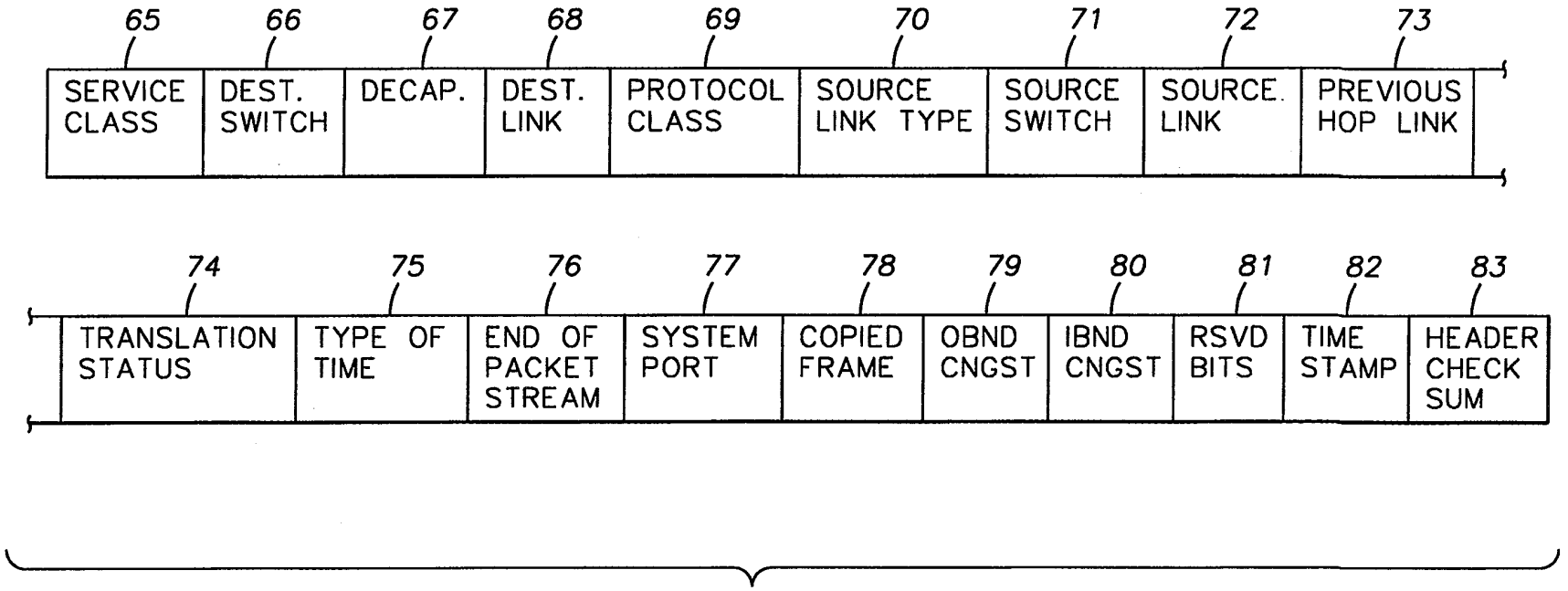

65
66
67
68
69
70
71
72
73
SERVICE CLASS
DEST. SWITCH
DECAP.
DEST. LINK
PROTOCOL CLASS
SOURCE LINK TYPE
SOURCE SWITCH
SOURCE LINK
PREVIOUS HOP LINK
74
75
76
77
78
79
80
81
82
83
TRANSLATION STATUS
TYPE OF TIME
END OF PACKET STREAM
SYSTEM PORT
COPIED FRAME
OBND CNGST
IBND CNGST
RSVD BITS
TIME STAMP
HEADER CHECK SUM

FIG. 5

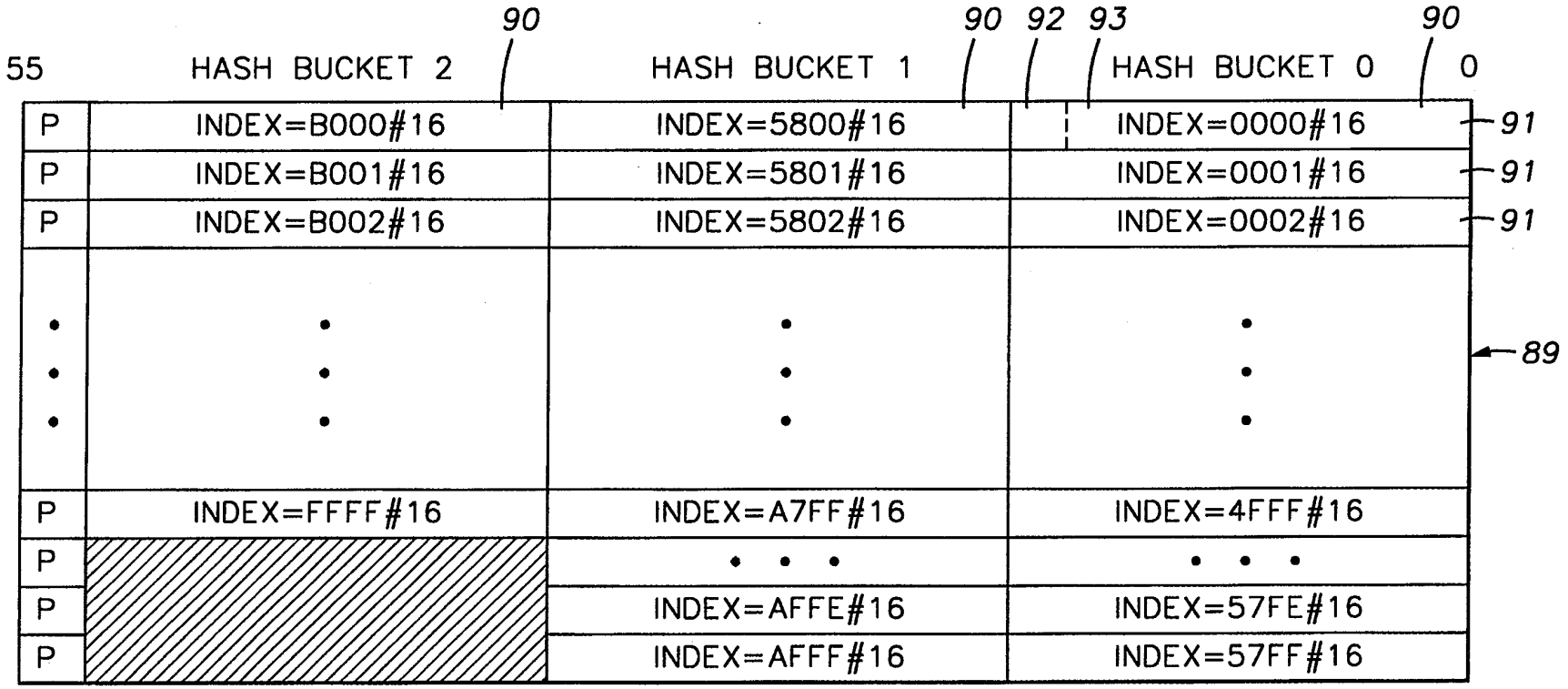

55
HASH BUCKET 2
90
HASH BUCKET 1
90
92
93
HASH BUCKET 0
90
0
P
INDEX=B000#16
INDEX=5800#16
INDEX=0000#16
91
P
INDEX=B001#16
INDEX=5801#16
INDEX=0001#16
91
P
INDEX=B002#16
INDEX=5802#16
INDEX=0002#16
91
89
P
INDEX=FFFF#16
INDEX=A7FF#16
INDEX=4FFF#16
P
P
INDEX=AFFE#16
INDEX=57FE#16
P
INDEX=AFFF#16
INDEX=57FF#16

FIG. 6

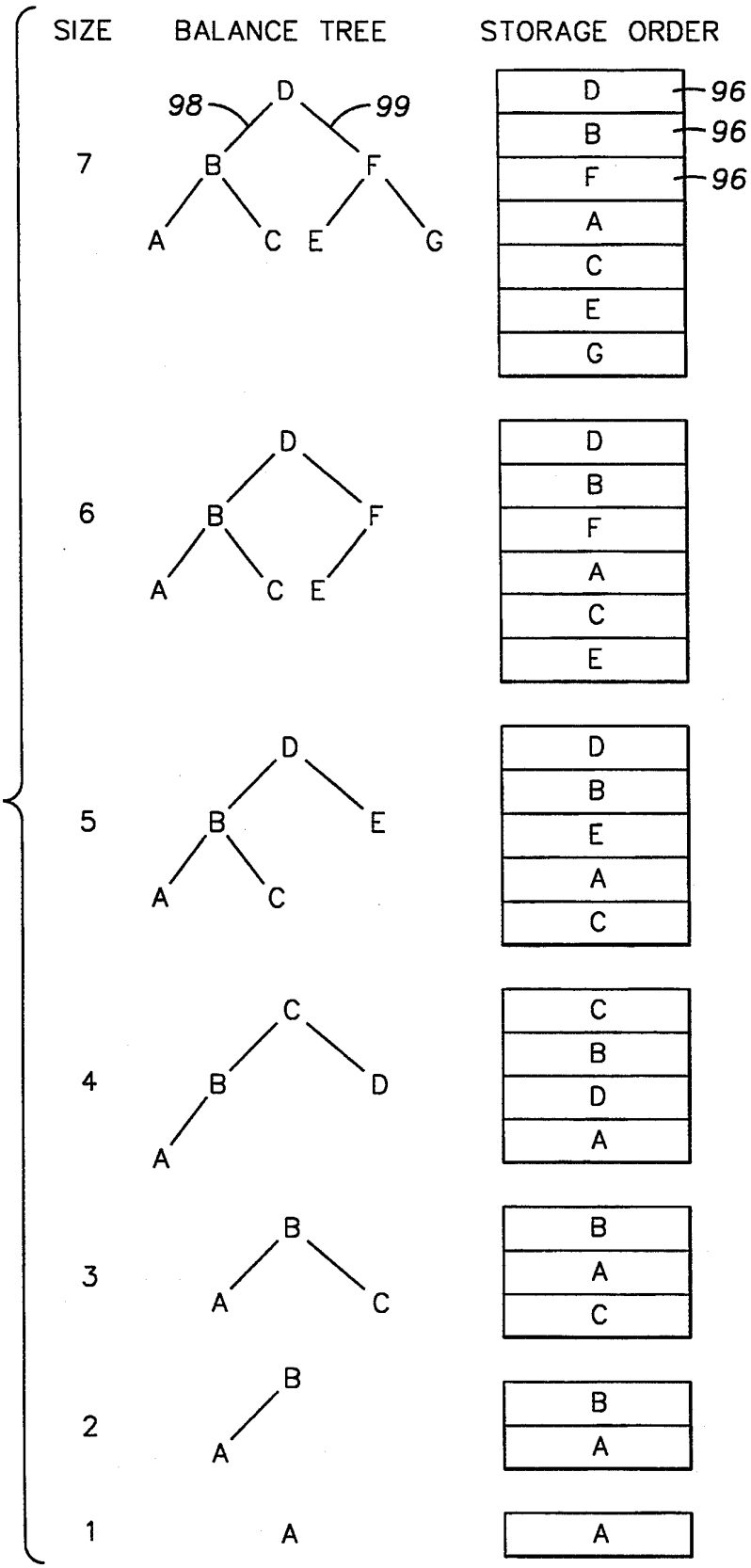

SIZE
BALANCE TREE
STORAGE ORDER
7
98
99
D
B
F
A
C
E
G
96
96
96
6
5
4
3
2
1
A

FIG. 7

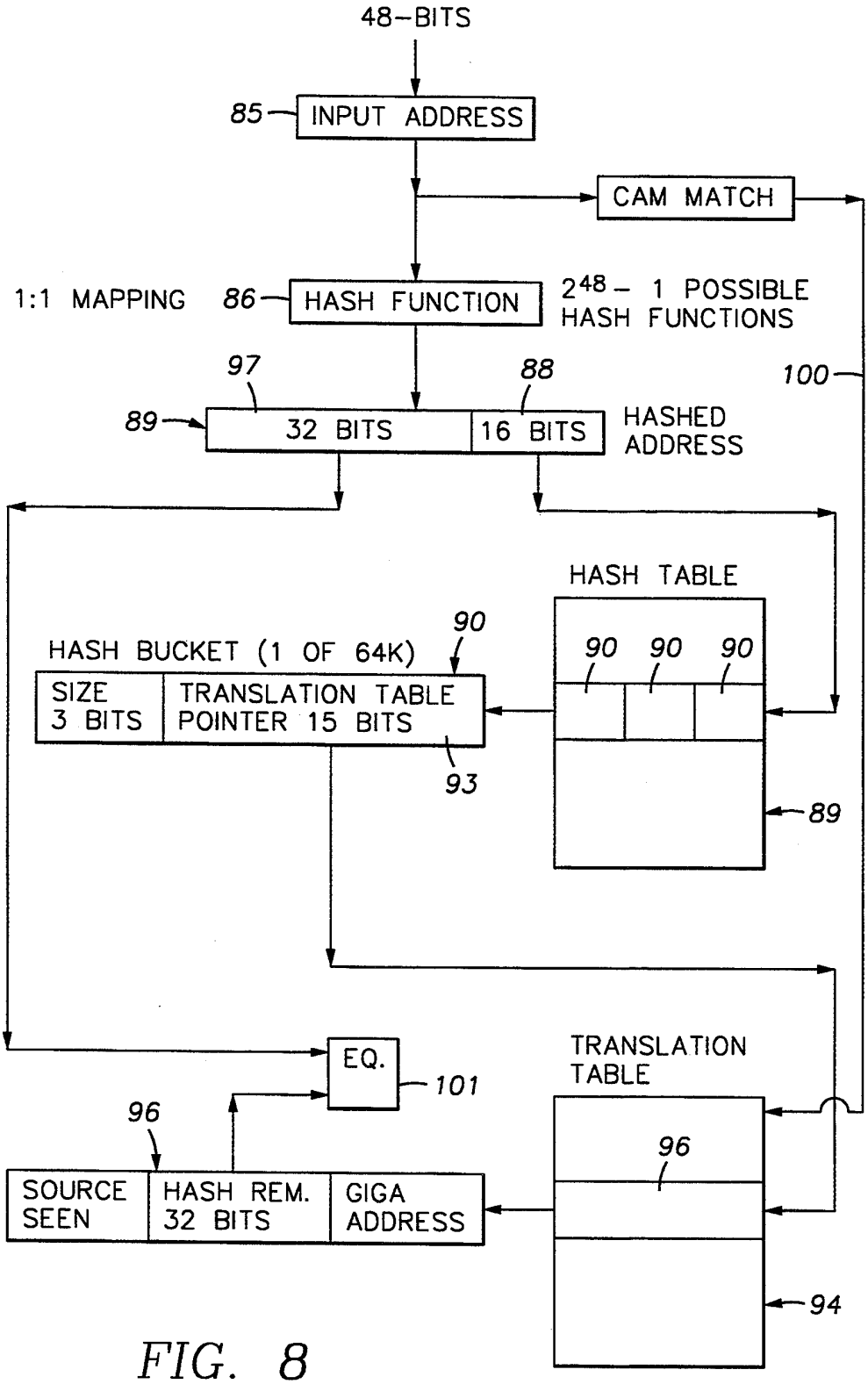

48–BITS
85
INPUT ADDRESS
CAM MATCH
1:1 MAPPING
86
HASH FUNCTION
2^{48} – 1 POSSIBLE HASH FUNCTIONS
97
88
100
89
32 BITS
16 BITS
HASHED ADDRESS
HASH TABLE
90
HASH BUCKET (1 OF 64K)
90 90 90
SIZE 3 BITS
TRANSLATION TABLE POINTER 15 BITS
93
89
EQ.
101
TRANSLATION TABLE
96
96
SOURCE SEEN
HASH REM. 32 BITS
GIGA ADDRESS
94

FIG. 8

PACKET FORMAT IN HUB FOR PACKET DATA COMMUNICATIONS SYSTEM

RELATED CASES

The application discloses subject matter also disclosed in the following copending U.S. patent applications, all of which are assigned to Digital Equipment Corporation:

Ser. No. 07/964,791, filed Oct. 22, 1992, by Nigel Terence Poole, for "BACKPLANE WIRING FOR HUB IN PACKET DATA COMMUNICATIONS SYSTEM" (PD92-0558);

Ser. No. 07/964,792, filed Oct. 22, 1992, by Nigel Terence Poole, for "CROSSBAR SWITCH FOR SYNTHESIZING MULTIPLE BACKPLANE INTERCONNECT TOPOLOGIES IN COMMUNICATIONS SYSTEM" (PD92-0559);

Ser. No. 07/964,738, filed Oct. 22, 1992, by Bryan Alan Spinney, for "ADDRESS LOOKUP IN PACKET DATA COMMUNICATIONS LINK, USING HASHING AND CONTENT-ADDRESSABLE MEMORY" (PD93-0016); and

Ser. No. 07/965,121, filed Oct. 22, 1992, by Martin Edward Griesmer et al, for "APPARATUS AND METHOD FOR MAINTAINING FORWARDING INFORMATION IN A BRIDGE OR ROUTER" (PD93-0013).

BACKGROUND OF THE INVENTION

This invention relates to packet data communication networks, and more particularly to a way of handling packets through a hub, switch or router for a network, using a special header attached to the packets.

Packet data communication networks of type using Ethernet, token ring, or FDDI technologies, or other network varieties, hubs are used for switching or routing, or for bridges to additional segments of the network. A bridge or router in a computer interconnect system is shown in U.S. Pat. No. 5,020,020, assigned to Digital Equipment Corporation. It is typical to use address translation in a bridge or router of this type, as shown, for example, in U.S. Pat. No. 4,933,937. The need to translate addresses is due to the address length. Some protocols or system specify a 48-bit source and destination address so that a globally unique address is provided. However, for efficient use of resources at a local segment of a large network (as within a bridge), it is advantageous to use smaller address fields instead of 48-bit addresses, for efficiency in bit-count of messages as well as efficiency in processing and storage. For this reason, while the 48-bit addresses are carried in the packet throughout its lifetime, shorter addresses are generated for local routing and processing. Thus, a translation mechanism is provided to allow switching between global and local addresses.

Various other information may be locally useful in a bridge or router that need not be carried by a packet throughout its lifetime. For example, the network segments (links) may be locally named (assigned an identifying number) so that packets may be routed by link number instead of or in addition to their local destination node address. The packets may be assigned a priority or "service class" for local use, aside from the priority attached to the packet by the protocol being implemented. Status information may be locally useful, but not part of the packet protocol and not necessary for transmission with the packet throughout the network.

For example, local congestion information, or results of address translations, or end-of-message information, are types of such status information.

SUMMARY OF THE INVENTION

In accordance with one embodiment of the invention, a packet data communication network employs a local switch, router or bridge device functioning to transfer packets between segments of a larger network. When packets enter this device, an address translation is performed to generate local source and destination addresses which are much shorter than the globally-unique addresses contained in the packet as dictated by the protocol. These local addresses are inserted in a header that is added to the packet, in addition to any header already contained in the packet. This added header travels with the packet through the local switch, router or bridge device, but then is stripped off before the packet is sent out onto another network segment. The added header may also contain other information, such as a local name for the source and destination segment (link), as well as status information that is locally useful, but not part of the packet protocol and not necessary for transmission with the packet throughout the network. Local congestion information, results of address translations, and end-of-message information, are examples of such status information.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as other features and advantages thereof, will be best understood by reference to the detailed description of a specific embodiment, when read in conjunction with the accompanying drawings, wherein:

FIG. **1** is a diagram in block form of a communications network which may use features according to one embodiment of the invention;

FIG. **1***a* is an electrical diagram in block form of a controller for the communications network of FIG. **1**;

FIG. **2** is a diagram of the controller of FIG. **1** and **1***a* showing processes executed in the controller;

FIG. **3** is a diagram of a controller of FIG. **1** and **1***a* connected in a network;

FIG. **4** is a diagram of frame formats used in the network of FIGS. **1** or **3**;

FIG. **5** is a diagram of the fields contained in an added header in the formats of FIG. **3**;

FIG. **6** is a diagram of a data structure used for a hash table in the system of FIGS. **1** and **1***a*;

FIG. **7** is a diagram of breadth-first binary trees used in the method of the invention, to store translated addresses; and

FIG. **8** is a flow chart of address lookup procedures used in the embodiment of the invention of FIGS. **1** and **1***a*.

DETAILED DESCRIPTION OF SPECIFIC EMBODIMENT

Referring to FIG. **1**, a packet data communications network which may use the features of the invention includes a controller **10** for interface between an FDDI link **11** and a crossbar switch device **12**. The crossbar switch device **12** has a number of input/output ports **13**, and each one of these ports **13** may be connected by another controller **10** to another network segment **11**

such as an FDDI link or a token ring or Ethernet bus, for example. The crossbar switch **10** ordinarily makes a direct point-to-point interconnect between one port **13** and another port **13**, so that the crossbar acts as a bridge or router in the network, linking one network segment to another. A station on a link **11** sends a packet onto its network segment with a destination address which is on another, different segment. The controller **10** for this segment detects the address as being that of a station on one of the remote segments, and generates local switching information to send to the crossbar **12** so that the appropriate interconnect can be made to send the packet to the proper port **13** and link **11**, via another controller **10**. As set forth in the above-mentioned copending applications, the crossbar switch device can function as a flexible interconnect device to create a ring or bus using the ports **13**, as well as functioning as a point-to-point connector as is the usual case for crossbar switches.

Referring to a more detailed view of FIG. **1***a*, each port **13** of the crossbar has a data-in path **14** and a separate data-out path **15**. The interface between the controller **10** and the FDDI link **11** is by way of a media access control (MAC) device **16**, functioning to convert the serial light transmission on the incoming fiber optic cable **17** to electrical pulses, to recover the clock, convert the serial data on the optic loop to 6-bit parallel symbols, act as an elastic buffer to allow reclocking of data entering the controller **10**, etc. Of course, all of these functions are reversed for outgoing data on the cable **18**. The interface between the controller **10** and the MAC device **16** is by an incoming 8-bit parallel data path **19***a* (with additional parity and control lines) and an outgoing 8-bit parallel path **19***b*.

The controller **10** contains a processor or state machine **20** to execute various processes as will be described, and accesses a packet memory **21** via an interface **22**, as well as a content addressable memory (CAM) **23** via interface **24**. The packet memory **21** is addressed by a 20-bit address bus, and data is transferred by a 56-bit bidirectional data bus, included in the interface **22**; various control lines are also in the interface **22**. The CAM **23** is driven by a 14-bit bus and various control lines in the interface **24**. The packet memory **21** is a RAM which stores a number of queues for incoming and outgoing data packets, as well as translation tables and hash tables as will be described. In addition, the packet memory stores certain data for which addresses are matched in the CAM **23**.

The controller **10** also interfaces with a line card processor **25** by bus **26**. The line card processor **25** is used to execute some diagnostic and initialization functions, and does not operate in routine packet transfer.

Logically, the processor **20** in the controller **10** executes six independent processes. There are two for inbound packet processing, two for outbound packet processing, one for interfacing to the external packet memory, and one for line card processor access. Packets inbound on FDDI line **17** and going through the controller **10** to the crossbar switch **12** are referred to as "inbound."Likewise, packets going in the direction of crossbar switch **12** through the controller **10** to the FDDI output line **18** are referred to as "outbound."By having independent processes which can operate in parallel, the controller **10** can process inbound and outbound packets at full speed. Distributed among the processes are control, parameter and status registers that are used to define the operational modes and to determine the internal state of the controller **10**; these registers are accessed through the node processor interface **26**.

Referring to FIG. **2**, the inbound receive (IR) process **27** executed on the processor **20** receives packets from the interface **19***a* to the FDDI ring via the media access control device **16**. The IR process **27** parses and decodes each incoming packet from line **19***a*, places the packet data into an intermediate FIFO **28**, and performs the necessary processing to determine if the packet is to be forwarded to the crossbar **12**. For bridged packets, this decision is made by performing destination address, source address, and protocol field lookups, with the aid of the content addressable memory **23** and the internal hash function, according to the invention. The result of the lookup will determine where the packet is to go. The packet may have to be transmitted to another port **13** on the crossbar **12** to reach its destination, the packet may be destined for the line card processor **25**, the packet may be destined for a processing engine **38** in the crossbar switch **12**, or the packet may be filtered and discarded. When the IR process **27** decides a packet is to be forwarded to the crossbar **12**, it uses the lookup results to generate the information in an added header as will be described. It will select an inbound queue to store the packet in the external packet memory **21** and will initiate memory requests to transfer the data from its intermediate FIFO **28** to the selected queue in the external packet memory **21**. The processor **20** executing this IR process **27** performs this operation by generating IR requests that are sent to the packet memory **21**.

The IR process **27** also handles a number of exception conditions. The forwarding of packets to processing engines in the switch **12** can be rate limited by software to prevent the traffic from a single port from overwhelming the shared processing resources of the switch **12**. When these rate limits are exceeded the IR process **27** will discard these packets until the rate limit is replenished.

The inbound transmit (IT) process **29**, again referring to FIG. **2**, services the queues in packet memory **21** that contain packets to be transmitted to the crossbar switch **12**. These packets are stored in queues in the external packet memory **21** of the controller **10**. When an enabled queue has a packet count greater than one, the IT process **29** will begin processing of the packet by initiating packet memory requests to move data from the packet memory **21** to an internal FIFO **30**. Part of the data transferred is the added header associated with the packet. The IT process **29** uses this added header field to request a connection on the crossbar **12** to the desired destination port **13**; it requests this connection by sending information to the port interface **13** that indicates the destination port of the switch **12** and service class information describing how the connection is to be serviced. Prior to requesting the connection, the IT process **29** checks the timestamp stored with the packet and will discard the packet if it has expired. If the IT process **29** determines that a packet has expired after a connection has been requested but before the connection is made, it will discard the packet, abort the requested connection and continue servicing the enabled queues.

When a connection is established, the IT process **29** transfers the data from its intermediate FIFO **30** to the crossbar **12** via path **14** and performs the proper packet formatting during transmission.

The outbound receive (OR) process **31** executing on the processor **20** receives packets from the crossbar switch **12** interface **15**, parses and decodes each packet, places the packet data into an intermediate FIFO **32** and performs the necessary validity checks of the packet's added header to determine if the packet should be received and placed at the tail of an enabled queue in packet memory **21**. Packet reception is established when the controller **38** for the crossbar indicates that this controller **10** is the desired destination port for a connection. The OR process **31** indicates its willingness to take part in a connection by asserting a "next" control signal in the interface **15** that the crossbar controller **38** samples. When this "next" signal is asserted, the crossbar controller may return a signal indicating that a connection between a source port and this destination port has been made on the crossbar **12**. The OR process **31** immediately monitors the outbound data received at the crossbar interface **15** to delimit a packet. It will also deassert the "next" signal and will not reassert it until it has determined that it is appropriate to establish a new connection.

The OR process **31** uses the service class field and one specific protocol class field value from the added header of the outbound packet to place the packet on a queue in external packet memory **21**. The OR process **31** also handles multiple exception conditions on the crossbar including parity errors in the symbols, and packets with bad delimiters.

The outbound transmit (OT) process **33** services the queues in packet memory **21** that contain packets to be transmitted to the FDDI line **18** via path **19***b* and device **16**. These packets are stored in the external packet memory **21** of the controller **10**. When an enabled queue has a packet count greater than one, the OT process **33** will begin processing the packet by initiating packet memory requests to move the data from the external memory **21** to an internal fifo **34**. The OT process **33** will discard the packet if the time stamp stored with the packet has expired, otherwise it will request the MAC device **16** to begin a transmission to the FDDI line **18**. If the OT process **33** is operating in a bridge link mode, it will decapsulate the packet by removing the added header that was stored with the packet, prior to transmission on the path **19***b* to the FDDI link. If the OT process **33** is attached to a switch link, it will keep and update the appropriate fields of the added header during transmission to the path **19***b* and FDDI link.

The packet memory (PM) access process **35** controls the interface **22** to the external packet memory **21**. The packet memory **21** is used to hold the packet data of the multiple inbound and outbound queues and is also used to store the PC table, the hash table as described herein, and the translation database used for lookups. The PM process **35** controls access to the external packet memory **21** by arbitrating among all the processes requesting access to it. The PM process **35** will indicate to the requesting process when a cycle has been granted. The external packet memory is accessed via a 56-bit data bus in interface **22**. The PM process **35** can select up to 1-million 56-bit words over its 20-bit external packet memory address bus in interface **22**. Read or write cycles can be performed every 80-ns to provide 600-Mbps of packet memory bandwidth.

When a process **27**, **29**, **31**, or **33** requests a memory cycle via paths **36**, **37**, **38**, or **39**, respectively, it also indicates to the PM process **35** the desired queue that the cycle is associated with by driving the queue number. When the PM process **35** grants the cycle to the requesting process, it uses the queue number provided to simultaneously enable the pointers for the queue to the external memory **21**. The PM process **35** will drive the corresponding tail (write cycle) or head (read cycle) pointer onto the external address pins of interface **22** and will perform the necessary pointer maintenance when the cycle completes. The maintenance includes wrapping the head or tail pointer when the boundary of a queue is reached. The PM process **35** will also monitor the queue pointers to detect and signal when a queue is approaching an overflow condition.

Each queue in the external packet memory **21** must be defined before the PM process **35** can perform an access to it. To define a queue, the boundaries for the queue must be specified by programming the external memory block addresses that point to the beginning and end of each queue. A queue is programmable in size in increments of 256 56-bit word units (1792 bytes) which constitute a block. In addition to defining the size of the queue, the owners of the queue head and tail must be specified, to control the direction of packet flow. A separate queue enable is provided for each queue in order to turn each one on and off independently.

The line card processor (LCP) access process **40** provides the interface between the processor **25** and the command and status registers **41** within the controller **10**. Some of these registers are accessed directly by the processor **25** and some are accessed indirectly using the LCP process **40**. The registers **41** that are indirectly accessible are shared among the processes **27**, **29**, **31**, **33**, and **35** of the controller **10** and must be arbitrated for. The indirect access interface of the LCP process **40** provides the mechanisms to select the desired indirect register and to trigger the desired read or write operation. Since the processor **25** does not have direct access to the external packet memory **21** of the controller **10**, it must also use the indirect access interface of the LCP process **40** to perform read and write cycles to the external memory **21**.

In addition to direct and indirect register **41** access, the LCP process **40** also maintains the traffic event counters and interrupt event registers. The LCP process **40** monitors the traffic events signaled by the processes **27**, **29**, **31**, **33**, and **35** to update the corresponding counter. In addition to specified counters, the LCP process **40** provides two general purpose counters that can be programmed to count a particular traffic event. The LCP monitors events in the controller **10** and will set corresponding interrupts. These interrupts are categorized to be associated with inbound traffic, outbound traffic, or fatal interrupts. All interrupts, except fatal interrupts, are maskable. The presence of a non-masked interrupt will result in the assertion of an interrupt signal.

The operation of the processor **20** in controller **10** using the processes referred to includes a number of functions related to inbound and outbound packets. These include filtering, address lookup, address matching, rate limiting, parity checking, etc.

The controller **10** performs input port filtering to prevent undesired traffic from entering the net, and this filtering is based on certain addresses contained in a packet. A 48-bit destination address and a 48-bit source address in the packet can be filtered for a number of addresses, up to the size of the translation table in memory **21**. Certain addresses identified in the protocol as IEEE 802.2 LLC DSAP and IEEE 802.2 LLC SNAP

can be filtered by checking against items stored in the CAM **23**, e.g., 256 entries.

For address lookup, the controller **10** implements an optimized hash algorithm which can perform a 48-bit address lookup in at most four references to the external memory **21**. The hash function is programmable, and is specified by a 48-bit value called the hash function. The lookup includes one memory reference to a hash table in memory **21**, followed by at most three references to the translation database which contains a breadth-first binary tree in which hashed 48-bit addresses are stored, as will be explained.

The controller **10** supports exact matching of certain 48-bit destination addresses (group or individual), certain 48-bit source addresses, 8-bit DSAP and 40-bit PID fields, by an interface to the external CAM **23**. The controller **10** can obtain by interface **24** a 14-bit associated data field for each entry stored in the CAM **23**. The ability to match data fields of varying byte widths is achieved by specifying a type code field driven by the controller **10** during the reception of packets. The bits of the type code indicate the size of the field to match in the CAM array **23** and the type of key data. A key can be from one to six bytes in length. When a 48-bit destination or 48-bit source address is found as an entry in the CAM **23**, the controller inhibits the hash lookup for this value.

The controller **10** provides programmable rate limits to throttle the packet rate and limit the transmit queue length of excess packets that are destined for switch processing engines. For example, these rate limits would be applied to errored packets, monitor packets, or bridge learning packets with an unknown source address to be sent to the controller **38** for the switch **12**.

The controller **10** performs output port filtering based on the added header protocol class, and performs parity checking of the crossbar data path.

The controller **10** can be used to attach a variety of links to the crossbar **12**. The parameters that define the operation of controller **10** describe the type of link to which the controller **10** is attached through the MAC interface **19***a*, **19***b*, the type of inbound and outbound forwarding procedures it is to perform, and the format of the packets it is to handle.

All external links are attached to the controller **10** through the MAC interface **19***a*, **19***b*. Any physical channel can be attached to the controller **10** through a formatter that adheres to the specification of the MAC receive and transmit data paths **19***a*, **19***b*. The MAC interface **19***a*, **19***b* and the controller **10** can operate in either half or full duplex modes. The crossbar interface **14**, **15** of the controller **10** is a full duplex interface that can establish connections and transmit to the crossbar **12** while simultaneously accepting connections and receiving frames from the crossbar.

In addition to the behavior of the links (i.e., data rate and duplex mode), the services that the controller **10** is to provide for the traffic on the attached link **11** must be specified. This aspect of link specification defines the controller **10** inbound and outbound forwarding/filtering procedures and additionally defines the format of the frames which the controller **10** will process. The controller **10** can attach to three types of links, particularly bridge links, internal switch links, and relay links.

A bridge link attaches a network segment **11** of an extended LAN to the net. That is, referring to FIG. **3**, what may be thought of as a higher level network **50** (referred to also as the "GigaNet") contains the crossbar switch **12** as one of its elements, and the controller **10** is the bridge between the FDDI link **11** and the net **50**. Of course, many other LANs such as the FDDI **11** or Ethernet LANs, etc., would be connected to the net **50** via other ports **13** of the crossbar switch **12**, or through other crossbar switches **12** in the net **50** (using other controllers **10** or their equivalent). Within the net **50**, the crossbar device **12** may function as classic point-to-point connector, or as set forth in the above-mentioned copending applications, may create a ring or bus, or may be create a mixed configuration of rings, buses and point-to-point connectors. The controller **10** will expect to receive and transmit frames in FDDI format at the MAC interface **19***a*, **19***b*, and will perform inbound lookups on the destination address, source address, and protocol fields, and translate the packet to switch frame format. The controller **10** will perform outbound checks and will perform protocol class lookup on the switch frames received from the crossbar **12** and translate the frames back to FDDI frame format prior to transmission to the MAC interface **19***a*, **19***b*.

An internal switch link connects switches in the net **50** together or attaches net end nodes to the net **50**. Again, the controller **10** can operate in this switch link mode. In this mode, the controller **10** will expect to receive and transmit frames in switch frame format at its MAC interface **19***a*, **19***b*. It will perform inbound checks on the switch frame and transmit the packet to the crossbar **12**. The controller **10** will perform outbound checks on the switch frame received from the crossbar and will transmit to the MAC interface **19***a*, **19***b* a switch frame.

A relay link uses a full duplex FDDI link to attach switches **12** in the net **50** together. The controller **10** functioning in this mode will expect to receive and transmit frames in a relay frame format at the MAC interface **19***a*, **19***b*, and here its operation is similar to switch link attachment and simply translates the packet format by adding or stripping a local packet header.

The type of link (bridge link, switch link, or relay link) that the controller **10** is attached to at the MAC interface **19***a*, **19***b* is specified by either decoding the frame control field of the received packet or by a type field in a control register **41** within the controller **10**.

Once the link type that the controller **10** is attached to is specified, the inbound and outbound processes **27**, **29**, **31**, and **33** know the format of the frames that will be processed and the forwarding procedures it is to perform.

Referring to FIG. **4**, the three different frame formats that may exist at the MAC interface **19***a*, **19***b* are illustrated, including an FDDI frame format **54**, a switch frame format **55** (which encapsulates the FDDI frame **54**), and a relay frame format **56** (which encapsulates the switch format **55**).

The original FDDI frame format **54** is of the form specified by the FDDI protocol, and will not be described in detail. Within the FDDI format **54** are an FDDI header **57**, an 802.2 LLC header **58**, an information field **59** (the payload), and an FDDI CRC field **60**. Various other elements such as starting and ending delimiters are present. Within the header **57** are 48-bit source and destination addresses **61** and **62**, and these are used in the address lookups as described herein. Other special addresses or identification specified in the protocol are contained in the LLC header **58**, such as the ones mentioned above (DSAP and SNAP). Each of these addresses is extracted by the controller **10** and

used for lookups and filtering as described herein. The FDDI frame format **54** is found at the MAC interface **19***a*, **19***b* of the controller **10**. When attached to bridge links, this will be the format of the frames received by the IR process **27** and will be the format of frames transmitted by the OT process **33**.

The switch frame format **55** of FIG. **4** is seen to be merely the original FDDI frame **54** to which is attached at the beginning an added header **63**, also referred to as the "gigaheader". This added header **63** is described in detail below and is an important feature of the invention. The switch frame format **55** can appear on both the MAC interface **19***a*, **19***b* and the crossbar interface **14**, **15**. At the crossbar interface **14**, **15**, this is the common packet format used by the switch ports **13**. It is the format transmitted by the IT process **29**, and it is the format received by the OR process **31**.

The relay frame format **56** of FIG. **4** consists of a switch frame format **55** (including FDDI format **54** and added header **63**), to which is added a local FDDI header **64** and a local FDDI CRC field **65**. The local FDDI header **64** is of the same specification as the FDDI header **57**, and is defined in the FDDI protocol. This type of frame **56** is found at the MAC interface **19***a*, **19***b* of the controller **10** when it is attached to a relay link.

Referring to FIG. **5**, the added header **63** (or "gigaheader") is a thirteen byte (104-bit) field that is present in switch frames **55** and relay frames **56**. It contains information that is generated by the inbound receive (IR) process **27** when a packet first enters the net **50** and is used to direct the packet to its destination within the net **50**. This field **63** defines how the packet is to be serviced within the net **50**. The packet processing within the net **50** is performed by examining the fields of this added header **63**. Each of the fields in the added header **63** will now be described.

The service class field **65** is a four-bit field that specifies the service class for the packet. For the receivers of the controller **10**, the service class is used to determine the queue number in memory **21** that a packet is to be placed on by doing a lookup using certain internal registers **41**. On the crossbar **12**, the service class is also used for connection queue servicing.

The destination switch field **66** is a 12-bit field representing the switch number (i.e., local address) of the final destination of the packet within the net **50**. When a packet is received from an FDDI link **11**, the IR process **27** performs a lookup operation to derive the destination switch value for the packet. A 12-bit field allows 4096 ports; a single crossbar **12** has only thirty-six ports in one embodiment, so the need for a 12-bit address is because a number of the crossbars **12** may be within a net **50**, and to allow for expansion to larger crossbars and larger nets **50**. Certain switch values are reserved: 02_{hex} and 03_{hex} indicate that the final destination is unknown to the IR process **29**, and 00_{hex} and 01_{hex} indicate that the final destination address is to be filtered at this port. Other values derived for this field will direct the packet to its destination within the switch.

The decapsulate bit **67**, when set to one, indicates that this packet is to be decapsulated when it goes out onto a link that supports both relay and bridge frames. The outbound processes **31** and **33** of the controller **10** will sample this bit to make a determination of the transmission format to use. Usually this bit is set to zero.

The destination link **68** is a 7-bit field representing the logical link number of the final destination of the packet, except when switch field **66** equals 00_{hex}, in which case this field **68** is the physical link number of the port that is to receive the message. The IR process **29** will generate this field when constructing the added header **63** for frames received from the FDDI interface **19***a*, **19***b*. When enabled, the OR process **31** will compare this field to its own logical link number to verify that the crossbar connection is established in the correct port. Logical link numbers 00-to-07_{hex} are reserved for addressing the switch control processors x for switches **12** in the net **50**.

The protocol class field **69** of FIG. **4** is an 8-bit field representing the protocol class assigned to the packet within the net **50**. The IR process **29** will generate this field **69** when constructing the added header **63** for frames received from the FDDI interface **19***a*, **19***b*. The OR process **31** will sample this field **69** for frames received from the crossbar **12** to determine if the packet is enabled to be forwarded in the outbound direction. The values F0-to-FD_{hex} of this field **69** are reserved for special net-wide protocols such as inter-switch control messages, initialization protocols, implementation of protocol trapping, and frames destined for a line control processor **25**. The values FE and FF_{hex} are used as filter protocol values. All other numbers are user-assignable.

The source link type field **70** is a four bit field representing the type of data link that sourced the packet that the current added header **63** is attached to. This field **70** can be used to specify additional translation processing during the forwarding of the frame. The controller **10** does not use this field during packet processing.

The source switch field **71** is a 12-bit field representing the switch number of the original source of this packet in the net **50**. The values 00-to-03_{hex} are illegal on all data links, and certain ones are reserved; all other numbers are user assignable.

The source link field **72** is a 7-bit field representing the logical link number of the original source of this packet in the net **50**. The values 00-to-07_{hex} are reserved for use by the switch control processor x.

The previous hop link field **73** is a 7-bit field representing the physical link number of the previous hop link. Thus it represents the link number that is currently sending the packet to the crossbar **12**.

The translation status field **74** is a 4-bit field which stores the result of the automatic translation performed on the current packet. The status may indicate whether the frame is to be forwarded to its destination, monitored, trapped, logged, or whether the source address of the frame was "new."

The type of time field **75** of FIG. **4** is a 1-bit field representing the format for the timestamp in the current packet. When this bit is "0" the timestamp field represents absolute time, and when set to "1" the time stamp field is interpreted as a relative time to live.

The end-of-packet stream field **76** is a 1-bit field which, when set to one, indicates that this packet is the last packet in a packet stream. Otherwise, when set to zero, it indicates the first or a middle packet in a packet stream.

The system port field **77** is a 1-bit field which, when set to one, indicates that this packet is being sent from a system port (e.g., a switch control processor x port). This bit can be used during outbound packet processing to restrict certain types of packet to originate only from system ports.

The copied frame field **78** is a 1-bit field which, when set to one, indicates that a copy frame is being sent to

the crossbar **12**. Copy frames are used to send exception traffic to engines within the switch **12**.

The outbound congestion bit **79** is a 1-bit field that is used to associate outbound congestion information with the added header **12** of the current packet.

The inbound congestion bit **80** is a 1-bit field that is used to associate inbound congestion information with the added header **12** of the current packet.

The timestamp field **81** is a 16-bit field representing a value from 00 to 65535 which is either the lowest 16-bits of an absolute time that the packet must be destroyed, or the number of time units the packet has to live. One time unit is 10-milliseconds. The timestamp is generated by the inbound receive process **27**.

Reserved field **82** represents 4-bits reserved for future specification. These should be sent as zeros and otherwise ignored.

The header check field **83** is an 8-bit CRC covering the previous 96-bits of the 104-bit added header **63** (the "gigaheader").

The frame translation performed within the controller **10** will now be described. The controller **10** forwards inbound and outbound frames and performs the necessary translation to generate the desired packet formats **54–56**, e.g., adding the header **63** if necessary. The forwarding and translation procedures that are performed by the processes **27**, **29**, **31**, and **33** of the controller **10** are a function of the link type that the controller **10** is attached to and of course a function of whether the packet is inbound or outbound.

For inbound processing of a packet received where the MAC link type is "bridge" and the MAC frame format is FDDI format **54**, with the crossbar switch **12** frame format being "switch" format **55**, the actions performed are:

Perform a lookup of the frame control field of the received packet. Process the destination address **61**, source address **62**, and protocol fields of the inbound FDDI frame **54**. Determine if the packet is to be forwarded/monitored and if there are any exception conditions. Generate the service class field **65**, the destination switch field **66**, the destination link field **68**, and the protocol class field **69**. Add a timestamp field **82** and the remaining header **63** fields to form a switch frame **55**. Queue the packet in memory **21**.

For inbound processing of a packet received where the MAC link type is "bridge" and the MAC frame format is "bridge" format **55**, with the crossbar switch **12** frame format being "switch" format **55**, the actions performed are:

Perform a CAM lookup in CAM **23** using the protocol class field **69** of the received header **63**. Check if the packet is to be monitored or trapped. Clear the system port bit **77** and check the timestamp **82** and checksum **83** for the header **63**. Add the updated header **63** to the frame.

For inbound processing of a packet received where the MAC link type is "relay" format **56** and the MAC frame format is "relay" format **56**, with the crossbar switch **12** frame format being "switch" format **55**, the actions performed are:

Perform a lookup of the frame control field of the received packet. The local header **64** is checked and removed from the relay frame **56**. The procedure then proceeds as specified by the switch link.

For outbound processing of a packet received where the MAC link type is "bridge," the crossbar switch **12** frame format is "switch" format **55**, and the MAC frame format is FDDI format **54**, the actions performed are:

Check added header **63** checksum **83** for the received frame. Check the fields of the header **63**, including destination switch field **66**, destination link field **68**, and system port bit **77**. Perform protocol class lookup for field **69**. Verify that the timestamp **82** is still good, if not discard the frame. Strip header **63** from the packet to realize an FDDI frame format **54**.

For outbound processing of a packet received where the MAC link type is "switch," the crossbar switch **12** frame format is "switch" format **55**, and the MAC frame format is "switch" format **55**, the actions performed are:

Check the added header **63** checksum **83** for the received frame. Check the fields of the header **63**, including destination switch field **66**, destination link field **68**, and system port bit **77**. Perform protocol class lookup for field **69**. Verify that the timestamp **82** is still good, if not discard the frame.

For outbound processing of a packet received where the MAC link type is "relay," the crossbar switch **12** frame format is "switch" format **54**, and the MAC frame format is "relay" format **56**, the actions performed are:

The local header **64** is added to the switch frame **55**. The procedures then proceed just as specified by the switch link, i.e., the frame is treated as if it were an FDDI frame **54**.

During the frame forwarding procedure, the controller **10** makes use of several lookup mechanisms to process the fields of the frame. These include the CAM lookup, the hash lookup, a class specifier lookup, and a protocol class lookup.

The CAM lookup consists of applying an address or value to the CAM **23**, and getting back an indication of whether or not a match is found in the CAM **23**. The CAM **23** is of limited size due to the cost (number of transistors) inherent in constructing a CAM. In an example embodiment, the CAM **23** holds 256 entries. Thus, there can be 256 values (addresses, etc.) that are searched for in a CAM lookup. The IR process **27** uses the CAM lookup to perform matches using the various fields of the received frame **54**, **55**, or **56** as keys. Variable width key matching is provided, up to six bytes wide, and this is specified by the IR process **27** during the parse of the frame, and a match result return an associated data value that will be used by the IR process **27**. This CAM match can be used to find the frame control, destination address, source address, DSAP and SNAP fields of bridge and relay link packets. It can also be used to find the protocol class field **69** of the added header **63** of the switch link packets **55**.

A protocol class lookup is performed by the OR process **31** to determine if the protocol class field **69** in the added header **63** for a frame received from the crossbar switch **12** via line **15** indicates that this protocol class is enabled to be forwarded outbound.

The hash lookup is a feature of the invention disclosed in the above-mentioned copending application Ser. No. 07/964,738. A hash lookup is used by the IR process **27** (in conjunction with a CAM lookup) to process the 48-bit destination and source fields **61** and **62** of received frames. The 48-bit addresses **61** and **62** may be globally unique addresses, i.e., each station may have an address that is unique to that station. The ad-

dress range covers 2^{40} or approximately 3×10^{14} (300 trillion) unique values, so an address may never need to be duplicated in any foreseeable systems. Within an extended network, however, the number of unique addresses needed is usually only a few thousands or at most tens of thousands. Therefore, a table of all of the addresses being used in a network at a given configuration only contains, for example, a maximum of 64K or 2^{16}, entries, which would use merely a 16-bit address. For this reason, the 48-bit globally-unique address **61** or **62** is hashed, producing a 16-bit locally-unique address. An incoming address field **85** as seen in FIG. **5** is subjected to a hash function **86** to produce another 48-bit value **87**, then a 16-bit part **88** of this 48-bit value **87** (the least significant 16-bit field) is used to index into a 64K-entry table in memory **21**, this being referred to as the hash table **89**. Each word of the memory **21** is 56-bits wide, so a word can contain three of these hash entries, in three "hash buckets" **90** for each word **91**, as seen in FIG. **6**. When a hashed address **87** is generated during the processing of packets, the 16-bit index **88** is used to select one of the 22K words **91** in the hash table **89** and one of the three hash buckets **90** at this index. Each bucket **90** is an 18-bit field containing a 3-bit value **92** indicating the bucket size, one to seven entries, and a 15-bit field **93** acting as a translation table pointer. The 16-bit input **88** indexes to a word **91** depending upon its lower order bits and one-of-three buckets **90** in a word **91** depending upon its higher ordered bits, indexed for one-of-22K selection of the word **91** in the table. The translation table pointer **93** returned by the hash bucket **90** in the hash table **89** is used to select a breadth-first balanced binary tree as illustrated in FIG. **7**. The trees are stored in a translation table **94** in memory **21**, and each tree has between one and seven entries, as indicated by the size field **92**. The binary tree cannot cross a block boundary in the translation table **94** in memory **21**. The ordering of entries **96** in a breadth-first balanced binary tree for various table sizes is illustrated in FIG. **7**. Note that there are from one to seven entries (each entry a 48-bit address) in a tree, ordered such that

$$A<B<C<D<E<F<G$$

Each entry **96** contains 32-bit hash remainder field **97** that is left after using the low-order 16-bit value **88** from the hashed 48-bit address to store an address. To traverse the tree, the IR process fetches the first entry **96**, and the 32-bit value **97** in this entry is compared to the upper 32-bit value **97** of the incoming hashed address **87** (from the packet being evaluated). If the two values match, the remaining fields of the entry **96** in the translation table **94** provide the IR process **27** with the information it needs for processing that field of the packet. Otherwise, the IR process will continue traversing the binary tree by selecting the next entry as a function of whether the has remainder is less than or greater than the current entry **96**. The hash remainders **97** are compared, and if the incoming value **97** is less than the stored value **97** the left branch **98** is picked, while if greater than the right branch **99** is picked. The tree traversal will continue until a match is found until a match is found or the tree is exhausted. For each hash lookup and tree traversal, up to four memory requests to external memory **21** will be made by the IR process **27**. One memory request is required to obtain the hash bucket **90**, and then up to three more memory requests are needed to traverse the binary tree of FIG. **7** (e.g., D—B—C of the 7-entry tree). The arbiter of the PM process **35** provides priority service for memory requests associated with this search process in the IR process **27**, and will allocate up to every other memory cycle for the IR lookup during packet reception.

The address lookup procedure used by the IR process **27**, as just described, is also illustrated in FIG. **8**. The 48-bit destination address **61** from an incoming packet is simultaneously subjected to the hash function **86** and a CAM match; if a CAM match is found in CAM **23**, an index into the translation table **94** is immediately generated as indicated by the line **100**. If not, the 16-bit field **88** of the hashed address **87** is used to index into the hash table **89** to select a hash bucket **90**, and this selected hash bucket contains a pointer **93** into the translation table **94**. The binary tree found in the translation table **94** will have a size indicated by the 3-bit field **92**. Each entry **96** in the tree has a 32-bit remainder field **97** which is compared at comparator **101** with the 32-bit remainder in the hashed address **87** to either find a match or progress through the tree from a maximum of three fetches.

When the hash function **86** is implemented to build the hash table **89** and translation table **94**, as upon initialization of a network and whenever the network is reconfigured by adding stations or bridges, there is a small but finite probability that more than seven addresses will hash to the same bucket. If this occurs, the address producing the collision in trying to load the table will be stored in the CAM **23**. Perhaps thirty-two or so of these collision addresses can be stored in the 256-entry CAM **23** without degrading its other functions. Thus, since every address is compared with the CAM entries anyway (to filter for multicast messages, SNAP filtering, etc.), and this compare in the CAM is done in parallel with the hash function, the CAM compare is without cost in time or new circuitry. Were this outlet not available, the occurrence of more than seven entries for a hash bucket would result in the necessity to chose a new hash function and recalculate all the addresses to rebuild the hash table **89** and the translation table **94**; this would require a period of down time where the controller **10** could not respond to incoming traffic since the hash and translation tables would be unavailable, and processor time would be monopolized by the rebuild. The 48-bit source and destination addresses are often not randomly selected, but instead are assigned in blocks or patterns and otherwise increase the likelihood of hashing to the same bucket. Alternatively, one way of avoiding the consequences of having to rebuild the hash table **89** is to generate two hash tables in memory **21** using two different hash functions, so if the one being used generates a collision (more than seven in a hash bucket), processing switches to the alternate hash table where a collision is very unlikely; this requires extra size for memory **21**, however. The availability of the CAM lookup for instances where there are a few collisions in loading the hash buckets has the advantage of not imposing any significant burden in time or in memory usage.

While the invention has been described with reference to a specific embodiment, the description is not meant to be construed in a limiting sense. Various modifications of the disclosed embodiment, as well as other embodiments of the invention, will be apparent to persons skilled in the art upon reference to this description. It is therefore contemplated that the appended claims will cover any such modifications or embodiments which fall within the true scope of the invention.

What is claimed is:

1. A packet data communications network comprising:

a first network segment having a plurality of stations, one of said stations sending a message packet onto said first network segment of a first format; said first format including a first header and a data field with network destination address in said communications network;

a first network transfer device having an input connected to said first network segment to receive said message packet and having an output; the first network transfer device applying a second header to said message packet, said second header including a switching address translated from said destination address, said second header further including local status information and a plurality of status fields to indicate a message packet servicing;

a switching device having a plurality of ports, a first of said ports being connected to said output of said first network transfer device; the switching device receiving said message packet with said second header and sending said message packet with said second header to a second port as selected by said switching address, and in response to said local status information and the plurality of status fields;

a second network transfer device having an input connected to said second of said ports of said switching device and having an output connected to a second network segment, the second network transfer device receiving said message packet via said switching device to forward to said second network segment; the second network transfer device removing said second header from said message packet.

2. A network according to claim 1 wherein said network has a plurality of links, and each of said links is assigned a link number, and said second header includes a link number for a source of said message packet and a link number for a destination of said message packet.

3. A network according to claim 1 wherein said destination address contains N bits, and said switch address contains M bits, where N and M are integers and N >>M.

4. A network according to claim 3 wherein said packet includes a network source address of N bits, and said added header contains a source switch address of M bits translated from said network source address.

5. A network according to claim 4 wherein said switching device is a crossbar switch.

6. A network according to claim 5 wherein said first network segment is a serial FDDI link, and said ports are parallel ports.

7. A network according to claim 6 wherein said added header contains a service class field, and said switching device processes said packet in response to said service class field.

8. A network according to claim 7 wherein said added header contains a protocol class field, and said switching device processes said packet in response to said protocol class field.

9. A network according to claim 8 wherein said added header contains a status field indicating local congestion, and said switching device processes said packet in response to said status field.

10. A method of operating a packet data communications network, the network including a first network segment having a plurality of stations and a second network segment having a plurality of stations, and including a switching device interconnecting said first and second segments, comprising the steps of:

sending from one of said stations of said first network segment a message packet of a first format onto said first network segment; said first format including a first header and a data field with a network destination address in said communications network;

receiving said message packet at a first network transfer device having an input connected to said first network segment; the first network transfer device applying a second header to said message packet, said second header including a switching address translated from said destination address, said second header further including local status information and a plurality of status fields to indicate a message packet servicing;

receiving at said switching device said message packet with said second header and sending said message packet with said second header to a port of said switching device as selected by said switching address, and in response to said local status information and the plurality of status fields;

receiving said message packet at said second network transfer device via said switching device and forwarding said message packet to said second network segment; the second network transfer device removing said second header from said message packet.

11. A method according to claim 10 wherein said network has a plurality of links, and each of said links is assigned a link number, and inserting in said second header a link number for a source of said message packet and a link number for a destination of said message packet.

12. A method according to claim 10 wherein said destination address contains N bits, and said switch address contains M bits, where N and M are integers and N >>M.

13. A method according to claim 12 wherein said packet includes a network source address of N bits, and inserting in said added header a source switch address of M bits translated from said network source address.

14. A method according to claim 13 wherein said switching device is a crossbar switch.

15. A method according to claim 14 including transmitting on said first network segment by a serial FDDI method, and said port of said switching device being a parallel port.

16. A method according to claim 15 including inserting in said added header a service class field, and said switching device processes said packet in response to said service class field.

17. A method according to claim 16 including inserting in said added header a protocol class field, and said switching device processes said packet in response to said protocol class field.

18. A method according to claim 17 including inserting in said added header a status field indicating local congestion, and said switching device processes said packet in response to said status field.

19. A packet data communications network comprising:

a first network segment having a plurality of stations, one of said stations sending a message packet onto said first network segment of a first format; said first format including a first header and a data field

with network destination address in said communications network;

a first network transfer device having an input connected to said first network segment to receive said message packet and having an output; the first network transfer device applying a second header to said message packet, said second header including a switching address translated from said destination address, said second header including local status information and a service class field;

a switching device having a plurality of ports, a first of said ports being connected to said output of said first network transfer device; the switching device receiving said message packet with said second header and sending said message packet with said second header to a second port as selected by said switching address, and in response to said local status information and said service class field;

a second network transfer device having an input connected to said second of said ports of said switching device and having an output connected to a second network segment, the second network transfer device receiving said message packet via said switching device to forward to said second network segment; the second network transfer device removing said second header from said message packet.

20. A packet data communications network comprising:

a first network segment having a plurality of stations, one of said stations sending a message packet onto said first network segment of a first format; said first format including a first header and a data field with network destination address in said communications network;

a first network transfer device having an input connected to said first network segment to receive said message packet and having an output; the first network transfer device applying a second header to said message packet, said second header including a switching address translated from said destination address, said second header including local status information and a protocol class field;

a switching device having a plurality of ports, a first of said ports being connected to said output of said first network transfer device; the switching device receiving said message packet with said second header and sending said message packet with said second header to a second port as selected by said switching address, and in response to said local status information and said protocol class field;

a second network transfer device having an input connected to said second of said ports of said switching device and having an output connected to a second network segment, the second network transfer device receiving said message packet via said switching device to forward to said second network segment; the second network transfer device removing said second header from said message packet.

21. A packet data communications network comprising:

a first network segment having a plurality of stations, one of said stations sending a message packet onto said first network segment of a first format; said first format including a first header and a data field with network destination address in said communications network;

a first network transfer device having an input connected to said first network segment to receive said message packet and having an output; the first network transfer device applying a second header to said message packet, said second header including a switching address translated from said destination address, said second header including local status information and a local congestion status field;

a switching device having a plurality of ports, a first of said ports being connected to said output of said first network transfer device; the switching device receiving said message packet with said second header and sending said message packet with said second header to a second port as selected by said switching address, and in response to said local status information and said local congestion status field;

a second network transfer device having an input connected to said second of said ports of said switching device and having an output connected to a second network segment, the second network transfer device receiving said message packet via said switching device to forward to said second network segment; the second network transfer device removing said second header from said message packet.

22. A method of operating a packet data communications network, the network including a first network segment having a plurality of stations and a second network segment having a plurality of stations, and including a switching device interconnecting said first and second segments, comprising the steps of:

sending from one of said stations of said first network segment a message packet of a first format onto said first network segment; said first format including a first header and a data field with a network destination address in said communications network;

receiving said message packet at a first network transfer device having an input connected to said first network segment; the first network transfer device applying a second header to said message packet, said second header including a switching address translated from said destination address, said second header including local status information and a service class field;

receiving at said switching device said message packet with said second header and sending said message packet with said second header to a port of said switching device as selected by said switching address, and in response to said local status information and said service class field;

receiving said message packet at said second network transfer device via said switching device and forwarding said message packet to said second network segment; the second network transfer device removing said second header from said message packet.

23. A method of operating a packet data communications network, the network including a first network segment having a plurality of stations and a second network segment having a plurality of stations, and including a switching device interconnecting said first and second segments, comprising the steps of:

sending from one of said stations of said first network segment a message packet of a first format onto said first network segment; said first format includ-

ing a first header and a data field with a network destination address in said communications network;

receiving said message packet at a first network transfer device having an input connected to said first network segment; the first network transfer device applying a second header to said message packet, said second header including a switching address translated from said destination address, said second header including local status information and a protocol class field;

receiving at said switching device said message packet with said second header and sending said message packet with said second header to a port of said switching device as selected by said switching address, and in response to said local status information and said protocol class field;

receiving said message packet at said second network transfer device via said switching device and forwarding said message packet to said second network segment; the second network transfer device removing said second header from said message packet.

24. A method of operating a packet data communications network, the network including a first network segment having a plurality of stations and a second network segment having a plurality of stations, and including a switching device interconnecting said first and second segments, comprising the steps of:

sending from one of said stations of said first network segment a message packet of a first format onto said first network segment; said first format including a first header and a data field with a network destination address in said communications network;

receiving said message packet at a first network transfer device having an input connected to said first network segment; the first network transfer device applying a second header to said message packet, said second header including a switching address translated from said destination address, said second header including local status information and a local congestion status field;

receiving at said switching device said message packet with said second header and sending said message packet with said second header to a port of said switching device as selected by said switching address, and in response to said local status information and said local congestion field;

receiving said message packet at said second network transfer device via said switching device and forwarding said message packet to said second network segment; the second network transfer device removing said second header from said message packet.

* * * * *

Exhibit 3

US006128665A

United States Patent [19]

Iturralde

[11] Patent Number: 6,128,665

[45] Date of Patent: Oct. 3, 2000

[54] **SYSTEM FOR BROADCASTING MESSAGES TO EACH OF DEFAULT VLAN PORTS IN SUBSET OF PORTS DEFINED AS VLAN PORTS**

[75] Inventor: **Carol E. Iturralde**, Framingham, Mass.

[73] Assignee: **Cabletron Systems, Inc.**, Rochester, N.H.

[21] Appl. No.: **08/774,541**

[22] Filed: **Dec. 30, 1996**

[51] **Int. Cl.**[7] **G06F 13/00**

[52] **U.S. Cl.** **709/238**; 709/242; 709/244

[58] **Field of Search** 370/401, 402; 395/500, 200.68, 200.72, 200.74

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,823,338	4/1989	Chan et al. .	
5,394,402	2/1995	Ross	370/402
5,613,069	3/1997	Walker	395/200.68
5,684,800	11/1997	Dobbins et al.	370/401
5,734,865	3/1998	Yu	395/500
5,740,171	4/1998	Mazzola et al.	370/401
5,742,604	4/1998	Edsall et al.	370/401
5,752,003	5/1998	Hart	395/500

OTHER PUBLICATIONS

Saunders, S., "Building Virtual LANS on a Real–World Budget Lanart's Segway Works with Ethernet Switches to Deliver Virtual LANS Powers at a Low Cost", Data Communications, vol. 24, No. 13, Sep. 21, 1995, pp. 39/40 XP000526194.

"Virtual LANS Get Real Ethernet Switch Makers are Taking the Lead in Deploying Virtual LANS Across Campus Networks," Data Communications, vol. 24, No. 3, Mar. 1, 1995.

Anderson, J. K., "Virtual LANS Take Network to Next Level", Computer Technology Review, vol. 16, No. 9, Sep. 1996, pP. 12, 14 XP00042987.

Morency et al., "VLANS: Can LaYer 3 Save the Day", Business Communications Review, vol. 26, No. 12, Dec. 1996, pp. 47–50 XP0020642987.

McGibbon, "Virtual LANS Come of Age" Telecommunications (International Edition), vol. 30, No. 6, Jun. 1996, pp. 48–52, XP002062191.

Networking Solutions Center, The Virtual LAN Technology Report, David Passmore and John Freeman. Dec. 9, 1996.

Strategic Directions, 3Com Transcend VLANS (Leveraging Virtual LAN Technology to Make Networking Easier), VLAN Strategic Directions Paper. Dec. 9, 1996.

Virtual LAN Communications (Statement of Direction: Cisco VLAN Roadmap, White Paper: Cisco IOS VLAN Services, Technology Brief: VLAN Interoperability). Dec. 9, 1996.

Cisco VLAN Roadmap (White Paper: Cisco IOS VLAN Services, White Paper: Virtual LAN Communications, Technology Brief: VLAN Interoperability). Dec. 9, 1996.

VLANs and Routers (Mixing routers, switches and VLANS) Dec. 9, 1996.

White Paper, Switching Paradigms (Everything Has Changed Except the Network) Dec. 9, 1996.

Fundamentals of Switching, Preface, Table of Contents, Chapters 1–5 (12 pages) Dec. 9, 1996.

Cisco Catalyst 5000 Modular Multilayer–Capable Switching System, Product Announcements.

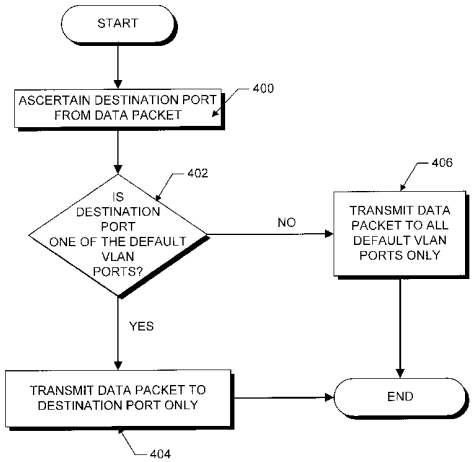
Anderson, J.K. ("Virtual LANs Take To Next Level", Computer technology Review, vol. 16, No. 9, Sep. 1996, pp. 12–14.

Primary Examiner—Krisna Lim
Assistant Examiner—Philip B. Tran
Attorney, Agent, or Firm—Wolf, Greenfield & Sacks, P.C.

[57] **ABSTRACT**

A data transmission network having a port-based default VLAN that limits flooding to other VLANs. The default VLAN receives a data packet, ascertains the destination address of the packet, and then determines if the destination port is one of the default VLAN ports. The data packet is transmitted to the destination port if it is one of the default VLAN ports, or to each of the default VLAN ports if the destination port is not one of the default VLAN ports. The data packet is not transmitted to any other non-default VLAN port.

8 Claims, 4 Drawing Sheets



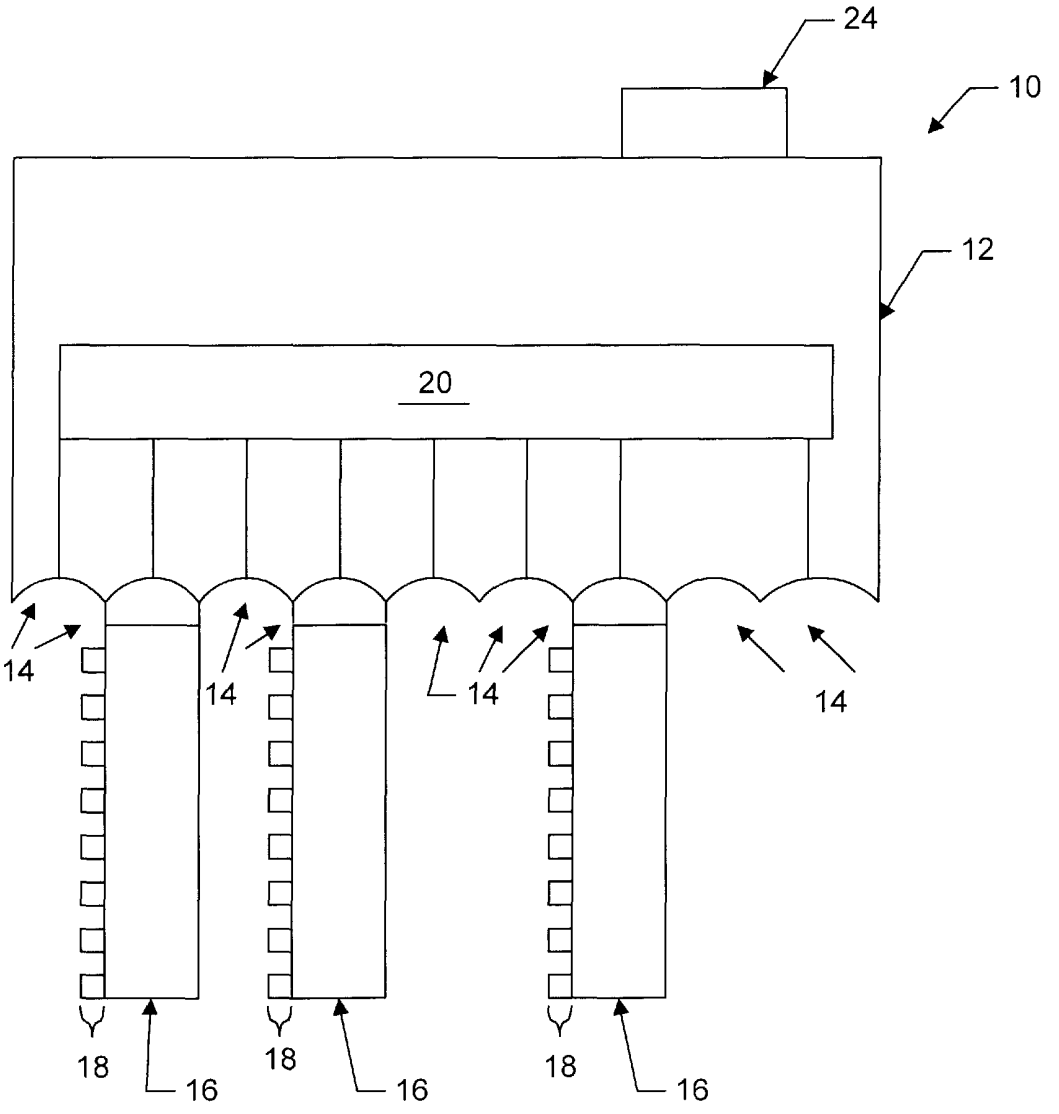


FIG. 1

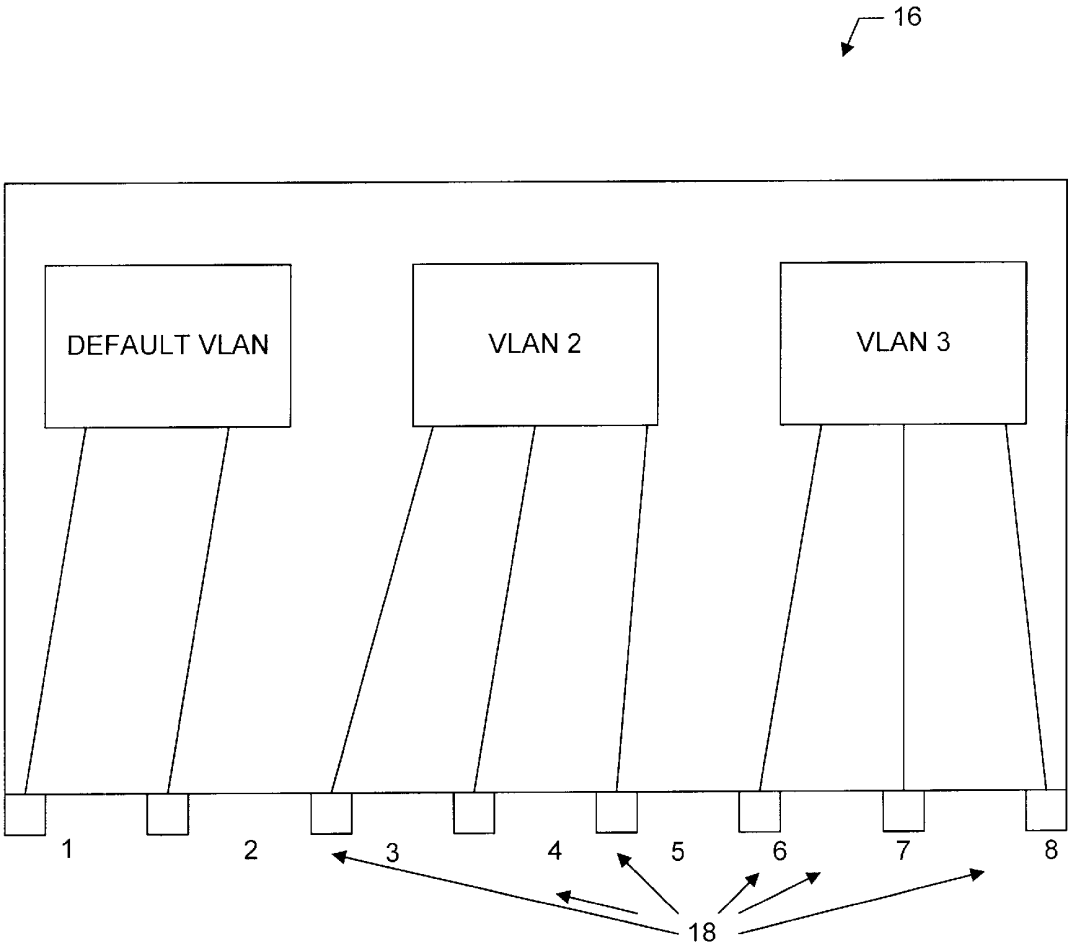


FIG. 2

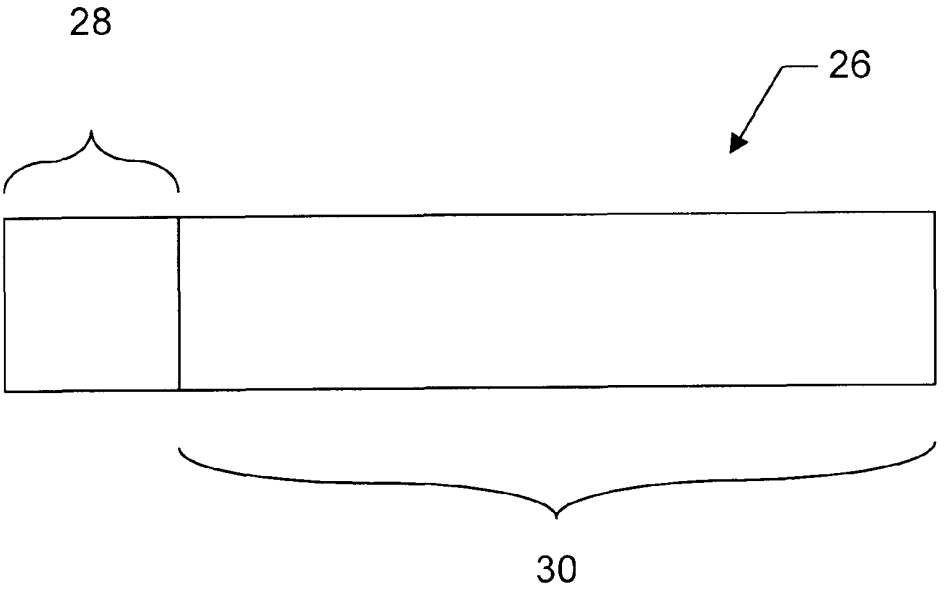


FIG. 3

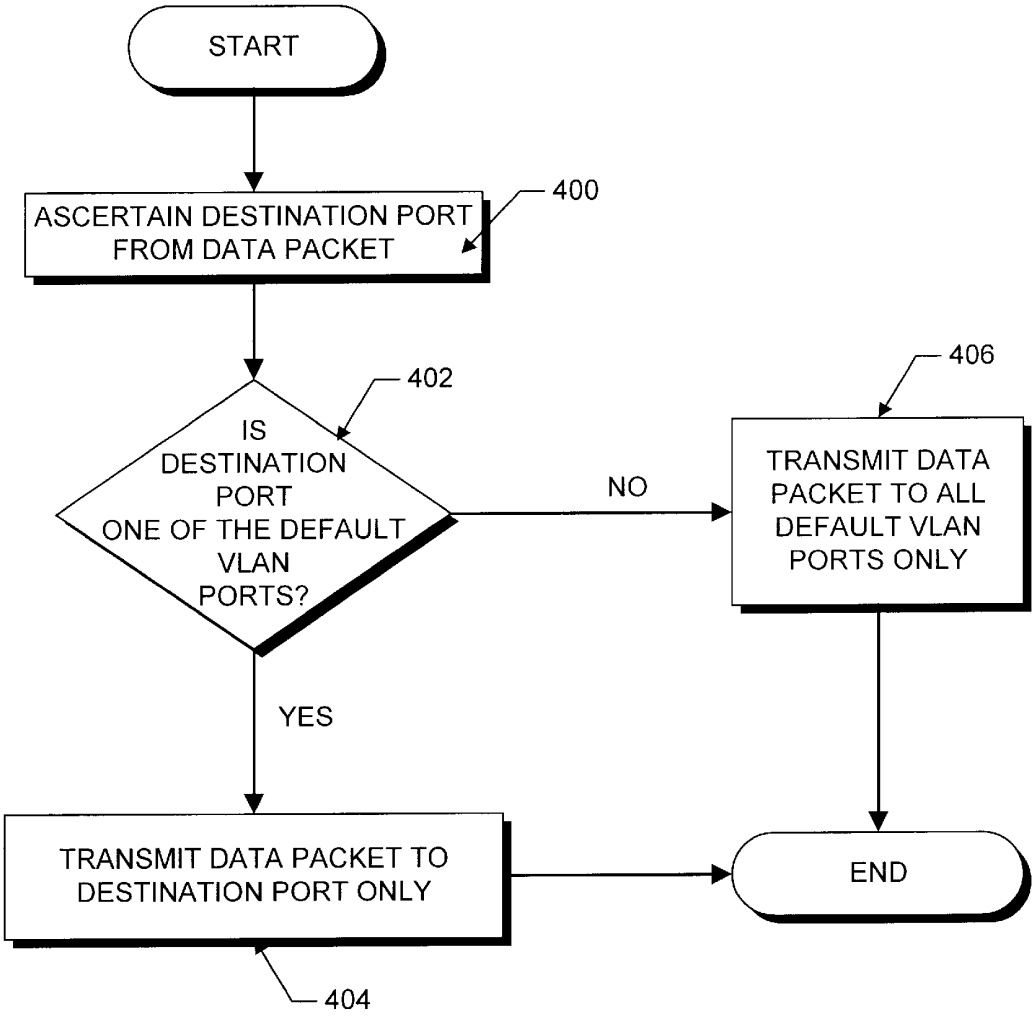

START
ASCERTAIN DESTINATION PORT FROM DATA PACKET
400
IS DESTINATION PORT ONE OF THE DEFAULT VLAN PORTS?
402
NO
TRANSMIT DATA PACKET TO ALL DEFAULT VLAN PORTS ONLY
406
YES
TRANSMIT DATA PACKET TO DESTINATION PORT ONLY
404
END

FIG. 4

SYSTEM FOR BROADCASTING MESSAGES TO EACH OF DEFAULT VLAN PORTS IN SUBSET OF PORTS DEFINED AS VLAN PORTS

FIELD OF THE INVENTION

This invention generally relates to data transmission networks and, more particularly, to virtual local area networks.

BACKGROUND OF THE INVENTION

A data network typically includes several nodes connected together by a data transport medium. One common method of transmitting data between the nodes is to break the data up into discrete "packets" of data. Packets can be transported over the medium by any one of a variety of transport techniques. In applications utilizing packetized data, data to be transported first is broken up into discrete packets of data, then transmitted through the network medium, and finally reassembled at a destination node. In accordance with current packet protocol, each packet generally comprises a header and an information field. The header contains the information used to transport the cell from one node to the next while the packet data is contained in the information field. Among other information in the header is the destination address of the data packet.

A local area network (i.e., "LAN") is a type of local data network commonly used in a single office or building. LANs are an efficient mechanism for maximizing use of network resources by members of the LAN. Simple LANs typically include two or more nodes (e.g., a server, computer, printer, or other resource) that are interconnected by a common physical connection such as, for example, a hub. Data switches also may be connected to the hub for directing data traffic and for connecting the LAN to other data networks.

LANs can be inconvenient and expensive to maintain. For example, moving a user to another location within a relatively large office building often requires that the LAN be rewired and reconfigured. This can be cumbersome and expensive. The art has responded to this problem by developing virtual local area networks (i.e. "VLANs").

A VLAN is generally defined as a group of nodes interconnected by software to form a single logical broadcast domain. VLANs may be connected to nodes that are members of any number of physical LAN segments. Among many advantages, VLANs enable network administrators to create logical groupings of users and network resources, thereby allowing remote users and resources to appear as if they are members of a single LAN. This enables companies and other organizations to build dynamic, flexible, and distributed LANs, thus simplifying physical moves of a user in a network.

VLANs may be formed by defining logical groups of users within the VLAN. One such VLAN, known as a "port-based" VLAN, defines the VLAN as a collection of switch ports on one or more switches across a hub. Users connected to those defined switch ports therefore are members of the defined VLAN. Broadcast messages directed to that VLAN may be transmitted through the defined switch ports only. Known port-based VLANs typically are implemented on a switch to include a default VLAN, in addition to other VLANs that may be formed on the switch. During manufacture, the default VLAN is defined as every port on a single switch. The number of switch ports defining the default VLAN decreases, however, as ports on the switch are used for defining other VLANs. Accordingly, on an exemplary eight-port switch having a first VLAN defined by ports one and two, the default VLAN will be defined by remaining ports three through eight.

Known port-based default VLANs have data leakage problems that can compromise the security of data transmitted across a network. Specifically, port-based default VLANs transmit a data packet to every switch port when that packet is received by the default VLAN and is destined for a port that is not in the default VLAN. Continuing with the above example, a data packet received on a port defining the default VLAN (i.e., one of ports three through eight) and destined for another port also on the default VLAN will be transmitted to the destination port only. In the event that the data packet was destined for a port on the first VLAN (i.e., port one or two), however, the packet would be transmitted to all of the ports on the switch, thus creating the above mentioned security problem.

Accordingly, it would be desirable to provide a port-based default VLAN that prevents such leakage problems between VLANs. It is among the general objects of this invention to provide such a device and method.

SUMMARY OF THE INVENTION

In accordance with the principles of the invention, a port-based default VLAN is provided that prevents leakage problems across VLANs. To that end, the default VLAN includes means for transmitting data received by the default VLAN to ports defining the default VLAN only. No other ports on the switch will receive a data packet that was received on a port defining the default VLAN.

In accordance with another aspect of the invention, each of the ports on a plurality of switches connected to a hub are configured, during manufacture, to define a default VLAN spanning the plurality of switches. To that end, the default VLAN includes a bus in the hub, an enable switch for electrically connecting each of switches to the bus, and means for defining each of the switch ports as the default VLAN.

It is among the objects of the invention to provide port-based default VLAN and method that prevents leakage across the ports of a switch.

It is another object of the invention to provide a port-based default VLAN that, is configured, during manufacture, to span a plurality of switches connected to a hub.

BRIEF DESCRIPTION OF THE DRAWINGS

The above and further advantages of the invention may be better understood by referring to the following description in conjunction with the accompanying drawings and which:

FIG. **1** is a block schematic diagram of a partial data network assembly for implementation of the invention;

FIG. **2** is a block schematic diagram of a switch that forms a port-based, default VLAN;

FIG. **3** is a schematic diagram of a data packet; and

FIG. **4** is a flow chart that specifies the method used for preventing leakage from the default VLAN.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

FIG. **1** shows a partial data network assembly **10** for implementation of the invention, comprising a hub **12** having hub ports **14**, and switches **16** connected to the hub ports **14**. The hub **12** may be a DEChub Multiswitch **900**, available from Digital Equipment Corporation of Maynard,

Mass. Each of the switches **16** has a plurality of switch ports **18** (e.g., eight) connecting various network resources, such as servers, computers, and printers, to the network. A bus **20** spanning each of the hub ports **14** may be enabled by an enable switch **24** to interconnect each of the switches **16**. This consequently interconnects each of the switch ports **18** across each of the interconnected switches **16**. In the preferred embodiment, the bus **20** is enabled during manufacture, thus defining the default VLAN as all of the ports of the interconnected switches **16**. The enable switch **24** may be implemented as firmware within the hub **12**, or as a manually actuated switch on the hub **12**.

New port-based VLANs may be formed across one or more of the switches **16** by selecting combinations of interconnected switch ports **18**. Selected switch ports **18** for new VLANs consequently are removed from the default VLAN definition, thus reducing the size of the default VLAN. No data packets received on any one of the default VLAN ports may be transmitted to the ports that define other VLANs.

FIG. **2** shows an exemplary eight port switch **16** forming a default VLAN, VLAN **2**, and VLAN **3**. Ports one and two define the default VLAN, ports three to five define VLAN **2**, and ports six to eight define VLAN **3**. Data packets received on switch ports one or two may be transmitted to either or both of those switch ports **18** only, thus preventing leakage to VLAN **2** and VLAN **3**. For example, a data packet received on port two having a destination address of port four will be transmitted to both ports one and two only. Similarly, a data packet received on port two having a destination address of port one will be transmitted to port one only. VLAN **2** and VLAN **3** limit leakage in like fashion.

FIG. **3** shows a data packet **26**, comprising a header **28** and an information field **30**. The destination address of the data packet **26** is stored in the header **28** of the data packet **26**. The switch port **18** associated with the destination address is ascertained by conventional means within the switch **16** receiving the data packet **26**. This information is used by the method shown in FIG. **4**.

FIG. **4** shows a flow chart that specifies the method used for preventing leakage from the default VLAN. More particularly, the destination port address is ascertained from the header **28** of a data packet received on one of the default VLAN ports (step **400**). At step **402**, it is determined if the destination port is one of the default VLAN ports. If the destination port is one of the default VLAN ports, that data packet is transmitted to the destination port only (step **404**). If the destination port is not one of the default VLAN ports, the data packet is transmitted to all of the default VLAN ports only (step **406**). The data packet is transmitted to no other switch ports **18**.

The default VLAN may be assigned a default VLAN tag that is assigned to a data packet when it enters through one of the default VLAN ports. The switch **16** then may be configured to prevent transmission of any data packet, having an associated default VLAN tag, through any of the other, non-default VLAN ports.

The invention may be implemented by means of a programmable logic chip within the one or more switches **16** used for the invention. The invention may also be implemented as firmware stored within those switches **16**. Both implementations may be programmed by conventional methods.

In an alternative embodiment, the invention may be implemented as a computer program product for use with a computer system. Such implementation may include a series of computer instructions fixed either on a tangible medium, such as a computer readable media (e.g. diskette, CD-ROM, ROM, or fixed disk) or transmittable to a computer system, via a modem or other interface device, such as communications adapter connected to the network over a medium. The medium may be either a tangible medium (e.g., optical or analog communications lines) or a medium implemented with wireless techniques (e.g., microwave, infrared or other transmission techniques). The series of computer instructions embodies all or part of the functionality previously described herein with respect to the invention. Those skilled in the art should appreciate that such computer instructions can be written in a number of programming languages for use with many computer architectures or operating systems. Furthermore, such instructions may be stored in any memory device, such as semiconductor, magnetic, optical or other memory devices, and may be transmitted using any communications technology, such as optical, infrared, microwave, or other transmission technologies. It is expected that such a computer program product may be distributed as a removable media with accompanying printed or electronic documentation (e.g., shrink wrapped software), preloaded with a computer system (e.g., on system ROM or fixed disk), or distributed from a server or electronic bulletin board over a network (e.g., the Internet or World Wide Web).

The inventive default VLAN thus prevents leakage to other VLANs by transmitting received data packets to default VLAN ports only. Security thus is ensured for data packets transmitted to the default VLAN. Furthermore, the initial size and scope of the default VLAN is increased by enabling the enable switch **24**, during manufacture, to interconnect each of the switches **16** connected to the hub **12**.

While the invention has been shown and described above with respect to various preferred embodiments, it will apparent that the foregoing and other changes of the form and detail may be made therein by one skilled in the art without departing from the spirit and scope of the invention. These and other obvious modifications are intended to be covered by the following claims.

What is claimed is:

1. A port based default VLAN formed on one or more interconnected networking switches, each switch having one or more switch ports, all of the switch ports collectively being a plurality of switch ports, the default VLAN being defined by one or more of the plurality of switch ports, the one or more of the plurality of switch ports being default VLAN ports, at least one of the plurality of switch ports defining a second VLAN, the default VLAN comprising:

means for receiving a data packet through one of the default VLAN ports;

means for ascertaining a destination port from the data packet, the destination port being one of the plurality of switch ports;

means for determining whether the destination port is one of the default VLAN ports;

first means, responsive to the determining means, for transmitting the data packet to the destination port if the determining means determines that the destination port is one of the default VLAN ports; and

second means, responsive to the determining means, for transmitting the data packet to each of the default VLAN ports if the determining means determines that the destination port is not one of the default VLAN ports,

the at least one switch port defining the second VLAN being free from transmission, from the default VLAN, of the data packet.

2. The default VLAN as defined by claim 1 wherein the data packet includes a header and the ascertaining means ascertains the destination port from the packet header.

3. The default VLAN as defined by claim 1 further including means for tagging the data packet.

4. A method of limiting broadcast messages from a port based default VLAN, the default VLAN formed on one or more interconnected networking switches, each switch having one or more switch ports, all of the switch ports collectively being a plurality of switch ports, the default VLAN being defined by one or more of the plurality of switch ports, the one or more of the plurality of switch ports being default VLAN ports, at least one of the plurality of switch ports defining a second VLAN, the method comprising the steps of:

A. receiving a data packet through one of the default VLAN ports;

B. ascertaining a destination port from the data packet, the destination port being one of the plurality of switch ports;

C. determining whether the destination port is one of the default VLAN ports;

D. transmitting the data packet to the destination port if the destination port is one of the default VLAN ports;

E. transmitting the data packet to each of the default VLAN ports if the destination port is not one of the default VLAN ports; and

F. preventing transmission, from the default VLAN, of the data packet to the at least one switch port defining the second VLAN.

5. The method as defined by claim 4 further including the step of:

G. tagging the data packet.

6. A computer program product for use with a switching device, the computer program product limiting broadcast messages from a port based default VLAN, the default VLAN formed on one or more interconnected networking switches, each switch having one or more switch ports, all of the switch ports collectively being a plurality of switch ports, the default VLAN being defined by one or more of the plurality of switch ports, the one or more of the plurality of switch ports being default VLAN ports, at least one of the plurality of switch ports defining a second VLAN, the computer program product comprising a computer usable medium having computer readable program code thereon, including:

program code for receiving a data packet through one of the default VLAN ports;

program code for ascertaining a destination port from the data packet, the destination port being one of the plurality of switch ports;

program code for determining whether the destination port is one of the default VLAN ports;

program code for transmitting the data packet to the destination port if the program code for determining determines that the destination port is one of the default VLAN ports; and

program code for transmitting the data packet to each of the default VLAN ports if the destination port is not one of the default VLAN ports,

program code for preventing transmission, from the default VLAN, of the data packet to the at least one switch port defining the second VLAN.

7. The computer program product as defined by claim 6 further including program code for tagging the data packet.

8. A port based default VLAN formed on a hub having at least two networking switches connected thereto, each switch having one or more switch ports, the port based default VLAN comprising:

a bus in the hub;

an enable switch for electrically connecting each of the VLAN ports to the bus;

means for defining a subset of the one or more switch ports of each switch as default VLAN ports;

means for receiving, at at least one of the default VLAN ports, a packet destined for a port that is not defined as one of the default VLAN ports, including means for broadcasting the received packet to each of the default VLAN ports in the subset of ports defined as default VLAN ports.

* * * * *

Exhibit 4

US006147995A

United States Patent [19]

Dobbins et al.

[11] Patent Number: 6,147,995

[45] Date of Patent: *Nov. 14, 2000

[54] **METHOD FOR ESTABLISHING RESTRICTED BROADCAST GROUPS IN A SWITCHED NETWORK**

[75] Inventors: **Kurt Dobbins**, Bedford; **Phil Andlauer**, Londonderry; **Michael Skubisz**, Durham, all of N.H.

[73] Assignee: **Cabletron Systems, Inc.**, Rochester, N.H.

[*] Notice: This patent is subject to a terminal disclaimer.

[21] Appl. No.: **09/387,317**

[22] Filed: **Aug. 31, 1999**

Related U.S. Application Data

[63] Continuation of application No. 08/960,919, Oct. 30, 1997, Pat. No. 5,946,308, which is a continuation of application No. 08/559,738, Nov. 15, 1995, Pat. No. 5,684,800.

[51] **Int. Cl.**[7] **H04L 12/28**; H04L 12/56

[52] **U.S. Cl.** **370/392**; 370/400

[58] **Field of Search** 370/392, 390, 370/396, 389, 400, 401, 402, 403, 404, 432, 469, 474, 386, 388, 395, 397, 398, 399, 405, 406, 409, 420, 422, 437

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,823,338	4/1989	Chan et al.	370/85.1
5,140,585	8/1992	Tomikawa	370/60.1
5,208,811	5/1993	Kashio et al.	370/94.1
5,394,402	2/1995	Ross	370/402
5,444,702	8/1995	Burnett et al.	370/254
5,485,455	1/1996	Dobbins et al.	370/60
5,684,800	11/1997	Dobbins et al.	370/401
5,740,171	4/1998	Mazzola et al.	370/401

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

WO 95/01023	5/1995	WIPO .
WO 95/04420	9/1995	WIPO .

OTHER PUBLICATIONS

Cheriton, David R. et al., "Host Groups: A Multicast Extension For Datagram Internetworks", *Computer Systems Laboratory Stanford University* , pp. 172–179, 1985.

Paliwoda, K. "Transactions Involving Multicast", *Computer Communications* , vol. 11, pp. 313–318, Dec. 11, 1988.

Truong, Hong Linh, "LAN Emulation on an ATM Network", *IEEE Communications Magazine* , pp. 70–85, May 1995.

Lin, Tzung–Pao, "A Platform For Seamless Interworking Among Conventional LANs and ATM Networks", *IEEE Communications* , pp. 296–301, Nov. 28, 1994.

Biagioni, Edoardo, "Designing a Practical ATM LAN", *IEEE Network* , pp. 32–39, Mar. 1993.

Catania, Vincenzo et al., "A Routing Strategy For Man Interconnection", *IEEE Communications Magazine* , pp. 608–615, 1991.

Newman, Peter, "ATM Local Area Network", *IEEE Communications Magazine* , pp. 86–98, Mar. 1994.

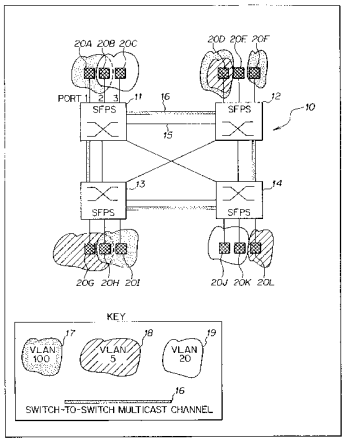
Asoh, J. et al. "Virtual LAN Realization on an ATM Connectionless Public Network", 2nd Asia Pacific Conference on Communications, vol. 2, pp. 516–520, Jun. 1995.

Primary Examiner—Dang Ton
Assistant Examiner—Phirin Sam
Attorney, Agent, or Firm—Wolf, Greenfield & Sacks, P.C.

[57] **ABSTRACT**

Method and apparatus for establishing restricted broadcast groups in a switched network. The method assigns different virtual LAN identifiers (VLAN-IDs) to different subsets of associated end systems or access ports. Tables are maintained for mapping the VLAN-IDs with associated end systems and access ports. When a broadcast packet is received at a first switch, it is encapsulated with a VLAN header, including the VLAN-IDs, and sent out a multicast channel to all other switches in the network (domain). The original packet is sent out the other access ports of the receiving switch for the designated VLAN-IDs. The switches receiving the VLAN packet remove the header and send the original packet out access ports associated with the VLAN-IDs extracted from the header. The method provides a mechanism for forwarding broadcast packets of a protocol not supported by the switching mechanism, as well as multicast packets and unicast packets from undiscovered end systems.

35 Claims, 11 Drawing Sheets



U.S. PATENT DOCUMENTS

5,742,604	4/1998	Edsall et al.	370/401
5,751,967	5/1998	Raab et al.	709/228
5,752,003	5/1998	Hart	709/223
5,818,842	10/1998	Burwell et al.	370/397
5,920,705	7/1999	Lyon et al.	709/240
5,963,556	10/1999	Varghese et al.	370/401
6,005,864	12/1999	Krause	370/395

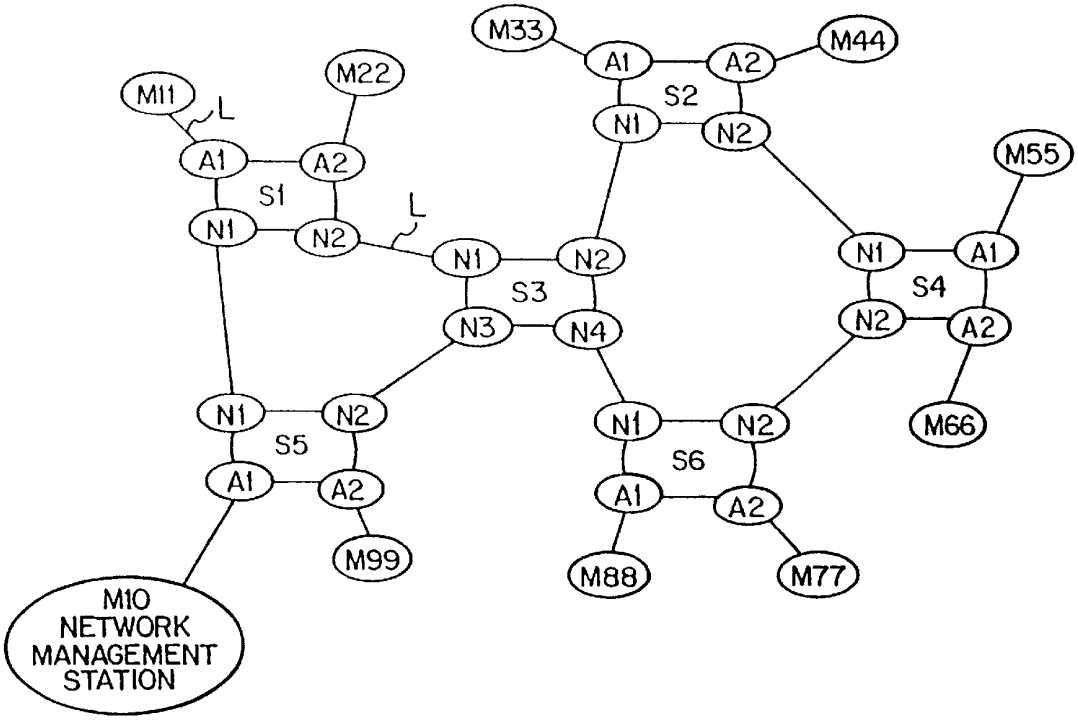

M33
A1
A2
M44
M22
M11
L
S2
N1
N2
A1
A2
M55
S1
L
N1
N2
N1
N2
N1
A1
S3
S4
N3
N4
N2
A2
N1
N2
N1
N2
M66
S5
S6
A1
A2
A1
A2
M99
M88
M77
M10
NETWORK
MANAGEMENT
STATION

FIG. 1

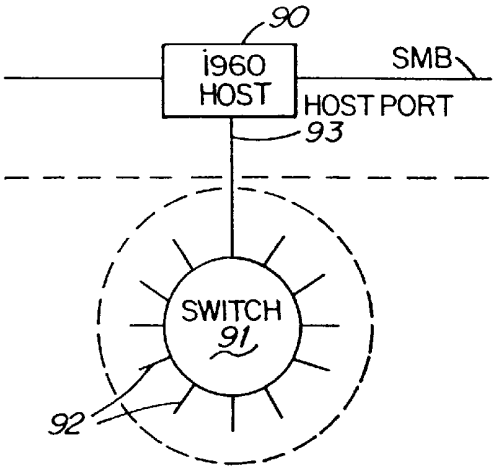

90
i960
HOST
SMB
HOST PORT
93
SWITCH
91
92

FIG. 2

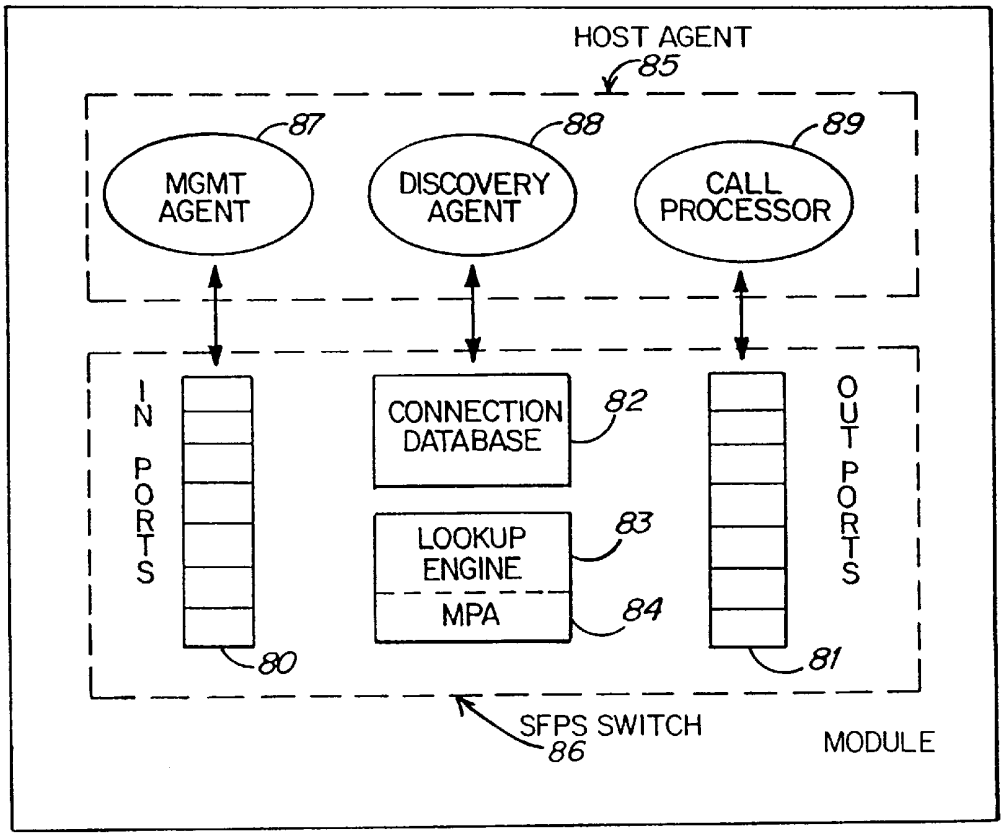

HOST AGENT
85
87
88
89
MGMT
AGENT
DISCOVERY
AGENT
CALL
PROCESSOR
IN
PORTS
CONNECTION
DATABASE
82
LOOKUP
ENGINE
83
MPA
84
OUT
PORTS
80
81
SFPS SWITCH
86
MODULE

FIG. 3

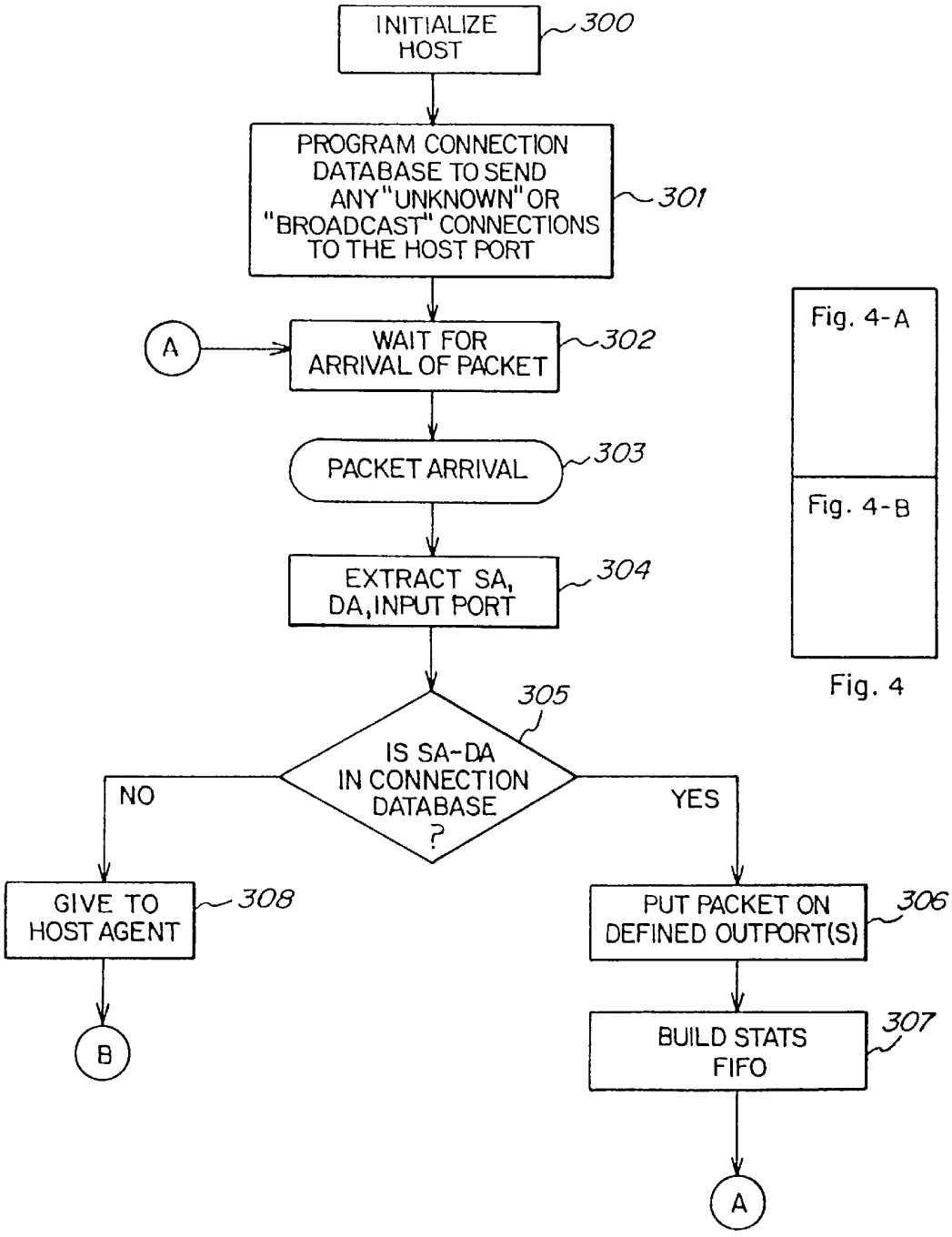

INITIALIZE HOST
300
PROGRAM CONNECTION DATABASE TO SEND ANY "UNKNOWN" OR "BROADCAST" CONNECTIONS TO THE HOST PORT
301
A
WAIT FOR ARRIVAL OF PACKET
302
PACKET ARRIVAL
303
EXTRACT SA, DA, INPUT PORT
304
305
IS SA-DA IN CONNECTION DATABASE ?
NO
YES
GIVE TO HOST AGENT
308
B
PUT PACKET ON DEFINED OUTPORT(S)
306
BUILD STATS FIFO
307
A

FIG. 4-A

Fig. 4-A
Fig. 4-B

Fig. 4

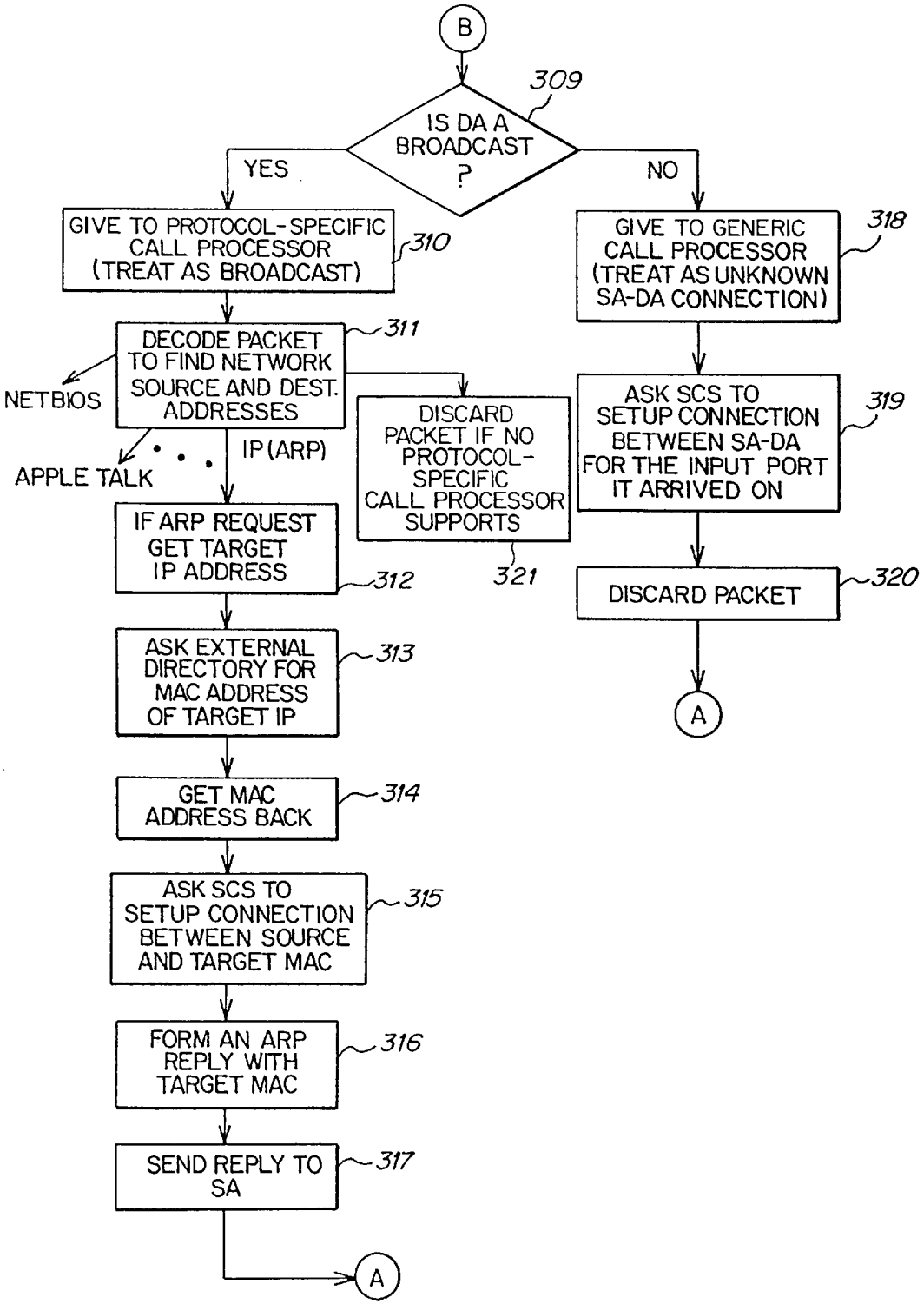

B
309
IS DA A BROADCAST ?
YES
NO
GIVE TO PROTOCOL-SPECIFIC CALL PROCESSOR (TREAT AS BROADCAST)
310
GIVE TO GENERIC CALL PROCESSOR (TREAT AS UNKNOWN SA-DA CONNECTION)
318
DECODE PACKET TO FIND NETWORK SOURCE AND DEST. ADDRESSES
311
NETBIOS
APPLE TALK
IP (ARP)
DISCARD PACKET IF NO PROTOCOL-SPECIFIC CALL PROCESSOR SUPPORTS
321
ASK SCS TO SETUP CONNECTION BETWEEN SA-DA FOR THE INPUT PORT IT ARRIVED ON
319
IF ARP REQUEST GET TARGET IP ADDRESS
312
DISCARD PACKET
320
ASK EXTERNAL DIRECTORY FOR MAC ADDRESS OF TARGET IP
313
A
GET MAC ADDRESS BACK
314
ASK SCS TO SETUP CONNECTION BETWEEN SOURCE AND TARGET MAC
315
FORM AN ARP REPLY WITH TARGET MAC
316
SEND REPLY TO SA
317
A

FIG. 4-B

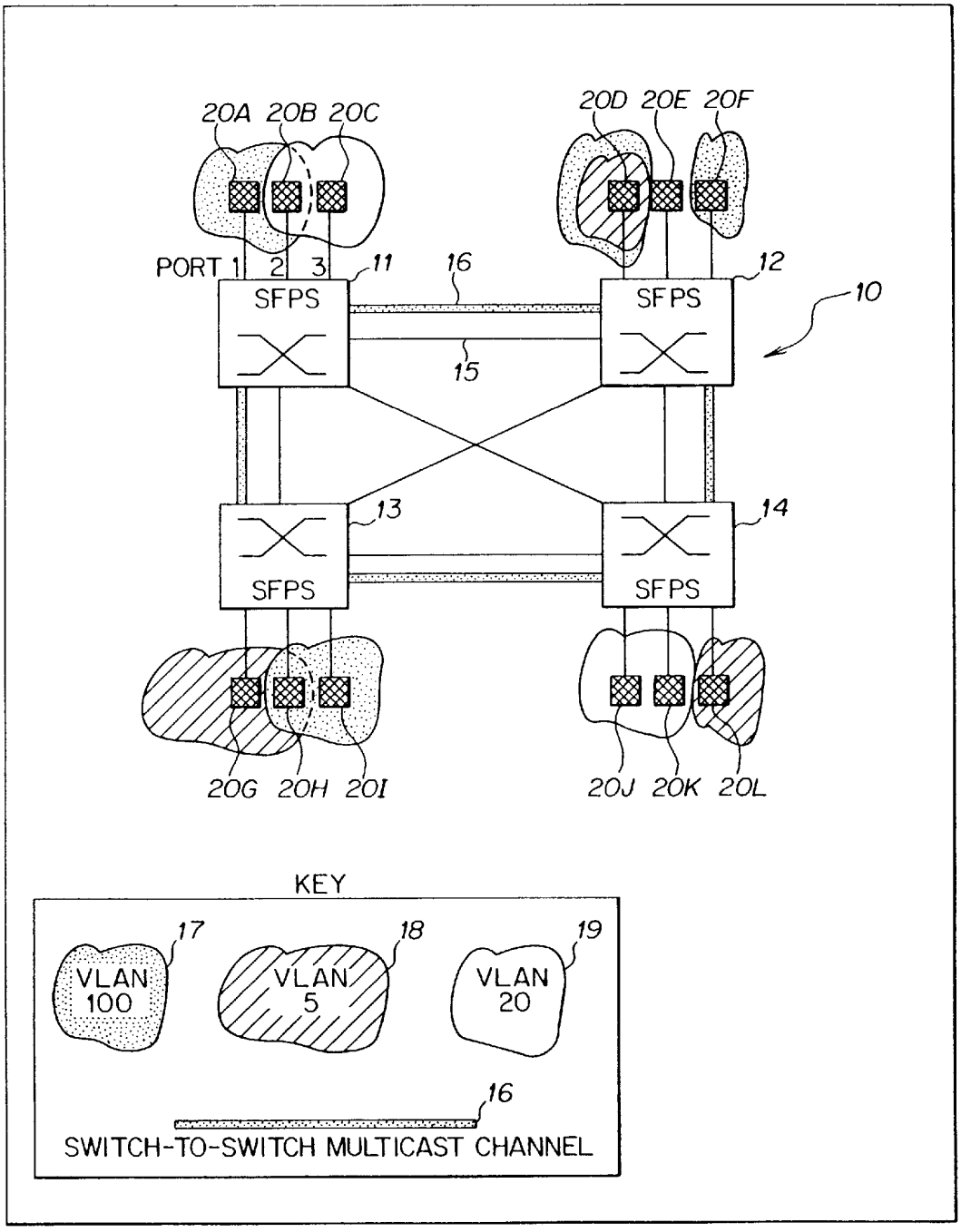

20A 20B 20C
20D 20E 20F
PORT 1 2 3
11
16
12
10
SFPS
SFPS
15
13
14
SFPS
SFPS
20G 20H 20I
20J 20K 20L
KEY
17
18
19
VLAN 100
VLAN 5
VLAN 20
16
SWITCH-TO-SWITCH MULTICAST CHANNEL

FIG. 5

TABLE 1: END SYSTEM/VLAN MAPPING FOR SWITCH 11

ACCESS PORT	END SYSTEMS HEARD	VLAN ID
1	20A	VLAN 100
2	20B	VLAN 100
		VLAN 20
3	20C	VLAN 20
•	•	•
•	•	•
•	•	•

FIG. 6

TABLE 2: VLAN/ACCESS PORT MAPPING FOR SWITCH 11

VLAN ID	ACCESS PORT
VLAN 100	1
	2
VLAN 20	2
	3
•	•
•	•
•	•

FIG. 7

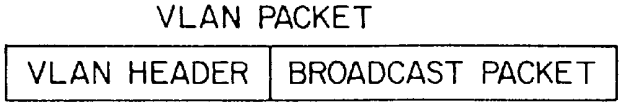


FIG. 8

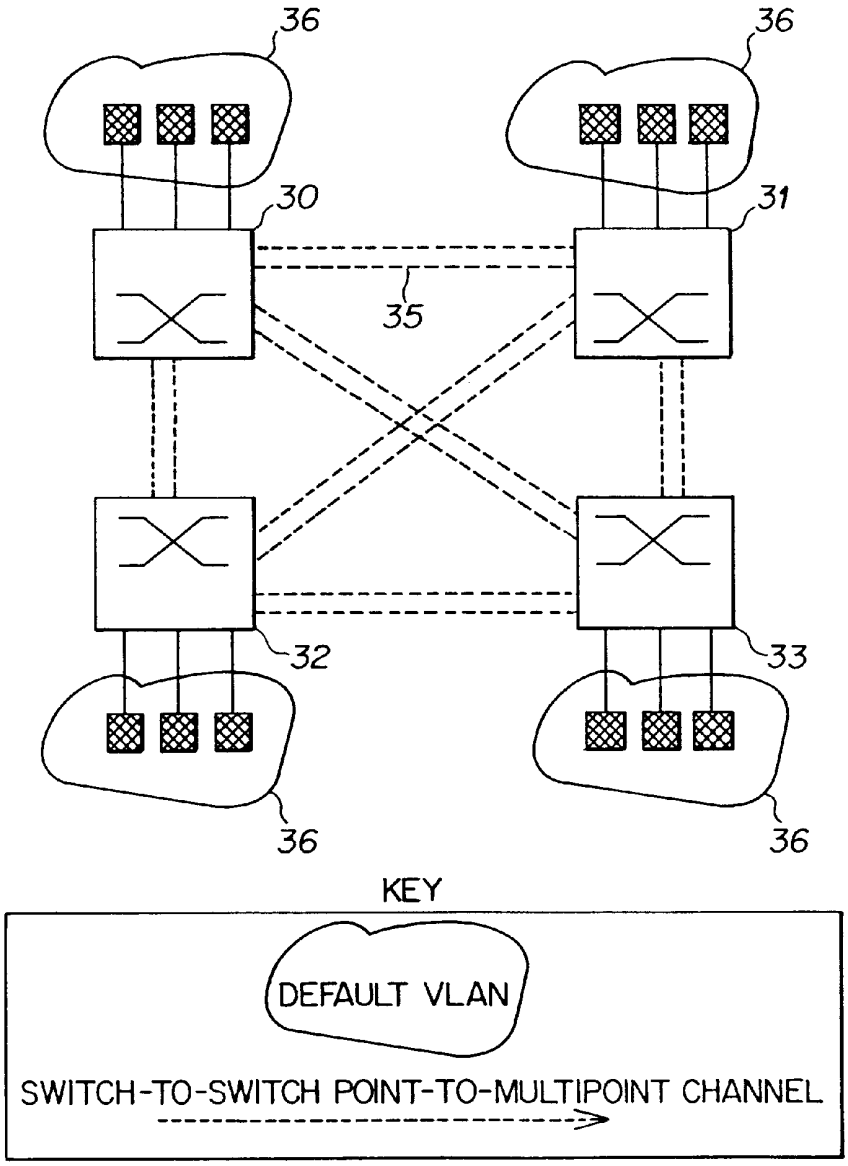

36
36
30
31
35
32
33
36
36
KEY
DEFAULT VLAN
SWITCH-TO-SWITCH POINT-TO-MULTIPOINT CHANNEL

FIG. 9

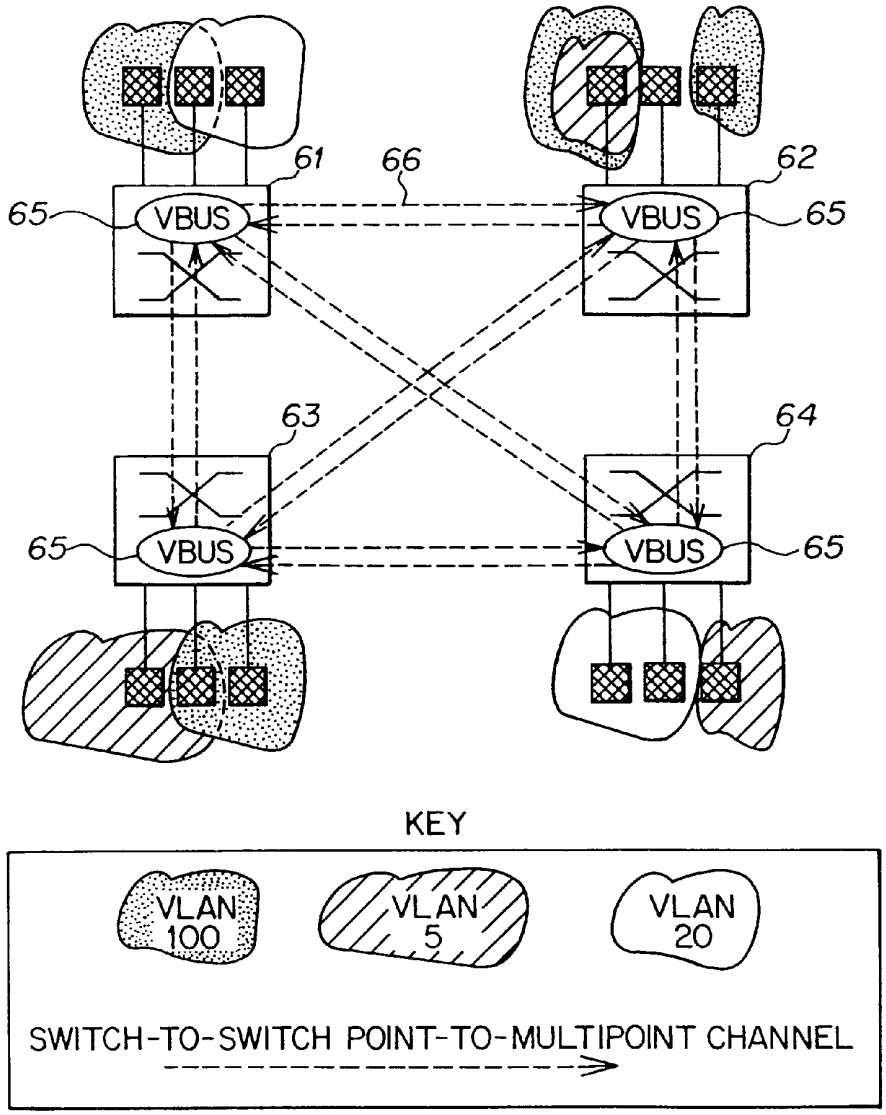

61
66
62
65
VBUS
VBUS
65
63
64
65
VBUS
VBUS
65
KEY
VLAN 100
VLAN 5
VLAN 20
SWITCH-TO-SWITCH POINT-TO-MULTIPOINT CHANNEL

FIG. 10

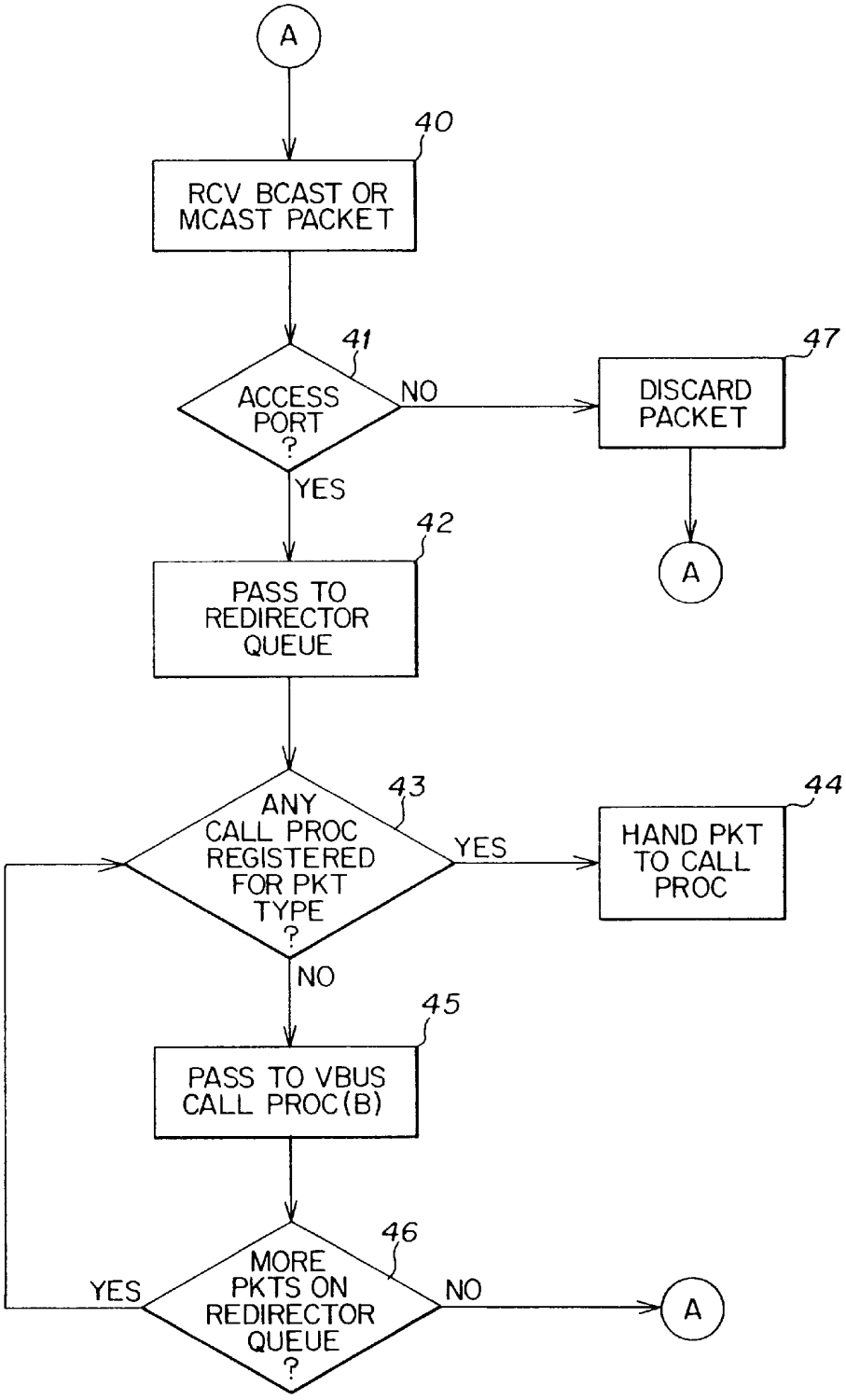

A
40
RCV BCAST OR MCAST PACKET
41
ACCESS PORT ?
NO
47
DISCARD PACKET
A
YES
42
PASS TO REDIRECTOR QUEUE
43
ANY CALL PROC REGISTERED FOR PKT TYPE ?
YES
44
HAND PKT TO CALL PROC
NO
45
PASS TO VBUS CALL PROC(B)
46
MORE PKTS ON REDIRECTOR QUEUE ?
YES
NO
A

FIG. 11

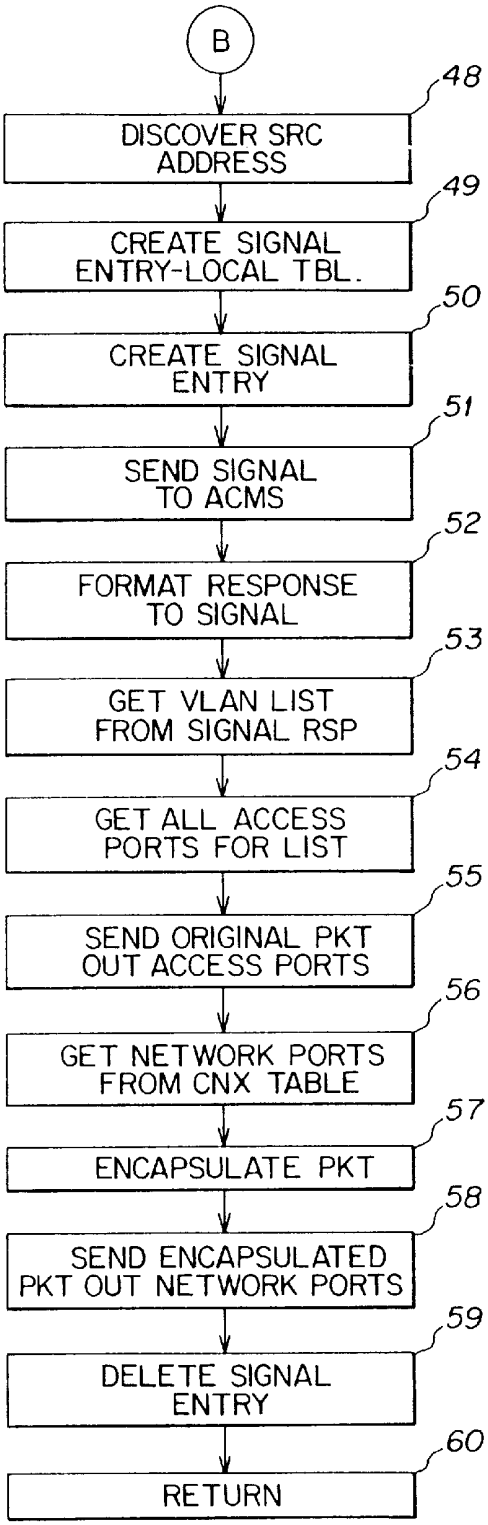

B
48
DISCOVER SRC ADDRESS
49
CREATE SIGNAL ENTRY-LOCAL TBL.
50
CREATE SIGNAL ENTRY
51
SEND SIGNAL TO ACMS
52
FORMAT RESPONSE TO SIGNAL
53
GET VLAN LIST FROM SIGNAL RSP
54
GET ALL ACCESS PORTS FOR LIST
55
SEND ORIGINAL PKT OUT ACCESS PORTS
56
GET NETWORK PORTS FROM CNX TABLE
57
ENCAPSULATE PKT
58
SEND ENCAPSULATED PKT OUT NETWORK PORTS
59
DELETE SIGNAL ENTRY
60
RETURN

FIG. 12

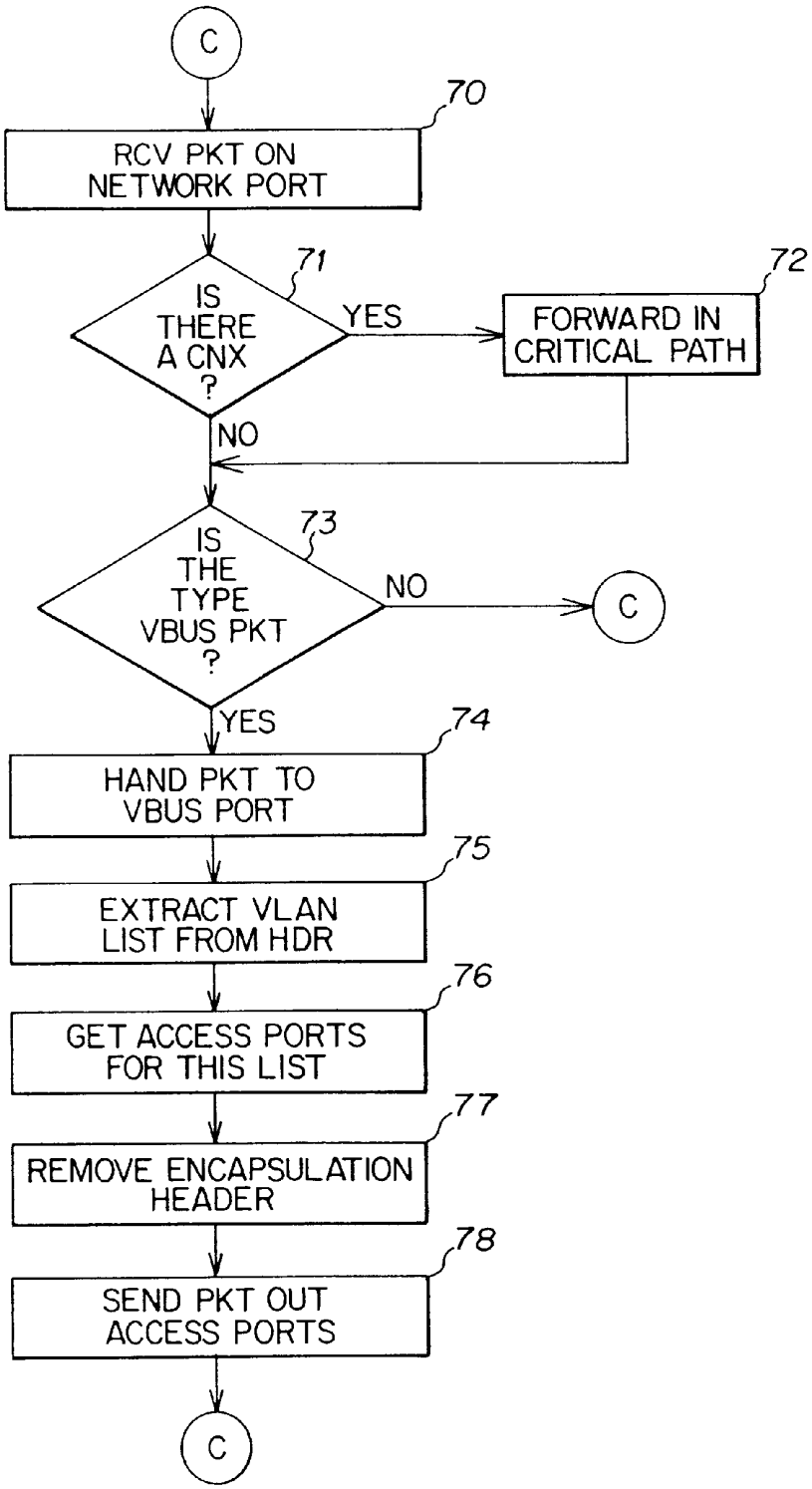

C
70
RCV PKT ON NETWORK PORT
71
IS THERE A CNX ?
YES
72
FORWARD IN CRITICAL PATH
NO
73
IS THE TYPE VBUS PKT ?
NO
C
YES
74
HAND PKT TO VBUS PORT
75
EXTRACT VLAN LIST FROM HDR
76
GET ACCESS PORTS FOR THIS LIST
77
REMOVE ENCAPSULATION HEADER
78
SEND PKT OUT ACCESS PORTS
C

FIG. 13

METHOD FOR ESTABLISHING RESTRICTED BROADCAST GROUPS IN A SWITCHED NETWORK

This application is a continuation of application Ser. No. 08/960,919 filed on Oct. 30, 1997 which issued as U.S. Pat. No. 5,946,308 on Aug. 31, 1999: which is a continuation of application Ser. No. 08/559,738 filed Nov. 15, 1995 and which issued as U.S. Pat. No. 5,684,800 on Nov. 4, 1997.

FIELD OF THE INVENTION

This invention relates to packet switched data communications networks, and more particularly to an apparatus and method for establishing restricted broadcast groups known as virtual LANs (VLANS) which provide a simple but robust mechanism that allows broadcast and/or multicast packets to be "flooded" through a switched domain and transmitted only to those users or ports defined for a particular VLAN.

RELATED APPLICATIONS

The subject matter of the above application may be advantageously combined with the subject matters of the following copending and commonly owned applications, which are hereby incorporated by reference in their entirety:

U.S. Ser. No. 08/188,238 entitled "Network Having Secure Fast Packet Switching And Guaranteed Quality Of Service," filed Jan. 28, 1994 by Kurt Dobbins et al.; and

U.S. Ser. No. 08/187,856 entitled "Distributed Chassis Agent For Network Management," filed Jan. 28, 1994 by Brendan Fee et al.

BACKGROUND OF THE INVENTION

Most data communications networks today rely heavily on shared medium, packet-based LAN technologies for both access and backbone connections. These networks use bridges and routers to connect multiple LANs into global internets. An internet router must be capable of processing packets based on many different protocols, such as IP, IPX, DECNET, AppleTALK, OSI, SNA and others. The complexities of building networks capable of routing packets on the global internet using different protocols is a challenge for both vendors and users.

In U.S. Ser. No. 08/188,238 to Dobbins (see related applications above), there is described a new secure fast packet switching (SFPS) technology which provides the same or better reliability and security as routers, but with much greater performance and without an increase in cost. The SFPS system avoids the complexities and costs of providing multi-protocol routers. Also, the SFPS system provides capabilities which routers do not, such as the ability to create separate logical work group LANs on the same physical network and the ability to guarantee a quality of service (QOS) by providing dedicated switched paths through the network.

SFPS provides high performance packet switching based on physical layer addresses such as source and destination MAC IDs—the unique medium access control (MAC) address assigned to each end system by the IEEE. End-to-end connections are determined by a network management application that provides security and best path routing determinations based on a number of constraints. By switching packets based only on MAC layer information, network infrastructure remains protocol insensitive.

More specifically, SFPS uses source and destination MAC addresses which alone, or in combination with an input port on a switch, form a unique "connection identifier" for any communication exchange between designated end systems. As an example:

input port=2

source MAC address=00:00:1D:01:02:03

destination MAC address=00:00:1D:11:22:33

together form a "tuple" bound to a specific unidirectional flow from a source address to a destination address. All packets that have this tuple are automatically switched according to the operation of the SFPS.

Network infrastructures are built around a core switching fabric, which provides the physical paths or routes that allow users to send information to one another. Access to the switching fabric is gained through an access port. Access ports provide several functions—most importantly, they provide security and accounting services. End systems such as personal computers (PCs), workstations and servers connect to an access port using one or more access technologies such as Ethernet, Token Ring, FDDI, or ATM.

In traditional bridge and router devices, each packet is treated as an independent unit of data which is individually processed by application of access and security constraints, as well as path determination. In contrast, with SFPS this processing is done only on initial probe packets which are decoded, and through use of a central directory of end system constraints policy, call attributes, location, paths, quality of service, etc., the connection is either rejected or accepted. If accepted, the path is determined and switches along the path are "programmed" to allow subsequent packets on this "connection" to be switched. In either case, subsequent datagrams are either switched or discarded without having to reapply all of the security and access control and path determination logic.

The SFPS switching technology may be constructed as: software objects which exist in embedded devices as firmware; software objects which are part of an application on a commercial computer system; application specific integrated circuits (ASIC); or functionally equivalent hardware components.

It is common for internetworking devices to "route" the protocols that a device supports, and "bridge" the protocols that are not supported for routing. In addition, some protocol frames (such as DECs LAT) are actually unroutable. In SFPS switches, there are protocol-specific call processors to route protocol-specific broadcast frames (note that unicast frames can be processed by a "generic" call processor that does not decode or translate the frame at all, but instead makes the connection request based on the source and destination unicast MAC addresses in the frame). However, a problem arises in that an SFPS switch has nothing equivalent to bridging of multicast and broadcast packets for non-supported protocols. Thus, until a protocol-specific call processor is implemented in a switch, the switch must discard any broadcast or multicast frames it does not understand.

SUMMARY OF THE INVENTION

A method and apparatus are provided for establishing restricted broadcast groups within a switching fabric, known as virtual LANs (VLANs). The VLANs provide a simple but robust mechanism for allowing broadcast and multicast packets to be "flooded" through the switching fabric and transmitted only to those users or ports defined for a particular VLAN.

More specifically, the switched network includes a plurality of end systems and switches connected by links. The switches have access ports connected to end systems and network ports connected to other switches. Each end system has a unique physical layer address, e.g., MAC address. In accordance with this invention, different virtual LAN identifiers (known as VLAN-IDs) are assigned to different subsets of associated end systems or access ports. Each access switch maintains a first table for mapping VLAN-IDs to associated end systems and/or access ports (the End System/VLAN table). Each access switch also maintains a second table for mapping access ports (of associated end systems) to associated VLAN-IDs (the VLAN/Access Port table).

According to a first embodiment, the restricted V-LAN flooding is used for broadcast packets of a protocol not supported by the switches. When an original broadcast packet (of an unsupported protocol) is received by a first access switch from a first end system, the first switch determines and adds a VLAN header to the original data packet to create a VLAN packet. The VLAN header includes designated VLAN-IDs from the first table. The designated VLAN-IDs are assigned based on the physical source address of the first end system. The first access switch then forwards the VLAN packet to all other switches on a multicast channel of point-to-point connections between the switches. The first switch also forwards the original broadcast packet out the access ports identified in the second table for the designated VLANS (except the originating port).

The other switches receive the VLAN packet and extract the designated VLAN-IDs from the VLAN header and then forward the original packet out the access ports, defined in the other switch's second table, for the designated VLAN-IDs.

Other embodiments include the designation of a default VLAN-ID which maps to all access ports, a reserved VLAN-ID for use with multicast packets, and restricted flooding for packets directed to an undiscovered end system. Still another embodiment provides a single or distributed network server on the multicast channel (between switches) to handle broadcast and multicast packets, which embodiment scales better for larger networks.

More specific methods and a particular apparatus for implementing the present invention are described in the following detailed description and drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a schematic illustration of a network topology built with SFPS switches.

FIG. **2** is a schematic illustration of an SFPS switch.

FIG. **3** is a logical view of an SFPS switch.

FIG. **4**, combining FIGS. **4**-A and **4**-B, is a flowchart showing processing of a data packet by an SFPS switch.

FIG. **5** is a schematic illustration of a network topology including three virtual networks (VLAN **100**, VLAN **5**, and VLAN **20**) according to the present invention.

FIG. **6** shows an end system table for mapping VLAN-IDs to associated end systems.

FIG. **7** shows a port table for mapping access ports (of associated end systems) to associated VLAN-IDs.

FIG. **8** shows one embodiment of a VLAN packet, in which a VLAN header is appended to an original data packet.

FIG. **9** is a schematic illustration of a network topology utilizing a default VLAN according to the present invention.

FIG. **10** is a schematic illustration of a network topology utilizing a distributed VBUS server according to an alternative embodiment of the present invention.

FIG. **11** is a flow chart showing the redirected flow of a broadcast or multicast packet to the VBUS service.

FIG. **12** is a flow chart showing the call processing performed by the VBUS service.

FIG. **13** is a flow chart showing the channel listening process of the VBUS service.

DETAIL DESCRIPTION

The SFPS Network—Switching of Unicast Packet with Generic Call Processor and Switching of Protocol-supported Broadcast Packets

According to one embodiment, the establishment of VLANs for transmitting broadcast/multicast packets of non-supported protocols is intended for use in the SFPS switched network described in U.S. Ser. No. 08/188,238. The following is a general description of the operation of switching "unicast" packets on that network, as illustrated in FIGS. **1–4**.

FIG. **1** shows a representative network topology built with six secure fast packet switches (SFPS) labeled **S1–S6** connected by links L. Each switch has for example four ports; some ports are labeled A for access and some are labeled N for network. The end systems are connected to the access ports by links L and are labeled "M__". One end system is a network management station or server (NMS) **M10**, which includes a connection server.

FIG. **2** is a schematic illustration of an SFPS switch **91** having a plurality of ports **92**. A host port **93** connects the switch to its host CPU **90**, which may be an i960 microprocessor sold by Intel Corporation. The host CPU is connected to a system management bus (SMB) for receipt and transmission of discovery and other control messages.

FIG. **3** illustrates the internal operation of the switch. The SFPS switch **86** includes in ports **80**, out ports **81**, a connection database **82**, a look-up engine **83**, and a multi-level programmable arbiter MPA **84**. The switch **86** sends and receives messages from the host agent **85**, which includes a management agent **87**, a discovery agent **88**, and a call processing agent **89**.

The management agent **87** provides external control of the configuration and operation of the SFPS switch, through the network management system.

The discovery agent **88** provides a mapping of end systems to switching ports through a passive listening (snooping) capability and a registering of end system addresses and port locations of the host switch with a common external directory. Adjacent switches are also discovered and mapped, but this may be done with an explicit switch-to-switch protocol (nonpassive).

The call processor **89** provides a means for requesting connections to be established between two end systems. Unicast frames are handled by a "generic" call processor which programs the switches based on the source and destination MAC addresses. In a case where the source and destination MAC addresses are not in the packet frame, i.e., usually in a frame that has a broadcast—all hosts—MAC address, a "protocol-specific" call processor (if available) will decode the packet to find source or destination network addresses and use these to map back to the physical addresses via the external directory. Once the end system MAC addresses are known, the protocol-specific call processor will then request the connection between the end systems. If the broadcast frame was a probe or address resolution packet (i.e., an implied connection request), the

call processor will return a probe reply as a "proxy" which gives the destination end system MAC addresses. Subsequently, the source end system can then send packets directly to the destination based on its MAC address.

FIG. **4** is a flow chart illustrating what happens from the time a data packet is received on an in port of the switch, until it is sent on the correct out port.

Referring to FIG. **4**-A, in step **300** the host CPU **90** is initialized. The host programs the connection database **82** to send any "unknown" or "broadcast" connections to the host port (step **301**). The switch then waits for a packet to arrive (step **302**). Once a packet has arrived (step **303**), the switch extracts the source MAC address, destination MAC address, and identifies the inbound port on which the packet was received (step **304**). The look-up engine **83** checks to see whether this source-destination pair is already located in the connection database **82** (step **305**). If it is not found, the packet is given to the host agent **85** (step **308**). The call processor and the host agent determine whether it is a broadcast destination (step **309**). If the answer is yes, a protocol-specific call processor (if available) decodes the packet to find the network protocol source and destination addresses (steps **310**–**311**). A different protocol decode logic would be provided for each network protocol. For example, in the IP protocol, if an ARP request is received, the call processor would get the target IP address (step **312**). It would then ask the external directory for the MAC address of the destination IP (step **313**). In the next step **314**, the directory sends the MAC destination address back to the call processor. The call processor **89** then asks the connection server (SCS) to set up a connection between the source MAC and destination MAC (step **315**). The call processor **89** forms an ARP reply packet by putting the destination MAC address inside the packet (step **316**), and the call processor sends a reply to the source address (step **317**). It should be noted that this reply allows the source end system to update its private mapping of the destination IP address to a nonbroadcast MAC address. All subsequent packets to this destination IP address will be properly framed with the source and destination MAC address for which connections will now exist.

Note that if no call processor exists which supports the relevant protocol, the broadcast packet is dropped (step **321**). The present invention is a method of handling such packets.

If the packet is not a broadcast packet, it is given to the "generic" call processor (treated as an unknown SA-DA connection—step **318**), which asks the connection server to set up a connection and forward the packet (step **319**); the call processor then discards the packet (step **320**).

Returning to step **305**, if the source and destination MAC pair are found in the connection database **82**, the data packet is sent to the switch outport(s) **81** defined in the database (step **306**). In next step **307**, the management agent **87** collects statistics regarding transmissions through the switch and sends them to the SCS (connection server).

Restricted Broadcast Groups for Non-supported Broadcast, MultiCast and Unknown Unicast Packets

FIG. **5** illustrates generally a logical view of the present invention for establishing restricted broadcast groups or virtual LANs (VLANs) within a switched network. The representative network **10** has four SFPS switches **11**–**14**, all of the switches being connected by physical links forming point-to-point connections **15**, and which physical connections together form a logical multicast channel **16**. The multicast channel **16** connects the network ports of all switches. A plurality of end systems **20**A–**20**L extend from access ports on the various switches **11**–**14**. The end systems are shown grouped into different subsets known as virtual LANs **17**, **18** and **19**, which are given VLAN identifiers VLAN **100**, VLAN **5**, and VLAN **20**, respectively. As shown **30** in FIG. **5**, "VLAN **20**" includes end systems **20**B, **20**C, **20**J and **20**K. "VLAN **5**" includes end systems **20**D, **20**G, **20**H and **20**L. "VLAN **100**" includes end systems **20**A, **20**B, **20**D, **20**F, **20**H and **20**I.

During a discovery time, as each switch **11**–**14** is discovered, it is put in a point-to-point connection that connects all SFPS switches. This forms the multicast channel **16** which all switches use between themselves.

Also during the discovery time, each switch **11**–**14** discovers its associated end systems (i.e., switch **11** discovers end systems **20**A, **20**B, **20**C) and enters these end systems in a common directory which assigns VLAN-IDs to the end systems. The directory returns a mapping of VLAN-IDs and associated end systems, which mapping each switch uses to build two internal tables: a first table that lists the VLAN-ID for each end system (the End System/VLAN Table—see FIG. **6**), and a second table that defines a port mask for each VLAN-ID (the VLAN/Access Port Table—see FIG. **7**).

During real time operation of the system, a first switch (for example switch **11**) receives a broadcast or multicast packet that it cannot process with a protocol-specific call processor. The switch will encapsulate the original packet and insert a VLAN header containing a list of VLAN-IDs for the source end system (see FIG. **8**), before flooding the encapsulated (VLAN) packet out the multicast channel **16** to all other switches. For example, if first switch **11** receives a broadcast packet from first end system **20**B, switch **11** returns from its end system table (FIG. **6**) that VLAN **100** and VLAN **20** are associated with source end system **20**B. First switch **11** will insert VLAN **100** and VLAN **20** into the VLAN header (FIG. **8**). In addition, first switch **11** determines the port masks for VLAN **100** and VLAN **20** from its port table (FIG. **7**), and then sends the original broadcast packet out all access ports of the first switch in VLAN **100** or VLAN **20** (except for the source port **2**); in this case, the original packet is sent out access port **1**, which connects to end system **20**A, also in VLAN **100**, and out access port **3**, which connects to end system **20**C, also in VLAN **20**.

As each of switches **12**, **13** and **14** receive the VLAN packet on multicast channel **16**, they strip off the encapsulated VLAN header and look up in their respective VLAN/Access Port table for any associate mapping to VLAN **100** and VLAN **20**. Switch **12** determines in its port table that it has associated access ports to end systems **20**D and **20**F designated for VLAN **100**. Similarly, switch **13** determines from its port table that it has associated access ports to end systems **20**H and **20**I for VLAN **100**. Switch **14** determines from its port table that it has associated access ports to end systems **20**J and **20**K for VLAN **20**. The original packet is thus transmitted out the access ports to end systems **20**D, **20**F, **20**H, **20**I, **20**J and **20**K.

The following describes the changes and additional functionality required of the SFPS access switches to support the establishment of VLANs for multicast and broadcast packets. Switches with only network ports continue to function as described in prior U.S. Ser. No. 08/188,238 to Dobbins et al.

The Switch-to-switch Multicast Channel

Each SFPS switch supports the multicast channel **16** by having a connection in each switch that connects it to all other switches in the network (or within a subsection of the network, such as a domain). This is in essence a point-to-multipoint connection in each switch. It should be noted that

this multipoint connection is only between the switches themselves, which scales better than having a multipoint connection between all users (end systems).

A connection server (i.e., **M10** in FIG. **1**), which includes a common directory of all switches, has the responsibility to program the multicast channel connection each time a new switch joins or leaves the topology, i.e., such a change may be detected by neighbor advertisement signals sent by the switches.

The End System/VLAN Table (FIG. **6**)

Each switch that has an access port maintains a table of end systems heard on each access port, and a list of VLANs to which each end system belongs. An end system can belong to more than one VLAN at any given time.

The assignment of VLAN-IDs may be accomplished in several ways. First, the VLAN-IDs may be maintained by a common directory. For example, as each end system is discovered by an access switch, it is registered with a common directory of end systems for the entire network, and the directory then returns a list of VLAN-IDs to the access switch with the "End System Discovery Message ACK." Alternatively, a management application may administratively assign the VLAN-IDs, and manage the end system and port tables in the switch. As a further alternative, an access switch may send a Resolve signal to a directory, which directory then returns a mapping of VLAN-IDs for an associated end system.

The VLAN/Access Port Table (FIG. **7**)

Each switch having an access port maintains a port table which maps VLANs to associated access ports. This table may be filled in dynamically through the implicit mapping of VLANs to end systems. Each time a VLAN is mapped to an end system, it is automatically inserted in a port-mapping entry for the source port on which the end system is located. Ports, like end systems, can belong to more than one VLAN at any given time; the port's VLAN mapping strictly depends upon the VLAN of the end systems on it. The out ports for each VLAN entry in the table essentially define the flood mask for the access ports.

The Default VLAN-ID

A default VLAN-ID (VLAN-ID=1) may be used to map an end system to a VLAN when no VLAN-ID had been administratively assigned. The default VLAN is a special case which maps to all access ports. This allows flooding out all ports when no VLAN is defined. For example, FIG. **9** illustrates a network of four switches **30**, **31**, **32**, **33**, connected by multicast channel **35**, prior to assignment of any specific VLAN-IDs such that all end systems fall within a default VLAN **36**.

The VLAN Call Processor

The VLAN call processor is essentially a default call processor for broadcast/multicast packets for which no protocol-specific call processor exists. For example, an ARP call processor would be able to decode an ARP broadcast message.

The VLAN call processor would take any packet it receives and then encapsulate the broadcast/multicast packet in a header, the header containing a list of VLAN-IDs on which the packet belongs. The VLAN-ID list may be determined by using the source MAC address of the original packet and doing a look-up in the end system table. In this example, VLAN-IDs are determined based on the source rather than the destination. Once this encapsulated packet is formed, it is then forwarded on the multicast channel to all other switches. The original packet would also be given to the local forwarder.

The Local Forwarder

Each switch has a process that listens on the multicast channel **16**. This process is responsible for processing any encapsulated frames (VLAN packets) sent from other switches. When the VLAN packet is received, it is stripped of its VLAN-ID list in the header. For every entry in the VLAN-ID list, the local port table is searched for a matching entry. The local forwarder then forwards the original data packet out any ports that are mapped to the VLAN-ID. If the VLAN-ID is the default VLAN-ID (=1), then the original packet is flooded out all access ports on the switch. If no VLAN-IDs match, then the packet is discarded.

The Central Connection Server and Common Directory

A central connection server programs the point-to-multipoint connections between all of the SFPS switches, as there is no provision in each switch to do so (see **M10** in FIG. **1**). Thus, any time the connection server "discovers" a change in a switched topology, it has to reprogram the multicast channel between the switches.

The server accesses a common directory for mapping end systems to VLAN-IDs. A management application may provide this on the front end, and in addition provide for changes to the mapping in the directory itself and in any switches that have been informed of the mapping. Any end system not defined with a VLAN would default to VLAN-**1**.

If the VLAN assignment is done inside an End System Discovery Message ACK, then a new TLV list is added to the message. This functions similar to an "alias" field in which more than one are allowed since multiple VLANs could be returned. If the VLAN assignment is done with Resolve messages, then only a new TAG type has to be assigned since the message supports returning multiple resolutions. The semantics would be "resolve this end system to its VLAN-IDs." If the assignment was done completely out of band, then no signalling changes would be required.

Reserved VLAN-IDs

In the previous embodiment, broadcast and multicast packets are propagated through the switches based on the VLAN-IDs to which the source belongs. In some cases, mostly with multicast frames, it may be desirable to map a VLAN to the destination, e.g., OSPF packets.

This may be accomplished by allowing the switches to support "well-known" VLANs without any run-time assignment. If a switch receives a packet destined for a reserved VLAN, it would encapsulate it and set the VLAN list without mapping it to the end system table. The packet would then be forwarded out the multicast channel and any switches supporting the reserved VLAN (or having heard a reserved VLAN-type packet), would flood the original packet out.

Unicast Flooding

VLANs may be supported for unicast frames, for example if a call processor has not yet discovered the end system. This works similar to the broadcast/multicast operation except that instead of mapping the outports at each flooding switch, each switch would look up the destination unicast address in the end system table and send the original packet out the port on which the end system belongs.

VBUS Service

Since point-to-point connections between switches does not scale well, in an alternative embodiment each switch has a connection to a single (or distributed) server in the network which will forward broadcast and multicast packets. This service, referred to as the Virtual Broadcast/Unknown Service (VBUS), is distributed into all SFPS switches in a first implementation as illustrated in FIG. **10**. Switches **61**, **62**,

63, 64 are connected by multicast channel 66, and each switch includes the distributed VBUS service 65.

FIG. 11 illustrates the redirected flow of data packets for the VBUS service. When a first switch receives a broadcast or multicast packet (step 40), it first determines whether the packet was received on an access port (step 41). If no, the packet is discarded (step 47). If yes, the packet is passed to a redirector queue (step 42), and if a call processor supports the packet type (step 43), the redirector delivers the packet to the protocol-specific call processor (step 44). If not, the packet is passed to the VBUS call processor (step 45). The redirector queue then handles the next packet on the queue (step 46).

FIG. 12 illustrates the operation of the VBUS call processor. Each switch listens for source addresses heard on each access port (step 48). The call processor then updates the End System/VLAN table with the access port and end systems heard (step 49). The call processor then creates a signal entry (step 50) which is sent to the connection server (step 51), which formats a response to the signal (step 52). The connection server returns a signal with the associated VLAN list, which is received by the call processor (step 53). The call processor gets the associated access ports from the VLAN/Access Port Table (step 54) and sends out the original packet on the associated access ports (step 55). The call processor gets the network ports from the switch's connection table (step 56), encapsulates the packet with the VLAN header (step 57), and sends the encapsulated packet out the network ports to the other switches (step 58). The call processor then deletes the signal entry (step 59) and returns to start (step 60).

FIG. 13 illustrates the operation of the VBUS channel listener. When a packet is received on a network port (step 70), it first determines whether there is a known connection in the connection database 82 (step 71), and if so, it forwards the packet out the appropriate outport (step 72). If there is no connection, it determines whether this is a VBUS packet (step 73). If no, it returns to the beginning. If it is a VBUS packet, the packet is handed to a VBUS port (step 74) and the VLAN list is extracted from the header (step 75). The access ports are obtained from the VLAN/Access Port Table (step 76), and the encapsulation header removed from the packet (step 77). The original packet is then sent out the associated access ports defined in the table (step 78).

In one embodiment, the switch provides a MIB interface to allow an external application to assign VLAN-IDs to access ports and/or end systems. The simplest model is to progam VLAN-IDs to the switched ports only; under this model, the administration is simpler and the VLAN assignment to end systems is implied by the switched port to which the end systems are physically connected. A more robust model would map the VLAN-IDs from policy work group definitions.

The application interface may be provided with an SNMP (Simple Network Management Protocol) MIB (management information base) which allows a simple interface to program connections via a single SNMP set message. The MIB interface provides the following semantics:

(map, unmap)[SFPS VLAN-ID][inPort][userMAC] This verb set assigns (and removes) a user MAC address and switch port to (or from) a specific VLAN.

(map-port, unmap-port)[SFPS VLAN-ID][inPort] This verb set assigns (and removes) a switch port to (or from) a specific VLAN. The switches provide managed objects accessible by the MIB which are all accessed with standard SNMP get, get next, and set messages.

In one embodiment, a VLAN status table is provided. This table allows an entire VLAN to be enable or disabled regardless of the number of user or switch ports assigned to the VLAN in the switch. Thus, it is possible to shut off a particular VLAN inside a particular switch without having to administer each individual switch port or end system.

One goal of the VBUS service is to require minimal support from the network server. The only server requirement is providing each switch with a connection to all other switches in the network (domain), which in effect provides the multicast channel for flooding VLAN packets.

While there have been shown and described several embodiments of the present invention, it will be obvious to those skilled in the art that various changes and modifications may be made therein without departing from the scope of the invention as defined by the appending claims.

What is claimed is:

1. A computer-readable storage medium comprising program instructions for restricting flooding of a data packet, of one of a broadcast, multicast and unknown destination type, in a switched data communications network, the network including a plurality of end systems and switches connected by links, the switches having access ports connected to end systems and network ports connected to other switches, the program instructions causing the network to:

a. assign at least one identifier to a respective subset of end systems;

b. map the at least one assigned identifier to an access port attached to at least one end system in the respective subset of end systems; and

c. when the data packet is received from a source end system at a receiving access port of a first switch:

i) determine one or more identifiers associated with the source end system;

ii) encapsulate the data packet by adding a header with the one or more determined identifiers;

iii) forward the encapsulated data packet to all or a subset of other switches in the network; and

iv) determine if at least one access port other than the receiving access port on the first switch is associated with the one or more determined identifiers and forward the data packet out the at least one determined access port.

2. The computer-readable storage medium as recited in claim 1, further comprising instructions to cause the network to:

d. when the encapsulated data packet is received at a second switch with access ports:

i) strip the header from the encapsulated data packet and to determine the one or more encapsulated identifiers in the header of the encapsulated data packet;

ii) determine if at least one access port of the second switch is associated with the one or more encapsulated identifiers; and

iii) forward the data packet out the at least one determined access port of the second switch.

3. The computer-readable storage medium as recited in claim 2, further comprising instructions to cause the network to:

listen to end systems heard on respective access ports at each switch and to maintain the end systems heard and their respective access ports in a mapping table at the respective switch; and

upon receipt of a unicast packet for a destination end system unknown to the first switch, complete step c for the unicast packet and then at the next switch review the mapping table for the respective access port for the

destination end system and forward the packet out the respective access port.

4. The computer-readable storage medium as recited in claim **1**, further comprising instructions to cause the network to, if in step c(iv) no other access port is determined, discard the data packet.

5. The computer-readable storage medium as recited in claim **1**, further comprising, in step b, instructions to cause the network to:

maintain a first table in each switch to relate the at least one assigned identifier to the end systems or access ports of the respective switch; and

maintain a second table in each switch to relate the access ports of the respective switch to assigned identifiers.

6. The computer-readable storage medium as recited in claim **5**, further comprising, in step c.i), instructions to cause the network to:

review the first table for the one or more identifiers associated with the source end system or the receiving access port.

7. The computer-readable storage medium as recited in claim **6**, further comprising, in step c.iv), instructions to cause the network to:

review the second table for an access port associated with the one or more determined identifiers.

8. The computer-readable storage medium as recited in claim **7**, wherein the assigned identifier is a virtual LAN identifier.

9. The computer-readable storage medium as recited in claim **1**,wherein the received data packet is of a protocol not supported by a protocol-specific call processor in the first switch.

10. The computer-readable storage medium as recited in claim **1**, further comprising instructions to cause the network to:

maintain a common registry of assigned identifiers.

11. The computer-readable storage medium as recited in claim **10**, further comprising instructions to cause the network to:

register each end system or access port with the common registry, and

return a list of assigned identifiers from the common registry to each switch for the end systems or access ports of the respective switch.

12. The computer-readable storage medium as recited in claim **10**, further comprising instructions to cause the network to:

send a signal from the first switch to the common registry to resolve an end system or access port to its assigned identifiers.

13. The computer-readable storage medium as recited in claim **1**, further comprising instructions to cause the network to:

maintain the mapping at each switch for the end system or access ports of the respective switch.

14. The computer-readable storage medium as recited in claim **1**, further comprising instructions to cause the network to:

prior to assigning an identifier to a specific end system or access port, maintain a default identifier for that specific end system or access port which maps to predetermined access ports.

15. The computer-readable storage medium as recited in claim **1**, further comprising instructions to cause the network to:

maintain a multicast channel of connections between all or a subset of switches and wherein step c(iii) comprises forwarding the encapsulated packet on the multicast channel.

16. The computer-readable storage medium as recited in claim **15**, wherein the channel includes:

a point-to-multipoint connection from each switch to all other switches in the network.

17. The computer-readable storage medium as recited in claim **1**, further comprising instructions to cause the network, at step c(iii), to provide a distributed service in the switches for forwarding the encapsulated data packet.

18. The computer-readable storage medium as recited in claim **1**, further comprising instructions to cause the network to assign the identifier based on a policy work group definition.

19. The computer-readable storage medium as recited in claim **1**, further comprising instructions to cause the network to:

maintain at least one mapping table at each switch for performing the mapping step.

20. The computer-readable storage medium as recited in claim **19**, further comprising instructions to cause the network to:

listen to end systems heard on respective access ports at each switch and maintain the end systems heard and their respective access ports in the mapping table at the respective switch.

21. The computer-readable storage medium as recited in claim **1**, further comprising instructions to cause the network to:

assign a reserved identifier without limitation as to end system and access port.

22. The computer-readable storage medium as recited in claim **1**, further comprising instructions to cause the network to:

maintain a Management Information Base (MIB) interface at each switch for programming at least one mapping table, the mapping table being used to perform the mapping step.

23. The computer-readable storage medium as recited in claim **22**, further comprising instructions to cause the network to:

use a Simple Network Management Protocol (SNMP) set message to maintain the mapping table at each switch.

24. The computer-readable storage medium as recited in claim **1**, further comprising instructions to cause the network to:

maintain a status table at each switch to enable and disable a respective subset.

25. A computer-readable storage medium comprising program instructions for restricting flooding of a data packet, of one of a broadcast, multicast and unknown destination type, in a switch to be used in a switched data communications network, the network to include end systems and switches connected by links, the switches having access ports connected to end systems and network ports connected to other switches, the program instructions causing the switch to:

a. assign at least one identifier to a respective subset of end systems;

b. map the at least one assigned identifier to an access port attached to at least one end system in the respective subset of end systems; and

c. when the data packet is received from a source end system at a receiving access port of the switch:

i) determine one or more identifiers associated with the source end system;

ii) encapsulate the data packet by adding a header with the one or more determined identifiers;

iii) forward the encapsulated data packet to all or a subset of other switches in the network; and

iv) determine if at least one access port other than the receiving access port on the switch is associated with the one or more determined identifiers and forward the data packet out the at least one determined access port.

26. The computer-readable storage medium as recited in claim **25**, further comprising instructions to cause the switch to:

d. when an encapsulated data packet is received:

i) strip the header from the encapsulated data packet and determine the one or more encapsulated identifiers in the header of the encapsulated data packet;

ii) determine if at least one access port of the switch is associated with the one or more encapsulated identifiers; and

iii) forward the data packet out the at least one determined access port of the switch.

27. The computer-readable storage medium as recited in claim **26**, further comprising instructions to cause the switch to:

listen to end systems heard on the access ports and to maintain the end systems heard and their respective access ports in a mapping table.

28. The computer-readable storage medium as recited in claim **25**, further comprising instructions to cause the switch to, if in step c(iv) no other access port is determined, discard the data packet.

29. The computer-readable storage medium as recited in claim **25**, further comprising, in step b, instructions to cause the switch to:

maintain a first table to relate the at least one assigned identifier to the end systems or access ports of the switch; and

maintain a second table to relate the access ports of the switch to assigned identifiers.

30. The computer-readable storage medium as recited in claim **29**, further comprising, in step c.i), instructions to cause the switch to:

review the first table for the one or more identifiers associated with the source end system or the receiving access port.

31. The computer-readable storage medium as recited in claim **30**, further comprising, in step c.iv), instructions to cause the switch to:

review the second table for an access port associated with the one or more determined identifiers.

32. The computer-readable storage medium as recited in claim **25**, further comprising instructions to cause the switch to:

prior to assigning an identifier to a specific end system or access port, maintain a default identifier for that specific end system or access port which maps to predetermined access ports.

33. The computer-readable storage medium as recited in claim **25**, further comprising instructions to cause the switch to:

maintain a Management Information Base (MIB) interface.

34. The computer-readable storage medium as recited in claim **33**, further comprising instructions to cause the switch to:

use a Simple Network Management Protocol (SNMP) message to maintain a mapping table.

35. The computer-readable storage medium as recited in claim **25**, further comprising instructions to cause the switch to:

maintain a status table to enable and disable a respective subset.

* * * * *

Exhibit 5

US006539022B1

(12) **United States Patent**
Virgile

(10) **Patent No.: US 6,539,022 B1**
(45) **Date of Patent: *Mar. 25, 2003**

(54) **NETWORK DEVICE WITH MULTICAST FORWARDING DATA**

(75) Inventor: **Kenneth Virgile**, Lexington, MA (US)

(73) Assignee: **Enterasys Networks, Inc.**, Rochester, NH (US)

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

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **09/299,970**

(22) Filed: **Apr. 26, 1999**

Related U.S. Application Data

(63) Continuation of application No. 08/766,784, filed on Dec. 13, 1996, now Pat. No. 5,898,686, and a continuation of application No. 08/428,403, filed on Apr. 25, 1995, now Pat. No. 5,608,726.

(51) **Int. Cl.**[7] **H04L 12/54**
(52) **U.S. Cl.** **370/401**; 370/432
(58) **Field of Search** 370/254, 235, 370/389, 390, 392, 395.2, 395.3, 395.31, 401, 428, 429, 432, 463; 709/249, 250

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,627,052	A	* 12/1986	Hoare et al.	370/402
5,136,580	A	* 8/1992	Videlock et al.	370/403
5,331,637	A	7/1994	Francis et al.	370/54
5,355,375	A	10/1994	Christensen	370/85.2
5,361,256	A	* 11/1994	Doeringer et al.	370/390
5,396,493	A	3/1995	Sugiyama	
5,400,326	A	3/1995	Smith	
5,428,615	A	6/1995	Backes et al.	
5,434,855	A	7/1995	Perlman et al.	
5,448,565	A	9/1995	Chang et al.	
5,481,540	A	1/1996	Huang	
5,500,860	A	3/1996	Perlman et al.	
5,511,168	A	* 4/1996	Perlman et al.	709/243
5,517,494	A	5/1996	Green	
5,517,620	A	* 5/1996	Hashimoto et al.	709/242
5,530,703	A	* 6/1996	Liu et al.	370/255
5,608,726	A	3/1997	Virgile	

FOREIGN PATENT DOCUMENTS

WO	WO 95/01023	1/1995

OTHER PUBLICATIONS

D.R. cheriton and S.E. Deering, "Host Groups: A Multicast Extension for Datagram Internetworks", *Proceedings Ninth Data Communications Symposium*, Whistler Mountain, British Columbia, pp. 172–179, (Sep. 10–13, 1985).

Ninth Data Communications Symposium, (Sep. 1985), USA, pp. 172–179, D.R. et al.: "Host groups: a internetworks".

Symposium on Communication Architectures and Protocols, (Aug. 1988), USA, pp. 55–64, S.E. Deering: "Multicast routing in internetworks and extended LANS" cited in the application.

S. Deering, *Multicast Routing in Internetworks and Extended LANs*, ACM Symposium on Communication Architectures and Protocols, ACM SIGCOMM, pp. 55–64, (Aug. 1988).

Cisco Systems Inc., *I.P. Multicast Streamlines Delivery of Multicast Applications*, The Packet: CISCO Systems Use Magazine, vol. 7, No. 1, pp. 7–9 (1995).

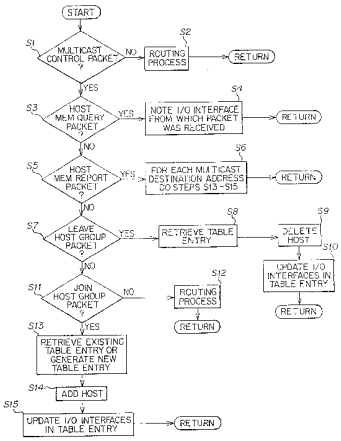
* cited by examiner

Primary Examiner—Kwang Bin Yao
(74) *Attorney, Agent, or Firm*—Wolf, Greenfield & Sacks, P.C.

(57) **ABSTRACT**

A system and method are disclosed for routing multicast packets in a subnetwork so as to conserve bandwidth in at least some of the network segments or collision domains of the subnetwork. In particular, multicast packets are only retransmitted in the network segments that are on a path to a host that is a member of the multicast group of hosts to which the multicast packet is destined.

20 Claims, 4 Drawing Sheets



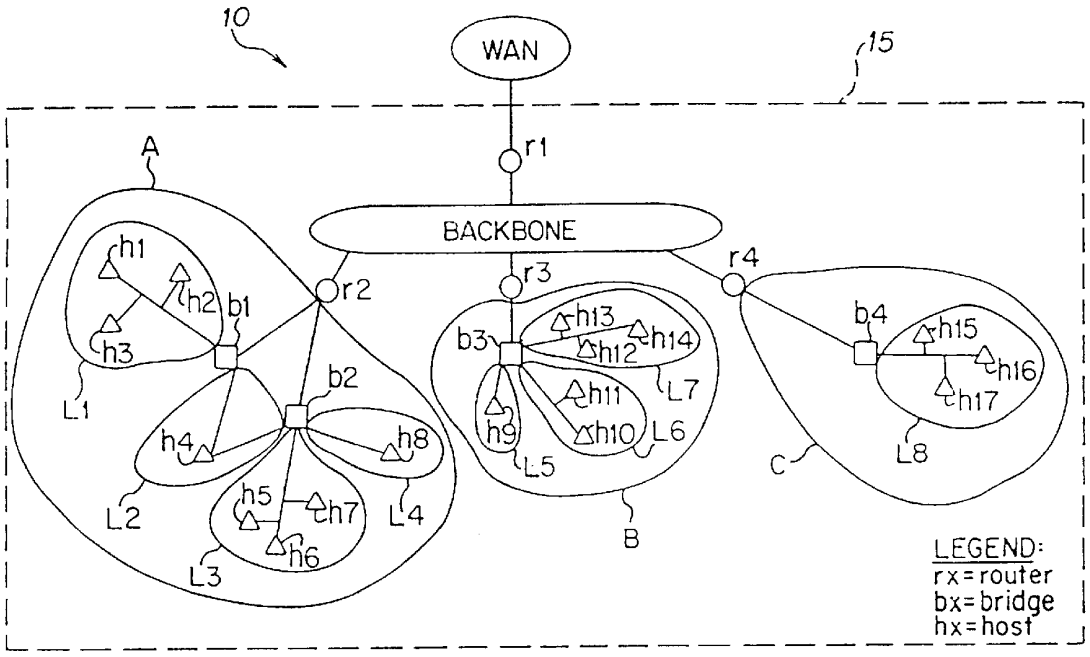


FIG. 1
(PRIOR ART)

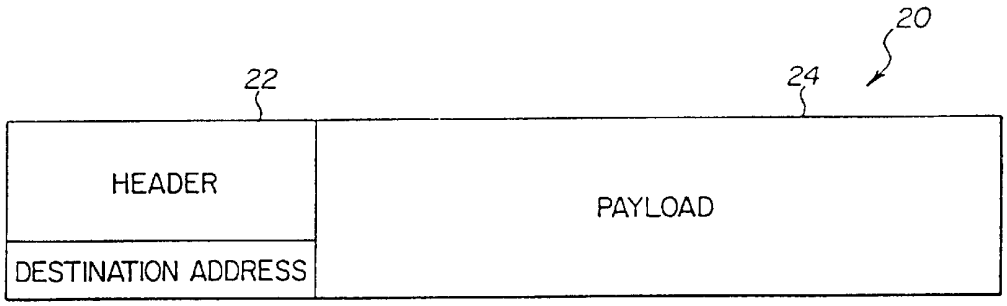


FIG. 2
(PRIOR ART)

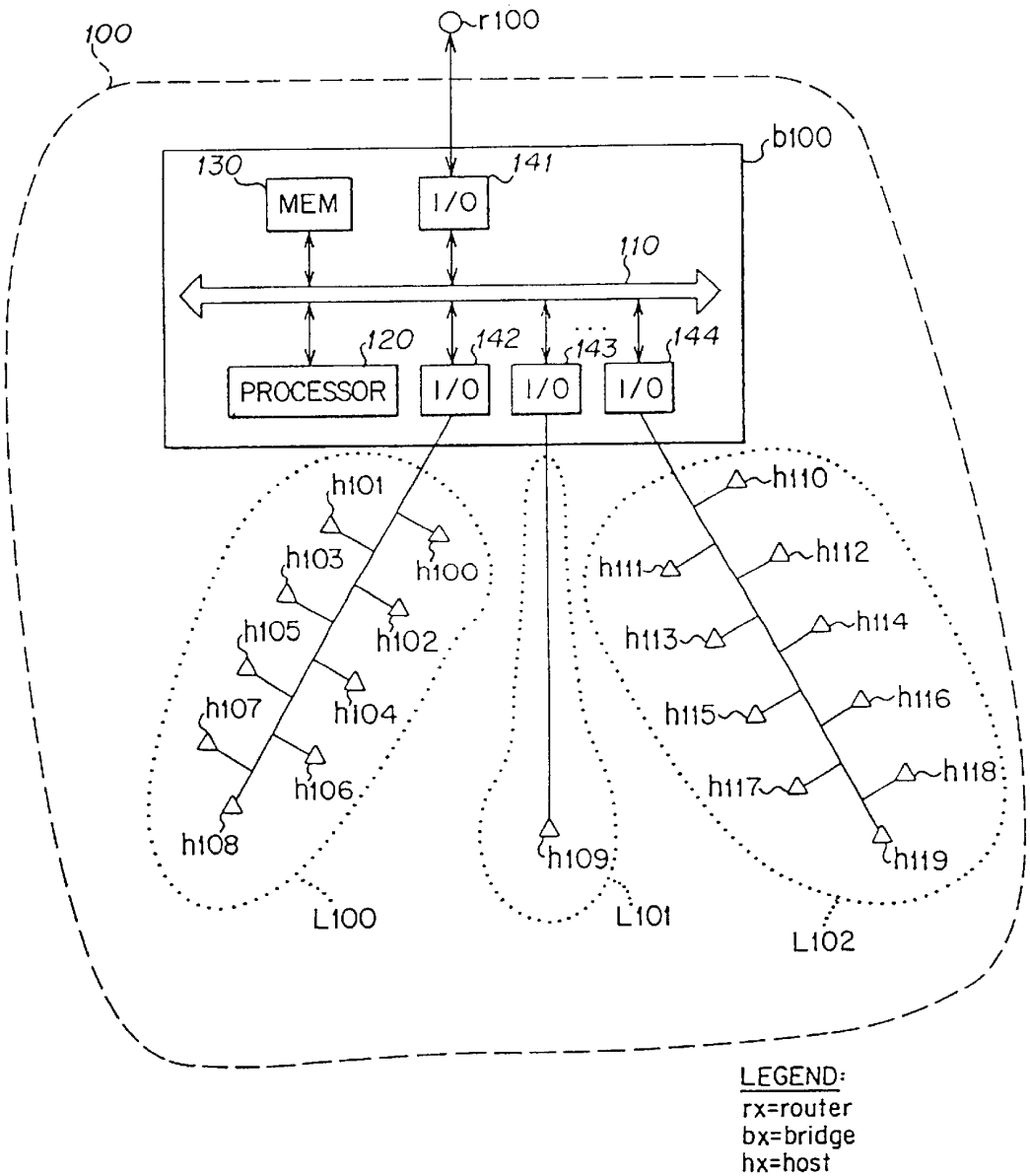

100
r100
b100
130
MEM
141
I/O
110
120
PROCESSOR
142
I/O
143
I/O
144
I/O
h101
h103
h100
h105
h102
h107
h104
h106
h108
h109
h110
h111
h112
h113
h114
h115
h116
h117
h118
h119
L100
L101
L102
LEGEND:
rx=router
bx=bridge
hx=host

FIG. 3

	MULTICAST FORWARDING TABLE		
	INDEX FIELD	I/O INTERFACE FIELD	LIST FIELD
210	A1 212	142,143,144,141 214	h108,h109,h110 216
220	A2 222	144,141 224	h112 226
230	AV1 232	143,144,141 234	h109,h114,h117,h119 236
240	AV2 242	142,144,141 244	h101,h102,h103,h113,h114 246
250	V1 252	142,141 254	h107 256
260	V2 262	142,143,141 264	h104,h109 266
270	V3 272	143,141 274	h109 276
280	A3 282	142,141 284	h100 286

200

FIG. 4

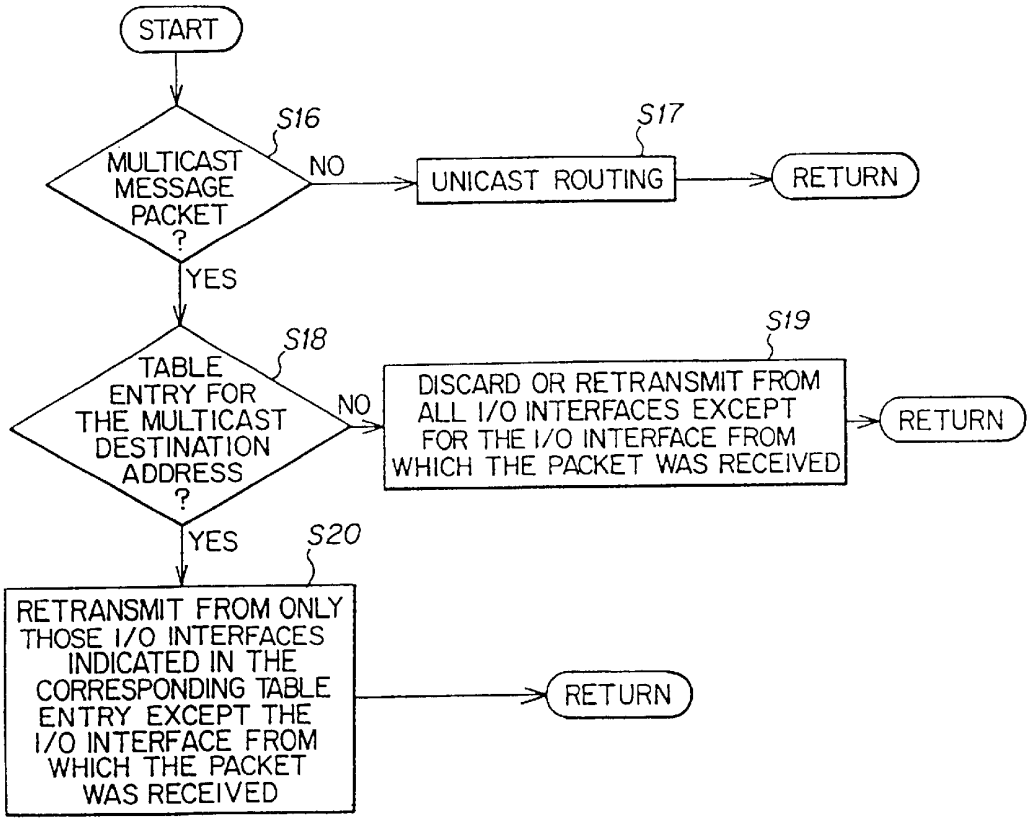


FIG. 6

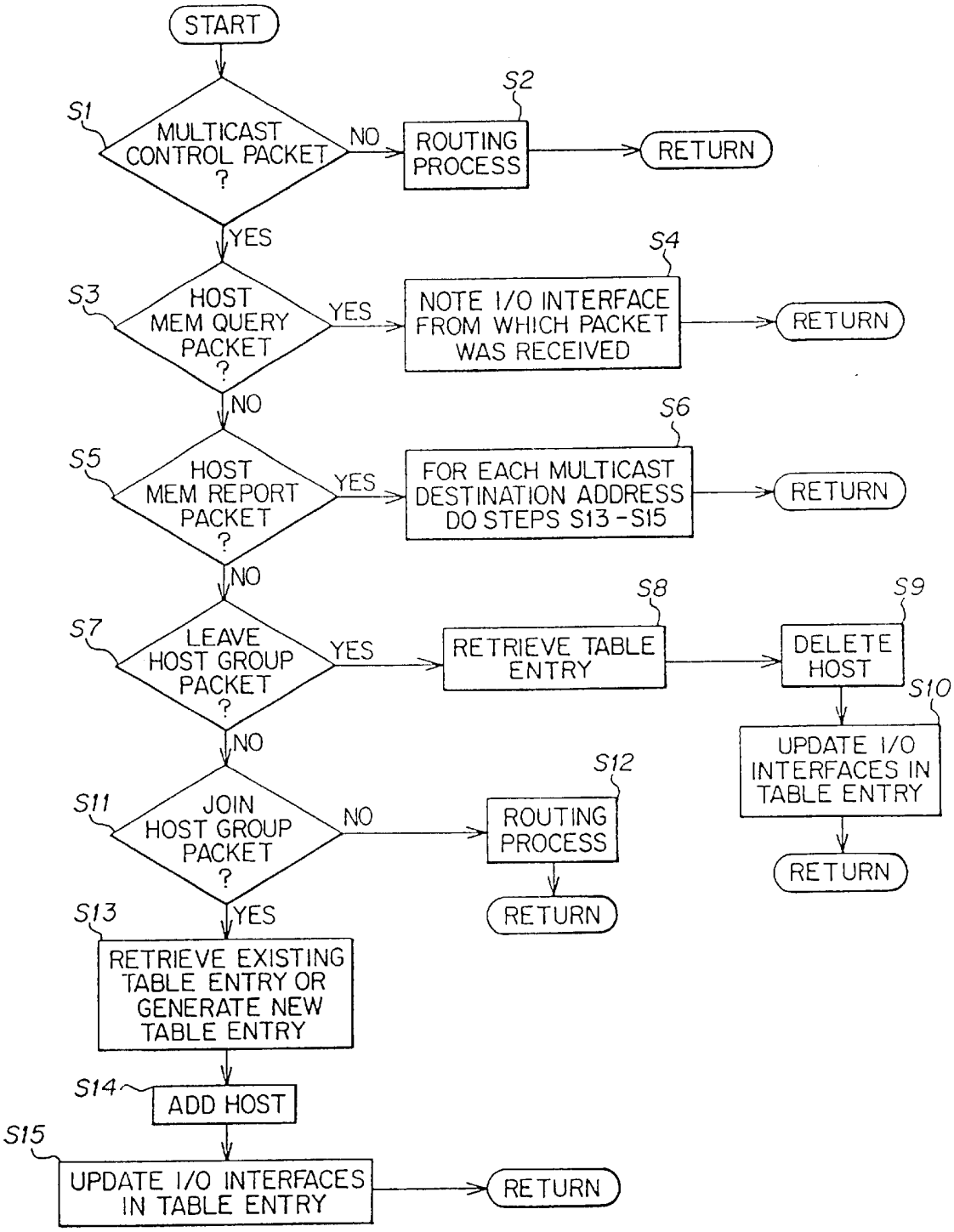

START
S1
MULTICAST CONTROL PACKET ?
NO
S2
ROUTING PROCESS
RETURN
YES
S3
HOST MEM QUERY PACKET ?
YES
S4
NOTE I/O INTERFACE FROM WHICH PACKET WAS RECEIVED
RETURN
NO
S5
HOST MEM REPORT PACKET ?
YES
S6
FOR EACH MULTICAST DESTINATION ADDRESS DO STEPS S13 - S15
RETURN
NO
S7
LEAVE HOST GROUP PACKET ?
YES
S8
RETRIEVE TABLE ENTRY
S9
DELETE HOST
S10
UPDATE I/O INTERFACES IN TABLE ENTRY
RETURN
NO
S11
JOIN HOST GROUP PACKET ?
NO
S12
ROUTING PROCESS
RETURN
YES
S13
RETRIEVE EXISTING TABLE ENTRY OR GENERATE NEW TABLE ENTRY
S14
ADD HOST
S15
UPDATE I/O INTERFACES IN TABLE ENTRY
RETURN

FIG. 5

NETWORK DEVICE WITH MULTICAST FORWARDING DATA

CROSS REFERENCE TO RELATED APPLICATION

This patent application is a continuation of U.S. patent application Ser. No. 08/766,784, filed Dec. 13, 1996 now U.S. Pat. No. 5,898,686, and also a continuation of U.S. patent application Ser. No. 08/428,403, filed Apr. 25, 1995 now U.S. Pat. No. 5,608,726.

FIELD OF THE INVENTION

The present invention relates to a communications network that transmits messages therein. The present invention especially relates to the transmission of multicast packets in such a network.

BACKGROUND OF THE INVENTION

FIG. **1** shows a conventional hierarchically organized communications network **10**. In particular, the communications network has a wide area network (WAN), such as the Internet, at the highest level of the hierarchy. A campus network **15** is interconnected to the WAN by a router r**1**. The campus network **15** is so called because it is typically located at a single geographic campus of several buildings. The interconnections may include any combination of wires, coaxial cables, optical fibers, circuit switches, packet switches, etc. The router r**1** interconnects a backbone network of the campus network **15** to the WAN. Connected to the backbone network are additional routers r**2**, r**3** and r**4**. Each additional router r**2**, r**3** and r**4** connects a respective subnetwork A, B or C to the backbone network. The campus network **15** is at a middle level of the hierarchy and subnetworks A, B and C of the campus network **15** are at a lower level in the hierarchy. The subnetworks A, B, C are typically isolated to a single, small geographic area such as an office building or floor of an office building. The WAN, on the other hand, typically sprawls from geographic area to geographic area. The WAN itself typically includes a number of routers (not shown) for routing communications from campus network to campus network.

The communication between routers on the WAN and on the backbone network is illustratively achieved according to the Internet protocol (IP). (Herein, protocol means a collection of semantic and syntactic rules obeyed by the devices which communicate according to the protocol).

The router r**2** is connected to bridges b**1** and b**2** of the subnetwork A, the router r**3** is connected to bridge b**3** of the subnetwork C. Each bridge b**1**–b**4** is connected to one or more network segments or collision domains which illustratively are local area networks (LANs). The bridge b**1** is connected to network segments L**1** and L**2**, the bridge b**2** is connected to network segments L**2**, L**3** and L**4**, the bridge b**3** is connected to network segments L**5**, L**6** and L**7** and the bridge b**4** is connected to network segment L**8**. Each network segment L**1**–L**8** comprises one or more interconnected host computers h**1**–h**17**. The network segment L**1** includes hosts h**1**, h**2** and h**3**, the network segment L**2** includes the host h**4**, the network segment L**3** includes the hosts h**5**, h**6** and h**7**, the network segment L**4** includes the host h**8**, the network segment L**5** includes the host h**9**, the network segment L**6** includes the hosts h**10** and h**11**, the network segment L**7** includes the hosts h**12**, h**13** and h**14** and the network segment L**8** includes the hosts h**15**, h**16** and h**17**. The network segments L**1**–L**8** may be Ethernet LANs, token ring LANs or FDDI LANs, for example.

Communication may be achieved locally within each network segment L**1**–L**8** according to one of a number of protocols. Since most deployed network segments L**1**–L**8** are Ethernet LANs, the Ethernet protocol for communication is used to illustrate the invention. According to the Ethernet protocol, each host computer is connected via an I/O interface to a common broadcast medium which broadcast medium may be carried by a coaxial cable, unshielded twisted pairs of wires, etc. A host communicates on the medium by transmitting a bitstream organized into packets FIG. **2** illustrates an illustrative packet **20**, which comprises a header section **22** and a payload section **24**. A host which desires to communicate writes data in the payload section **24**, and an address of the intended recipient host in the header section **22**. (Illustratively, all hosts on a network segment are assigned a unique identifier or address). If the common broadcast medium is not currently being used, then the host transmits its packet **20** from an I/O interface connected to the host onto the common broadcast medium. If the common broadcast medium is currently being used by another host to transmit a packet, then the host waits until the common broadcast medium is available. The transmitted packet **20** is received at the I/O interface of each other host on the network segment. Each host then examines the destination address written in the header section **22** of the packet **20**. If the destination address matches the destination address of the host, the host accepts the packet **20** and may examine the contents of the payload section **24**. If the destination address does not match, the host discards the packet **20**.

It is possible that two hosts of the same network segment may attempt to transmit a packet concurrently. If this happens, a collision is said to occur. According to the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol, in the event of a collision, each host transmits a jamming signal for a specified period of time and waits a variable amount of time before reattempting transmission of its packet. See U.S. Pat. No. 5,355,375. Collisions are only possible within an individual collision domain or network segment. For instance, the communication amongst the hosts h**1**–h**3** and bridge b**1** in the network segment L**1** does not effect the communication amongst the host h**4** and bridge b**1** in the network segment L**2**. The delay incurred in transmitting a packet on a network segment (as caused by collisions or otherwise) increases with the increase of communications traffic on each segment.

Sometimes it is desirable to communicate from a host in one network segment to a host in another network segment of the same subnetwork. Such communication may be achieved using the bridges b**1**–b**4**. The bridge enables inter-segment communication while isolating the two network segments so that they operate as independent collision domains. The bridges b**1**–b**4** also can enable communication between network segments that communicate according to different protocols. For instance, the bridge b**1** can enable communication by the host h**4** in the network segment L**2**, which is a Token Ring LAN, with the host h**2** in the network segment L**1**, which is an Ethernet LAN.

The bridges within a subnetwork (e.g., the bridges b**1**–b**2** in the subnetwork A) pass control packets between each other to determine the best route for reaching each host in each attached network segment. Thereafter, each bridge receives each packet transmitted on its attached network segments. If the bridge (e.g., bridge b**1**) receives a packet from one network segment (e.g., L**1**) containing a destination address of a host in another network segment, (e.g., the host h**8** in network segment L**4**) the bridge transmits the

packet in another attached network segment on a route to the network segment that contains the destination host. For instance, the route from host h1 in network segment L1 to the host h8 in the network segment L4 illustratively comprises: h1-L1-b1-L2-b2-L4-h8. Thus, the bridge b1 retransmits a packet received from host h1 destined to host h8 on network segment L2.

In addition to enabling inter-segment communication within a subnetwork, the bridges enable the hosts in one subnetwork to communicate with the hosts in other subnetworks. To that end, IP (Internet protocol) addresses may be assigned to each host which includes the destination host's address in the particular subnetwork concatenated to at least a unique address that is assigned to the subnetwork in which the destination host is located. As an example, suppose the host h1 in the subnetwork A wishes to transmit a packet to the host h15 in the subnetwork C. The host h1 writes the IP address of the node h15 (which includes at least the destination address of the subnetwork C concatenated to the destination address of the host h15) in a packet and transmits the packet on its network segment L1. The packet is received at, amongst other places, the bridge b1. If a bridge (e.g., the bridge b1) in a particular subnetwork receives a packet with a destination address that is foreign to the particular subnetwork, the bridge transmits the packet to its attached router (e.g., r2). The packet is then transmitted via the backbone network to the router that connects to the subnetwork containing the destination host. For instance, the packet may be transmitted from the router r2 via the backbone network to the router r4. To that end, each router which receives a packet illustratively uses the destination address (or a portion thereof) to index a routing table stored at the router. The indexed router table entry indicates the next router to which the packet must be transmitted. When the packet reaches the router (e.g., router r4) that attaches the subnetwork (e.g., subnetwork C) containing the destination host (e.g., host h15), the router transfers the packet to the attached bridge (e.g., b4). The bridge then transmits the packet to the destination host.

The discussion above has been limited to unicast packet communication wherein a packet is transmitted from a single source host to a single destination host. The network 10 also supports multicast communication, wherein a packet is transmitted from a single source host to multiple hosts. U.S. Pat. No. 5,331,637 describes multicast routing and, in particular, how to implement multicast routing at the WAN level of the hierarchy. Illustratively, multicast communication of packets is supported in the communications network 10 at the network level of communications according to the Internet Group Management Protocol (IGMP), IETF RFC 112, *Host Extensionsfor IP Multicasting* According to this protocol, multicast groups of hosts are identified, wherein each group is a collection of destination hosts for packets for a particular communication. Each multicast group is assigned a special multicast address which bears no relation to any single host of the multicast group.

Each router which connects a campus network 15 to the WAN (e.g., the router r1) periodically transmits a "Host Membership Query" multicast control packet with a destination address that specifies all of the hosts of the campus network 15. In response, each host transmits back to the router (e.g., the router r1) a "Host Membership Report" multicast control packet that indicates all of the groups to which the host belongs. Furthermore, a host can transmit a "Join Host Group" or "Leave Host Group" multicast control packet to the router (e.g., the router r1) at any time to join or leave a multicast group. The router receives these messages and updates its routing tables accordingly.

When a host, e.g., the host h1, desires to transmit a multicast packet, it writes a multicast address of an appropriate multicast group in the destination field. The host then transmits the packet to its attached bridge, e.g., the bridge b1 (via the subnetwork L1). The bridge has no way of knowing the location of the destination host. (Because the multicast destination address bears no relationship to the destination address of a single host). Thus, the bridge retransmits the multicast packet to each attached subnetwork and router, other than the subnetwork or router from which the packet originated, e.g., the subnetworks L2, L3 and the router r2. The attached router, e.g., the router r2, accesses its routing table using the multicast destination address. However, unlike before, the accessed routing table entry may indicate more than one next router to which the packet must be transmitted, e.g., the router r1 and the router r3. The router transmits a copy of the packet to each indicated next router. Thus, the packet is selectively routed and replicated in route. Each router that receives a copy of the multicast packet performs the same table access procedure. Eventually, a router, e.g., the router r4, receives a packet that must be transmitted to an attached subnetwork, e.g., the subnetwork C.

When a multicast packet is received at a bridge of the subnetwork, e.g., the bridge b4 of the subnetwork C, the bridge has no way of knowing to which attached network segment (or router) the packet is destined. This is because the packet has a multicast address which bears no relationship to any individual host. Furthermore, the multicast packet can be destined to more than one host in more than one attached network segment and or router. Thus, the bridge retransmits the multicast packet in each attached network segment and to each attached router. The packets transmitted in the network segments are received by each host. Each host then compares the multicast destination address to the multicast destination addresses of the groups of which it is a member. If the host is a member of the same group as indicated in the packet, the host receives the packet. Otherwise, the host discards the packet.

The problem with the above noted multicast communication scheme is that it wastes bandwidth in the subnetworks. In particular, a bridge retransmits a received multicast packet in each attached network segment even if one of the attached network segments is devoid of destination hosts of the multicast packet (i.e., even if the network segment does not have any hosts that are members of the multicast group of the multicast packet). This results in unnecessary bandwidth reduction in some attached network segments that are devoid of destination hosts. Considering that much multicast traffic in the future is intended to be bandwidth intensive multimedia traffic, i.e., video and/or audio, the wasted bandwidth can be very high and can noticeably degrade performance on a network segment. In the past, the solution to improving network segment performance is to reconfigure the campus network by increasing the number of routers and redistributing (i.e., reconnecting) the network segments or hosts amongst the routers. However, this solution is disadvantageous because routers are relatively expensive and difficult to manage.

S. Deering, *Multicast Routing in Internetworks and Extended LANs,* ACM SYMPOSIUM ON COMMUNICATION ARCHITECTURES AND PROTOCOLS, ACM SIGCOMM pp. 55–64 August 1988 proposes an alternative solution. According to the Deering reference, bridges only retransmit multicast packets over "links" on routes to a destination host of the multicast packets (wherein a "link" is a communication connection). To that end, each bridge

constructs a multicast forwarding table which is maintained at the bridge. The bridge accesses the multicast forwarding table using the multicast group as an address to determine onto which links a received multicast packet must be retransmitted. The Deering reference teaches that the hosts transmit special control packets that are destined to all bridges of the campus network indicating to which multicast group the host belongs. The bridges compile such information, in order to construct the multicast forwarding table and to determine when entries of the multicast forwarding table have become stale and therefore must be discarded.

There are two problems with the proposed Deering solution. First, extra control packets must be transmitted between the hosts and the bridges in order to construct the multicast forwarding tables. This increases traffic on the network segments. Second, and more importantly, all hosts are specially adapted in accordance with the Deering scheme so that they periodically transmit the special multicast control packets in order to maintain their memberships. The solution is therefore not entirely "plug-and-play" from the perspective of the hosts. It is therefore an object of the present invention to overcome the disadvantages of the prior art. It is a particular object of the present invention to prevent multicast communication traffic that originates from, or is destined to, outside of a subnetwork from degrading the communication performance within the subnetwork.

SUMMARY OF THE INVENTION

According to a first embodiment, a network device for reducing multicast communications in a network includes at least two I/O interfaces to each receive and transmit data packets and also includes multicast forwarding data. The device includes means for determining, by examining data in a first portion of a first packet, whether the first packet is a control packet destined for a device other than the network device; means for identifying a multicast group from the received first packet when the first packet is a control packet type; and means for updating the multicast forwarding data according to the identified multicast group received from the multicast group identifying means and the data in the first portion of the first packet.

According to a second embodiment, a method of reducing multicast communications in a network device, the device including multicast communications forwarding data and at least two I/O interfaces to each receive and transmit data packets, comprises: determining the I/O interface on which a first packet arrived; and determining, by examining data in a first portion of the first packet, whether the first packet is a control packet addressed to a device other than the network device. When the first packet is determined to be a control packet addressed to a device other than the network device: determining a multicast group from the first portion of the control packet; and updating the multicast forwarding data as a function of data in the first portion of the control packet.

According to a third embodiment, a method of reducing multicast communications in a network, the network including a network device, the network device having at least two I/O interfaces to each receive and transmit data packets and the network device including multicast forwarding data, includes (a) determining, by examining data in a first portion of a first packet, whether the first packet is a control packet destined for a device other than the network; and when the first packet has been determined to be a control packet: (b) identifying a multicast group from the data is the first portion of the first packet; and (c) updating the multicast forwarding data according to the data in the first portion of the first packet.

According to yet another embodiment, a method of conserving bandwidth in a communications network by reducing transmission of unneeded multicast messages comprises: monitoring, at a first location, control packets destined for another location other than the first location; from each monitored control packet, identifying a multicast group, a source of the each monitored control packet and an interface on which the each monitored control packet arrived; maintaining an association of the identified multicast group with the identified interface; and for any multicast message, received at the first location and destined for the identified multicast group, transmitting the received multicast message on any interface associated with the multicast group identified in the multicast message.

This provides a robust solution for preventing multicast packets from flooding all attached network segments without requiring that the individual hosts be adapted in any special fashion. In short, an apparatus and method are provided for controlling the retransmission of multicast packets within a subnetwork to conserve bandwidth. Thus, multicast packets originating from, or destined to, outside the subnetwork do not interfere with the network segments that do not contain any hosts that belong to the multicast group of the multicast packets.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** shows a conventional communications network.

FIG. **2** shows a conventional packet.

FIG. **3** shows a subnetwork including a bridge according to the present invention.

FIG. **4** shows an illustrative forwarding table according to an embodiment of the present invention.

FIG. **5** is a flowchart which schematically illustrates a multicast forwarding table maintenance process according to an embodiment of the present invention.

FIG. **6** is a flowchart which schematically illustrates a multicast message packet routing process according to an embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

FIG. **3** shows a subnetwork **100** according to the present invention. As shown, the subnetwork **100** is connected to a router r**100** of a campus network or directly to a WAN. The subnetwork **100** includes a bridge b**100** connected to three network segments, namely, L**100**, L**101** and L**102**. The network segment L**100** includes a number of hosts h**100**, h**101**, h**102**, h**103**, h**104**, h**105**, h**106**, h**107** and h**108** interconnected to form a LAN such as an Ethernet LAN. Likewise, the network segment **102** includes a number of hosts h**110**, h**111**, h**112**, h**113**, h**114**, h**115**, h**116**, h**117**, h**118**, h**119** connected to form a LAN. The network segment L**101** only includes the host h**109** which is also connected to form a LAN.

As shown, the bridge b**100** includes a bus **110**, a processor **120**, a memory **130** and four I/O interfaces **141**, **142**, **143** and **144**. However, this is illustrative. Any arbitrary number of I/O interfaces may be provided to support each router and network segment attached to the bridge b**100**. The I/O interface **141** is connected to the router r**100**. The I/O interface **142** is connected to the network segment L**100**. The I/O interface **143** is connected to the network segment L**101**. The I/O interface **144** is connected to the network segment L**102**. Each I/O interface **141**–**144** is capable of receiving packets and transmitting packets according to the

protocol of the router or network segment to which the I/O interface is attached. For instance, the I/O interface **142** can transmit packets, receive all packets, detect a collision on the network segment L**100** and transmit a jamming signal, if necessary, as per the Ethernet and CSMA/CD protocols. Thus, the I/O interfaces **141–144** enable the bridge b**100** to receive packets from the router r**100** and network segments L**100**, L**101** and L**102** and to transmit packets thereto.

Packets which may be received via the I/O interfaces **141–144** may be transferred via the bus **110** to the processor **120** or memory **130**. Illustratively, the memory **130** is for, amongst other things, temporarily buffering received packets. The processor **120** can access the packets in the memory **130**, for example, to read or write data in the header portions of the packets for purposes of translating a packet from one protocol to another protocol. The processor **120** can also examine the address of the packet header to determine from which I/O interface **141–144** the packet should be retransmitted. By selectively retransmitting unicast packets from only the I/O interfaces **141–144** connected to a network segment (or router) containing the destination host of the packet, the bridge b**100** conserves the bandwidth of the other network segments (or routers) which do not contain the destination host of the packet.

According to an embodiment of the present invention, multicast packets received at the bridge b**100** (from the router r**100** or from a host h**100–h119** in an attached network segment L**100**, L**101** or L**102**) are only retransmitted onto the network segments L**100**, L**101** or L**102** that contain destination hosts of the multicast packets. Stated another way, the bridge b**100** refrains from retransmitting multicast packets onto those attached network segments L**100**, L**101** or L**102** that are devoid of destination hosts of the multicast packets. To that end, the processor **120** maintains a forwarding table with entries corresponding to multicast addresses. Alternatively, additional forwarding table entries as described below may be added to a forwarding table already maintained by the bridge b**100**.

An illustrative forwarding table **200** is shown in FIG. **4**. As shown, each table entry **210**, **220**, **230**, **240**, **250**, **260**, **270**, and **280** includes a multicast destination address index field **212**, **222**, **232**, **242**, **252**, **262**, **272**, and **282**. The multicast destination address index field contains a multicast destination address of a particular multicast group. This field is used by the processor **120** for retrieving a multicast forwarding table entry **210–280** that corresponds to a particular multicast group. The table entry also includes an I/O interface field **214**, **224**, **234**, **244**, **254**, **264**, **274** and **284**. This field is for storing indications of only those I/O interfaces **142–144** that connect the bridge b**100** to a network segment that contains a destination host of the multicast address. This field **214–284** also stores indications of I/O interfaces **141** that connect the bridge b**100** to a router r**100** that provides communication access external to the network segment **100**. Furthermore, each table entry **210–280** includes a list field **216**, **226**, **236**, **246**, **256**, **266**, **276** or **286** for storing a list of the hosts which are members of the multicast group corresponding to the table entry. For example, the multicast forwarding table entry **230** corresponds to the multicast group for an audio-video teleconference which has the address AV**1** stored in the multicast destination address index field **232**. The multicast forwarding table entry **230** also includes the indications of the I/O interfaces **143**, **144** and **141** (from which the packets must be retransmitted) stored in the I/O interface field **234**. Furthermore, the multicast forwarding table entry **230** includes a list of the hosts h**109**, h**114**, h**117** and h**119** which are members of the multicast group AV**1** stored in the host list field **236**.

FIG. **5** is a flowchart that illustrates a process executed by the bridge b**100** in constructing and modifying the multicast forwarding table entries **210–280** of FIG. **4**. In a first step S**1**, the processor **120** determines if a received packet is a multicast control packet. Examples of such packets according to the IGMP protocol are Host Membership Query packets, Host Membership Report packets, Join Host Group packets and Leave Host Group packets. Such packets have a well defined and easily identifiable form. If the received packet is not a multicast control packet, the processor executes a routing process (step S**2**) described below. If the received packet is a multicast control packet, the processor **120** executes step S**3**. In step S**3**, the processor **120** determines if the packet contains an instruction from the attached router to each host h**101–h119** to report all of its current multicast group memberships (e.g ., a Host Membership Query packet). If so, then the processor **120** keeps track of from which router and via which I/O interface **141–144** the packet was received, e.g., by temporarily storing this information in the memory **130**. As discussed in greater detail below, the processor **120** uses such information in the construction of forwarding table entries corresponding to multicast groups. In particular, such information is used to update the forwarding tables to ensure that multicast packets originating in the subnetwork **100**, and destined outside the subnetwork **100**, are transmitted from the bridge b**100** to the routers r**1–r4** that can deliver multicast packets to such destinations external to the subnetwork **100**. If the packet is not a Host Membership Query packet, the processor **120** proceeds to step S**5**. In step S**5**, the processor determines if the packet contains a report from a host of all of its current multicast group memberships (e.g., a Host Membership Report packet). Such a packet contains the multicast destination address of each multicast group of which the host wishes to be a member. If a Host Membership Report packet is received, then, in accordance with step S**6**, the processor **120** executes steps S**13–S15** for each multicast destination address (of each multicast group) specified in the packet. If the packet is not a Host Membership Report packet then the processor **120** proceeds to step S**7**.

In step S**7**, the processor **120** determines if the packet contains an indication that a host is leaving a multicast group (e.g., is a Leave Host Group packet). If not, the processor **120** executes step S**11**. If the packet does contain an indication that the host is leaving a multicast group, the processor **120** executes steps S**8–S10**. In step S**8**, the processor **120** retrieves the multicast forwarding table entry corresponding to the destination address of the multicast group designated in the received packet. For example, suppose the packet contained an indication that host h**109** desired to leave the multicast group AV**1**. The processor **120** would thus retrieve from the memory **130** the forwarding table entry **230** that has the multicast destination address AV**1** stored in the multicast destination address index field **232**. Next, in step S**9**, the processor **120** deletes the indication of the host which transmitted the packet from the list of hosts contained in the entry. For instance, the processor **120** would delete the indication for h**109** from the host list field **236** of the multicast forwarding table entry **230**. Thereafter, the processor **120** executes step S**10**, wherein the processor **120** updates the indicators stored in the I/O interface field **234** of the I/O interfaces from which the packet must be transmitted. In particular, the processor **120** examines each remaining host indication contained in the host list field **236**. If necessary, the processor **120** deletes an indicator for an I/O interface from the I/O interface field **234** if none of the remaining hosts in the host list are connected via that I/O

interface. Such would be the case if the host which transmitted the multicast control packet was formerly the only host on a particular network segment that belonged to that multicast group. Continuing with the above example, the host h**109** was formerly the only host of the list **236** connected to the bridge b**100** by the I/O interface **143**. Thus, the indication for the I/O interface **143** is deleted from the I/O interface field **234**.

In step S**11**, the processor **120** determines if the received packet contains an indication that a host wishes to join a particular multicast group (e.g., a Join Host Group packet). If not, then the packet must be some unidentifiable multicast control packet. In such a case, in step S**12**, the processor executes the routing process described below. If a multicast control packet for joining a multicast group was received, the processor **120** executes steps S**13**–S**15**. In step S**13**, the processor **120** first attempts to retrieve a multicast forwarding table entry using, as an index, the multicast forwarding address of the group that the host wishes to join. If the processor **120** is able to retrieve a corresponding multicast forwarding table entry from the memory **130**, the processor **120** then executes step S**14**.

However, it is possible that the host which transmitted the packet wishes to join a multicast group for which no multicast forwarding table entry exists at the bridge b**100**. Such is the case if the multicast forwarding table was recently flushed or if this host is the first host in the subnetwork **100** (FIG. **3**) to join this particular multicast group. If no multicast forwarding table entry exists for the multicast group, then the processor **120** generates one in the memory **130**. In so doing, the processor **120** writes the multicast destination address for the group in the multicast destination address index field. The processor **120** also writes the indication of the I/O interface S**141**–**144** connected to one or more routers, i.e., the interface **141**, that can deliver multicast traffic to destinations external to the subnetwork **100**, in the I/O interface field. This is so the bridge b**100** will retransmit multicast packets that are locally generated in the subnetwork **100** to other subnetworks via the router r**100**. As noted above, the processor **120** can determine which I/O interfaces **141** are connected to such routers r**100** in response to receiving Host Membership Query Packets (steps S**3**–S**4**). The processor **120** can thus use the previously stored indication of the host interface **141** of the router r**100** which issues such multicast query packets to construct the table entry. The processor **120** then executes step S**14**.

In step S**14**, the processor **120** adds the destination address of the host which transmitted the control packet to the host list field of the multicast forwarding table entry. Next, in step S**15**, the processor **120** updates the I/O interface indications in the I/O interface field of the multicast forwarding table entry in accordance with the revised host list. That is, suppose the newly added host is connected to the bridge b**100** via an I/O interface not already indicated in the I/O interface field. Such would be the case if the newly added host is the only multicast group member on a particular network segment. If such is the case, then an indicator for the I/O interface which connects the network segment of the newly added host must be added to the I/O interface field of the table entry.

After each of the steps S**4**, S**6**, S**10** and S**15**, the bridge also routes the received packet towards its intended destination. In the case of step S**4**, the multicast packet contains an "all hosts" address. This packet is thus retransmitted on every network segment on a route to a host of the subnetwork **100**. In the cases of steps S**6**, S**10** and S**15**, such multicast control packets are destined to an attached router, such as the router r**100**. The bridge b**100** retransmits such packets on a route to that router. The router in turn updates its internal routing tables in response to receiving such packets in a well known manner.

In any event, it should be noted that the hosts need not transmit special packets to the bridges. Rather, the bridges b**100** according to the present invention "eavesdrop" on the very same packets used to change host membership on the WAN or campus network level according to, for instance, the IGMP protocol. Thus, network segment bandwidth is conserved since a single multicast control packet serves two purposes. Moreover, neither the hosts nor routers need to be specially adapted to implement the invention. Rather, only the bridges need be adapted.

While not shown, the processor **120** also performs an "aging out" process. Generally speaking, the forwarding table **200** is only accurate for a limited period of time. This is due to various reasons, such as the turning on and off of hosts, the reconfiguration (the changing of the interconnection of routers, bridges and hosts) of the network **100**, the failing of a router, bridge, etc. Thus, the forwarding table **100** is presumed to be stale or inaccurate after a fixed period of time, e.g., every five minutes. At each such regular interval, the processor **120** flushes or discards the forwarding table **200** in the memory **130** and begins constructing a new forwarding table **200** as per the above-described process. See S. Deering, *Multicast Routing in Internetworks and Extended LANs, ACM* Symposium On Communication-Architectures and Protocols, ACM SIGCOMM pp. 55–64, August, 1988 for an exemplary "aging-out" process.

FIG. **6** shows a process executed by the bridge b**100** in routing, i.e., retransmitting received packets. In step S**16**, the processor **120** examines the received packet in the memory to determine if it is a multicast message packet (i.e., a packet for delivering data to multiple destinations). If not, the processor **120** executes step S**17** wherein the processor routes the packet using conventional unicast routing (described above). If the packet is a multicast message packet, the processor **120** executes step S**18**. In step S**18**, the processor **120** uses the multicast destination address as an index to retrieve a corresponding entry from the multicast forwarding table **200** (FIG. **4**). If the table does not have an entry corresponding to the multicast destination address of the received packet then the processor **120** executes step S**19**.

This may occur for various reasons including:

(a) a multicast message packet was received destined to a multicast group not known by the bridge b**100**,

(b) the processor **120** aged out the forwarding table **200**,

(c) the bridge b**100** was recently installed and has not yet determined all of the multicast groups,

(d) the bridge b**100** did not receive all of the multicast control packets, and

(e) the memory **130** ran out of storage locations and therefore could not complete the forwarding table **200**.

In step **19**, the processor **120** discards or retransmits the received packets depending on why the multicast destination group is not contained in the forwarding table **200**. For example, in the event of conditions (a)–(b), the received packet is discarded. On the other hand, in the event of conditions (c)–(e) the processor **120** retransmits the received packet in a conventional fashion, namely, the processor **120** retransmits the received packet from each I/O interface **142**–**144** other than the I/O interface **141** from which the packet was received.

If, on the other hand, the multicast forwarding table has an entry corresponding to the multicast address of the packet, the processor **120** executes step S**20**. In step S**20**, the processor **120** retrieves the indications of the I/O interfaces stored in the I/O interface field of the retrieved multicast forwarding table entry. The processor **120** then retransmits the received multicast packet only from the I/O interfaces **141**–**144** indicated in the I/O interface field except for the I/O interface from which the packet was received. For example, suppose the bridge b**100** receives a multicast packet from the router r**100** destined to the multicast group V**1**. This packet is received via I/O interface **141** and temporarily stored in the memory **130**. Using V**1** as an index, the processor **120** retrieves the table entry **250** corresponding to the multicast group of the address V**1**. The processor then retrieves all of the I/O interface indications from the I/O interface field **254**. The only indicated I/O interfaces are **141** and **142**. However, the packet was received from I/O interface **141**. Thus, the received multicast packet is only retransmitted from the I/O interface **142**. As such, the received multicast packet is retransmitted on the segment L**100** but not on the attached network segments L**101**, L**102**.

Consider now a second example, where the host h**101** wishes to transmit a multicast message packet to the multicast group AV**2**. The host **101** writes the multicast destination address AV**2** into the packet header and transmits the multicast message packet in the network segment L**100**. This packet is received at hosts h**102**–h**103** which accept the packet. The packet is also received at the I/O interface **142** of the bridge b**100**. The received packet is temporarily stored in the memory **130**. The processor **120** retrieves the multicast forwarding table entry **240**. The indicated I/O interfaces are **142**, **144**, and **141**. However, because the packet was received from I/O interface **142**, the processor **130** only retransmits the packet from I/O interfaces **144** and **141**. Thus, the packet is transmitted to the router r**100** (from which the packet may be transmitted to a host external to the subnetwork **100**) and to the hosts h**113**, h**114** via the network segment L**102**. However, the packet is not retransmitted on the network segment L**101**.

Note the improvement over the conventional multicast routing system wherein a conventional bridge simply retransmits all received multicast packets over each network segment (other than the network segment from which the packet was received). In contrast, according to the present invention, the received multicast packets are retransmitted from the bridge onto only those network segments containing hosts that are members of the pertinent multicast group. This is especially important in the context of multimedia, i.e., audio-video multicast packet communication traffic. Such traffic tends to be continuous rather than bursty and tends to require a high bandwidth.

The invention has been described so far with reference to only a single internetworking device (e.g., bridge) subnetwork. However, the invention is also equally applicable to a multiple internetworking device (e.g., multi-bridge) subnetwork.

In such a case, it is possible that some network segments are not directly connected to all of the bridges. For instance, consider the topology of the subnetwork A of FIG. **1**. As shown, the network segment L**1** is directly connected to bridge b**1** but not to bridge b**2**. Likewise, the network segments L**3** and L**4** are directly connected to the bridge b**2** but not the bridge b**1**. As noted above, the bridges b**1** and b**2** self configure routes to the hosts of subnetworks to which they are not directly attached by passing control packets back and forth to each other. For instance, the bridge b**1** may determine that packets destined to hosts in the subnetworks L**3** and L**4** may be transmitted thereto via the intervening network segment L**2** and the bridge b**2**.

Thus, in the multiple internetworking device topology, consideration must be made for routing to multicast destination hosts via intervening subnetworks. This can be achieved by modifying steps S**10** and S**15**. In particular, when the processor **120** updates the I/O interface field, the processor **120** ensures that only I/O interfaces that attach network segments on a route to a host of the host list are included therein. Such I/O interfaces include those that attach network segments which contain multicast destination hosts and those which attach intervening network segments on a route to multicast destination hosts.

In short, a system and method are disclosed for routing multicast packets in a subnetwork so as to conserve bandwidth in at least some of the network segments or collision domains of the subnetwork. In particular, multicast packets are only retransmitted in the network segments that are on a path to a host that is a member of the multicast group of hosts to which the multicast packet is destined.

Finally, the invention has been described above with reference to illustrative embodiments. Numerous alternative embodiments may be devised by those having ordinary skill in the art without departing from the spirit and scope of the following claims.

What is claimed is:

1. A network bridge for reducing multicast communications in a network, the network bridge including at least two I/O interfaces to each receive and transmit data packets and also including multicast forwarding data, the device comprising:

means for determining, by examining data in a first portion of a first packet received at the network bridge, whether the first packet is a multicast control packet destined for a device other than the network bridge;

means for identifying a multicast group from the first packet when the first packet is a multicast control packet destined for a device other than the network bridge; and

means for updating the multicast forwarding data according to the identified multicast group and the data in the first portion of the first packet.

2. The network bridge as recited in claim **1**, further comprising:

means for identifying the I/O interface on which the first packet arrived,

wherein the updating means include means for cross-referencing the identified I/O interface to the identified multicast group in the multicast forwarding data.

3. The network bridge as recited in claim **1**, further comprising:

means for identifying a source of the first data packet,

wherein the updating means include means for cross-referencing the identified source to the identified multicast group in the multicast forwarding data.

4. The network bridge as recited in claim **1**, further comprising:

means for identifying the I/O interface on which the first packet arrived; and

means for identifying a source of the packet,

wherein the updating means include means for cross-referencing the identified multicast group with the identified source and the identified I/O interface in the multicast forwarding data.

5. The network bridge as recited in claim 1, further comprising:

means for determining, when the first packet is not a control packet, whether the first packet is a message packet having a multicast group as a destination address; and

means for identifying, when the first packet is determined to be a multicast message packet, the destination multicast group;

the destination multicast group identifying means further comprising:

means for retrieving first data from the multicast forwarding data according to the identified destination multicast group, the retrieved first data including at least one cross-referenced I/O interface; and

means for sending the multicast message packet out each at least one cross-referenced I/O interface in the first data other than the identified I/O interface.

6. A method of reducing multicast communications in a network, the network including a network device having multicast communications forwarding data and at least two I/O interfaces to each receive and transmit data packets, the method comprising:

determining an I/O interface on which a first packet arrived;

determining, by examining data in a first portion of the first packet, whether the first packet is a multicast control packet addressed to a device other than the network device;

when the first packet is determined to be a multicast control packet addressed to a device other than the network device:

determining a multicast group from the first portion of the multicast control packet; and

updating the multicast forwarding data as a function of data in the first portion of the multicast control packet.

7. The method as recited in claim 6, wherein the step of updating the multicast forwarding data comprises:

cross-referencing the identified multicast group with the determined I/O interface.

8. The method as recited in claim 6, wherein the step of updating the multicast forwarding data comprises:

updating the multicast forwarding data by removing any cross-reference between the identified multicast group and the identified I/O interface.

9. The method as recited in claim 8, further including a step of:

updating the multicast forwarding data for the identified destination multicast group by deleting any cross-referenced I/O interface that is not connected to a device that is a member of the identified multicast group.

10. The method as recited in claim 6, wherein:

when the first packet is not a multicast control packet:

determining whether the first packet is a message packet having a multicast group as a destination; and

when the first packet is determined to be a multicast message packet:

identifying a destination multicast group from the multicast message packet;

retrieving first data from the multicast forwarding data according to the identified destination multicast group, the retrieved first data including at least one cross-referenced I/O interface; and

sending the multicast message packet out each at least one cross-referenced I/O interface in the retrieved first data other than the identified I/O interface.

11. The method as recited in claim 10, further comprising a step of:

when there is no data associated with the identified destination multicast group in the multicast forwarding data, transmitting the multicast message packet out all I/O interfaces other than the identified I/O interface.

12. The method as recited in claim 6, wherein the control packet is defined according to an Internet Group Management Protocol (IGMP) standard.

13. The method as recited in claim 12, wherein the multicast control packet is one of:

a host membership query packet;

a host membership report packet;

a join host group packet; and

a leave host group packet.

14. A method of reducing multicast communications in a network, the network including a network bridge, the network bridge having at least two I/O interfaces to each receive and transmit data packets and the network bridge including multicast forwarding data, the method including:

(a) determining, by examining data in a first portion of a first packet received at the network bridge, whether the first packet is a multicast control packet destined for a device other than the network bridge; and

when the first packet has been determined to be a multicast control packet destined for a device other than the network bridge:

(b) identifying a multicast group from the data in the first portion of the first packet; and

(c) updating the multicast forwarding data according to the data in the first portion of the first packet.

15. The method as recited in claim 14, further comprising:

identifying the one I/O interface on which the first packet arrived;

wherein, step (c) includes a step of updating the multicast forwarding data so as to cross-reference the identified I/O interface with the identified multicast group.

16. The method as recited in claim 14, further comprising:

identifying a source of the first packet;

wherein, step (c) includes a step of updating the multicast forwarding table so as to cross-reference the identified source to the identified multicast group.

17. The method as recited in claim 14, further comprising:

identifying the one I/O interface on which the first packet arrived; and

identifying a source of the first packet;

wherein, step (c) includes a step of updating the multicast forwarding data so as to cross-reference the identified multicast group with the identified I/O interface and the identified source.

18. The method as recited in claim 14, wherein:

when the first packet is not a multicast control packet:

determining whether the first packet is a message packet having a multicast group as a destination; and

when the first packet is determined to be a multicast message packet:

identifying a destination multicast group from the multicast message packet;

retrieving first data from the multicast forwarding data according to the identified destination multicast group, the retrieved first data including at least one cross-referenced I/O interface; and

sending the multicast message packet out each at least one cross-referenced I/O interface in the first data other than the identified I/O interface.

19. The method of claim **14**, wherein the network includes at least one host and at least one router that exchange multicast control packets to maintain multicast routing information, and wherein the first packet is a multicast control packet destined for a host or a router.

20. A method of conserving bandwidth in a communications network by reducing transmission of unneeded multicast messages, the method comprising:

monitoring, at a first location, multicast control packets destined for a location other than the first location;

from each monitored multicast control packet, identifying a multicast group, a source of the monitored multicast control packet and an interface on which the monitored multicast control packet arrived;

maintaining an association of the identified multicast group with the identified interface; and

for any multicast message received at the first location and destined for the identified multicast group, transmitting the received multicast message on any interface associated with the multicast group identified in the multicast message.

* * * * *

Exhibit 6

US006560236B1

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

(10) **Patent No.: US 6,560,236 B1**
(45) **Date of Patent: May 6, 2003**

(54) **VIRTUAL LANS**

(75) Inventors: **George Varghese**, Bradford, MA (US); **John Bassett**, Reading (GB); **Robert Eugene Thomas**, Hudson, MA (US); **Peter Higginson**, Herts (GB); **Graham Cobb**, Farnham (GB); **Barry A. Spinney**, Wayland, MA (US); **Robert Simcoe**, Westboro, MA (US)

(73) Assignee: **Enterasys Networks, Inc.**, Rochester, NH (US)

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

(21) Appl. No.: **09/411,773**

(22) Filed: **Oct. 4, 1999**

Related U.S. Application Data

(63) Continuation of application No. 08/731,905, filed on Oct. 22, 1996, now Pat. No. 5,963,556, which is a continuation of application No. 08/081,622, filed on Jun. 23, 1993, now abandoned.

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

(52) **U.S. Cl.** **370/401**; 370/432

(58) **Field of Search** 370/254, 259, 370/260, 261, 262, 263, 264, 265, 270, 351, 389, 390, 431, 432, 908, 911, 912, 312, 401; 379/156, 157, 158, 202.01, 221.15, 901

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,460,807	A	*	7/1984	Kerr et al.	179/18 BC
4,713,806	A		12/1987	Oberlander et al.	
4,893,307	A		1/1990	McKay et al.	
4,937,856	A	*	6/1990	Natarajan	379/158
5,311,593	A		5/1994	Carmi	
5,313,465	A		5/1994	Perlman et al.	
5,341,372	A		8/1994	Kirkham	
5,343,471	A		8/1994	Cardagnol	
5,361,256	A		11/1994	Doeringer et al.	
5,371,852	A		12/1994	Attanasio et al.	
5,379,296	A		1/1995	Johnson et al.	
5,430,726	A		7/1995	Moorwood et al.	
5,483,587	A	*	1/1996	Hogan et al.	379/202
5,638,434	A	*	6/1997	Gottlieb et al.	379/203
5,850,606	A	*	12/1998	Bedingfield, Sr. et al.	455/439
5,991,385	A	*	11/1999	Dunn et al.	379/202
6,163,692	A	*	12/2000	Chakrabarti et al.	455/416
6,219,412	B1	*	4/2001	Wellner et al.	379/202
6,236,644	B1	*	5/2001	Shuman et al.	370/261

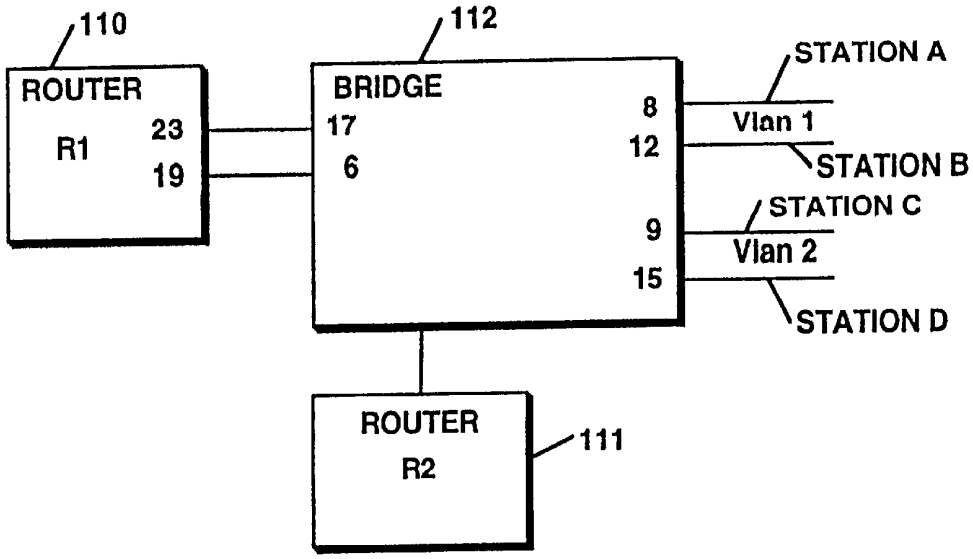
* cited by examiner

Primary Examiner—Ajit Patel
(74) *Attorney, Agent, or Firm*—Wolf, Greenfield & Sacks, P.C.

(57) **ABSTRACT**

A network device for interconnecting computer networks, the device including a bridge having a plurality of ports through which network communications pass to and from the bridge, the bridge also including a first interface enabling a user to partition the plurality of bridge ports into a plurality of groups, wherein each group represents a different virtual network, wherein the bridge treats all ports within a given group as part of the virtual network corresponding to that group and the bridge isolates the virtual networks from each other, whereby any communications received at a first port of the bridge are directly sent by the bridge to another bridge port only if the other bridge port and the first bridge port are part of the same group.

18 Claims, 6 Drawing Sheets



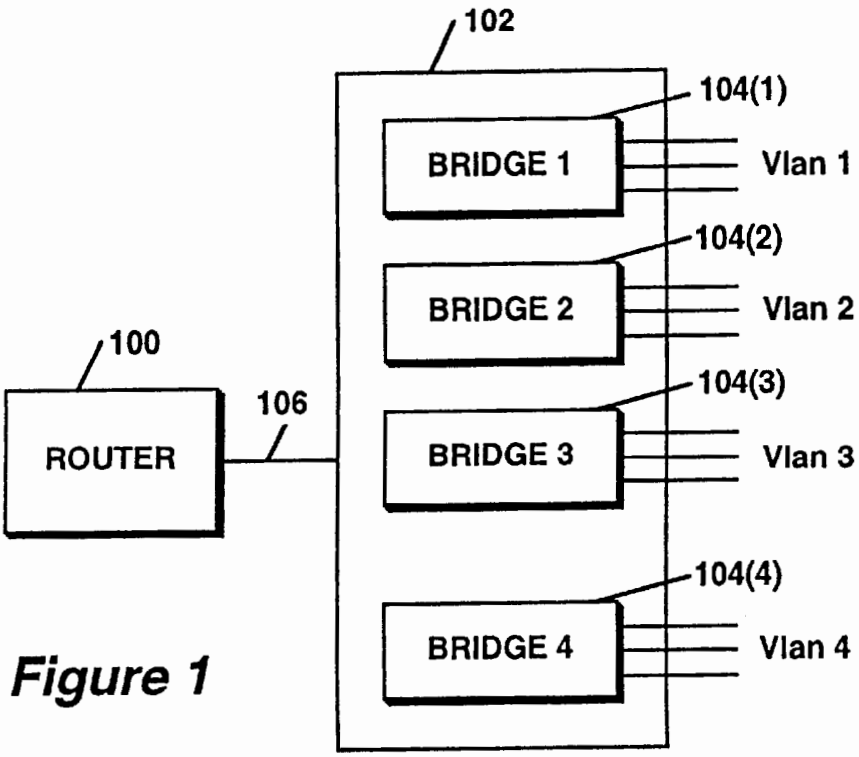


Figure 1

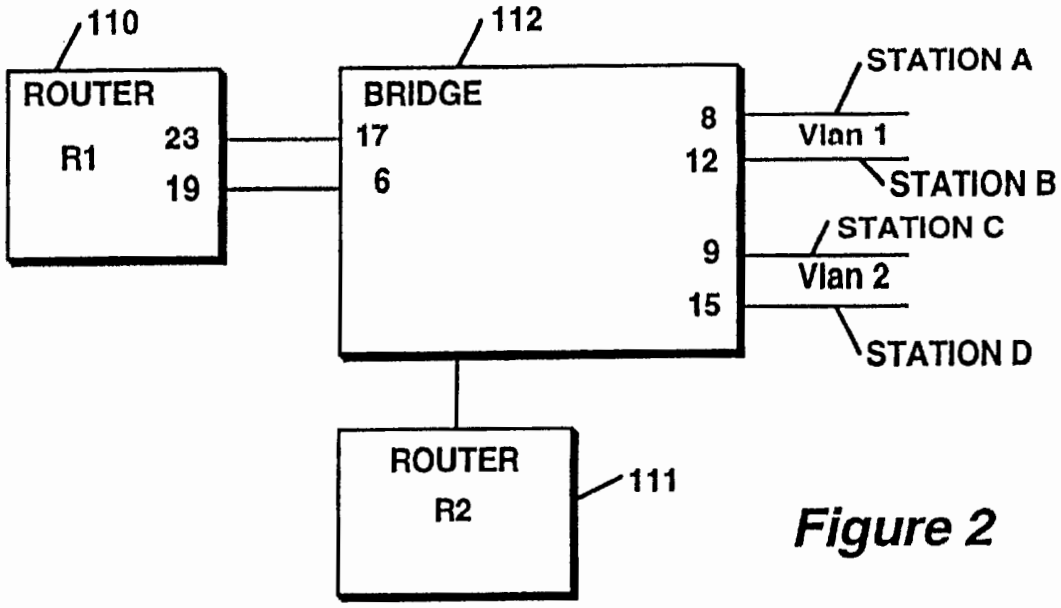


Figure 2

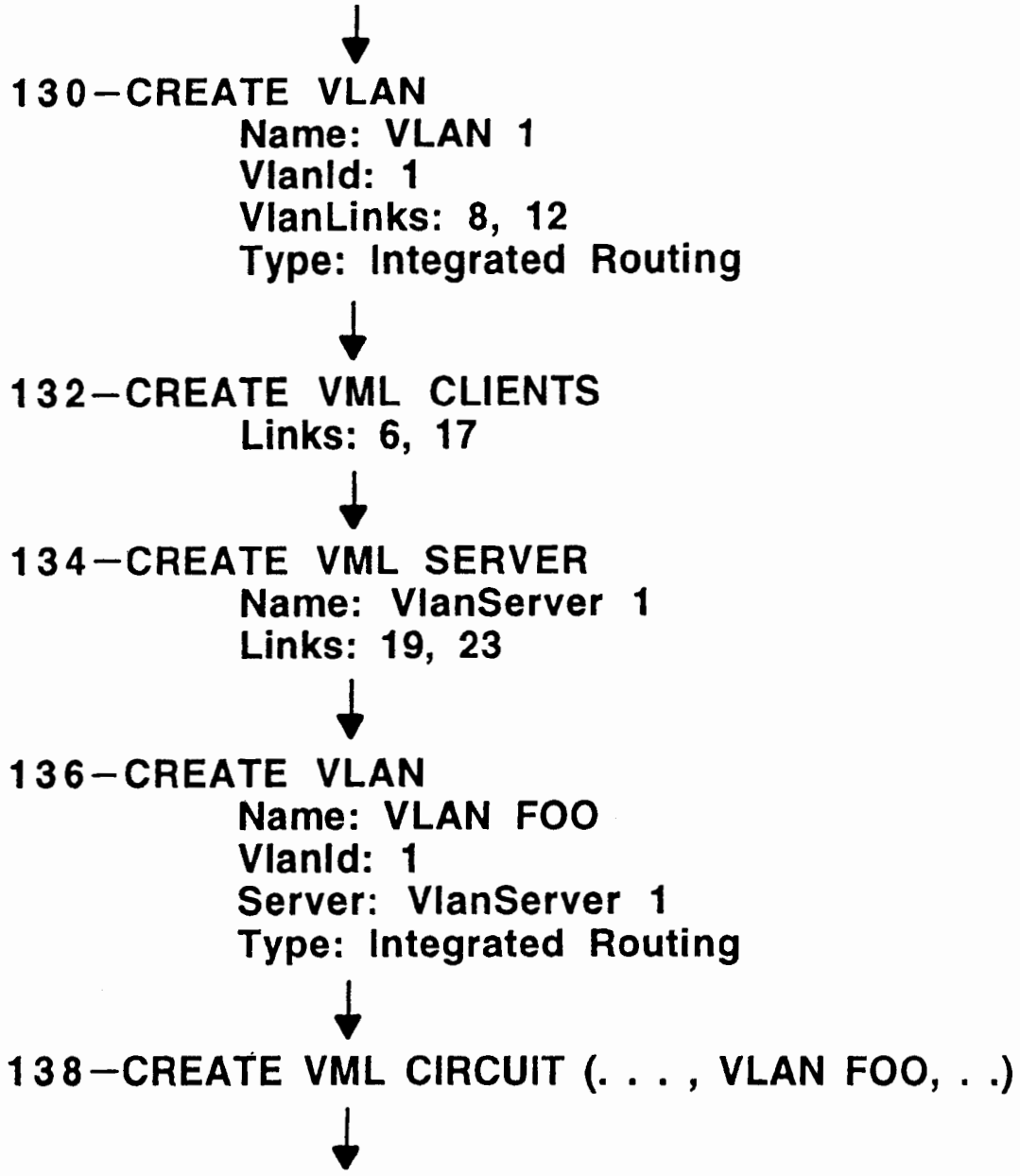

130—CREATE VLAN
Name: VLAN 1
VlanId: 1
VlanLinks: 8, 12
Type: Integrated Routing
132—CREATE VML CLIENTS
Links: 6, 17
134—CREATE VML SERVER
Name: VlanServer 1
Links: 19, 23
136—CREATE VLAN
Name: VLAN FOO
VlanId: 1
Server: VlanServer 1
Type: Integrated Routing
138—CREATE VML CIRCUIT (. . . , VLAN FOO, . .)

Figure 3

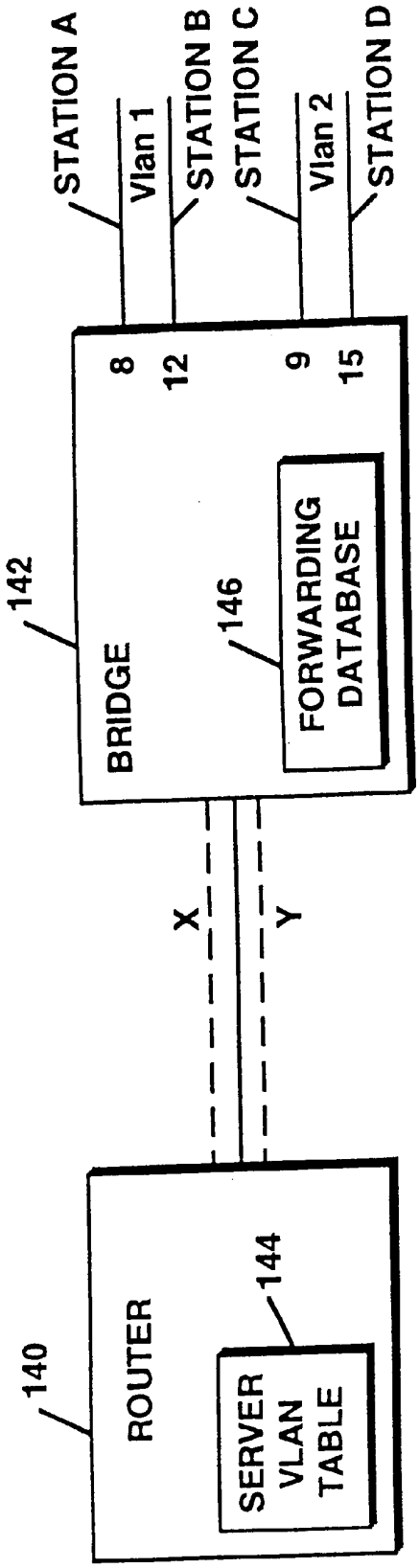

STATION A
Vlan 1
STATION B
STATION C
Vlan 2
STATION D
8
12
9
15
142
BRIDGE
146
FORWARDING DATABASE
X
Y
140
ROUTER
144
SERVER VLAN TABLE

Figure 4

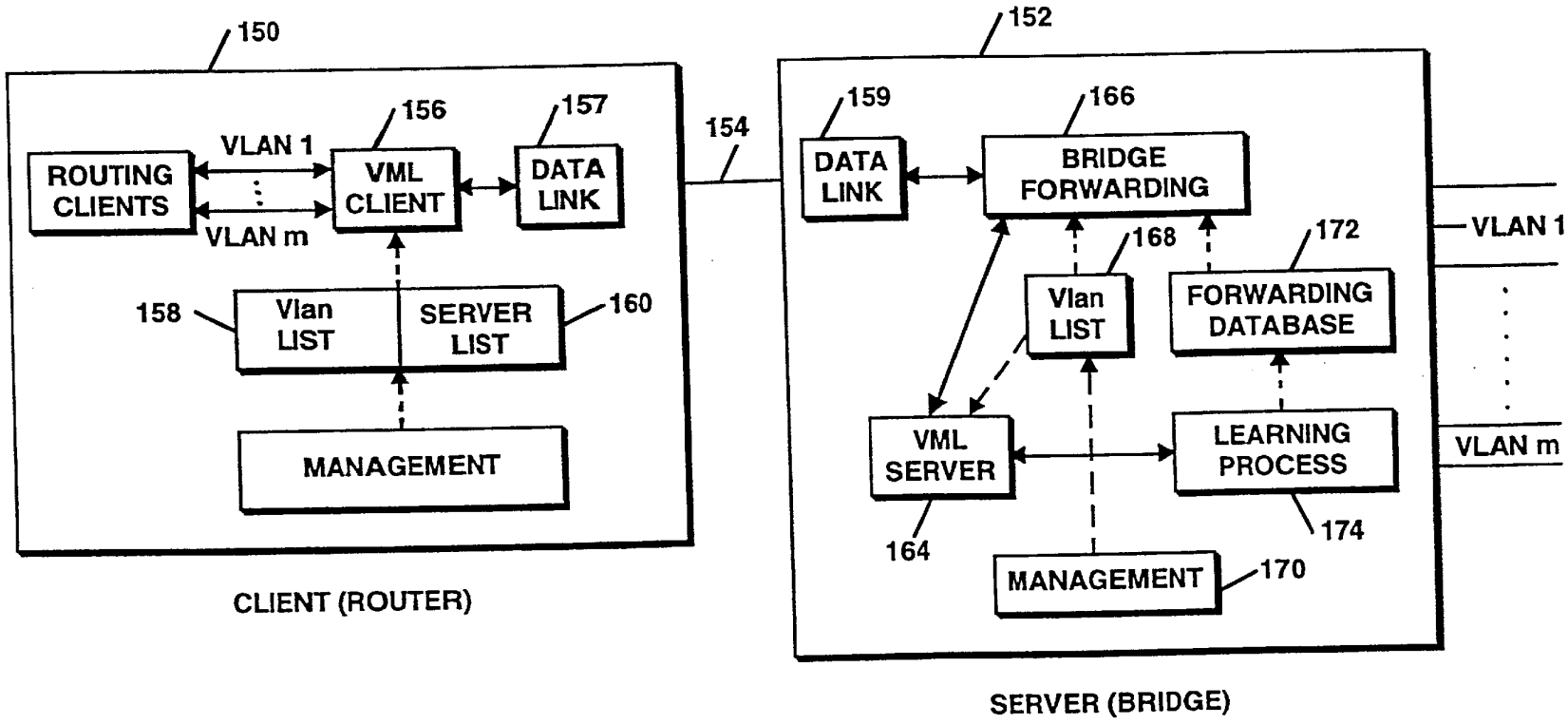

150
ROUTING CLIENTS
VLAN 1
VLAN m
156
VML CLIENT
157
DATA LINK
158
Vlan LIST
SERVER LIST
160
MANAGEMENT
CLIENT (ROUTER)
154
152
159
DATA LINK
166
BRIDGE FORWARDING
168
Vlan LIST
172
FORWARDING DATABASE
VML SERVER
164
LEARNING PROCESS
174
MANAGEMENT
170
VLAN 1
VLAN m
SERVER (BRIDGE)

Figure 5

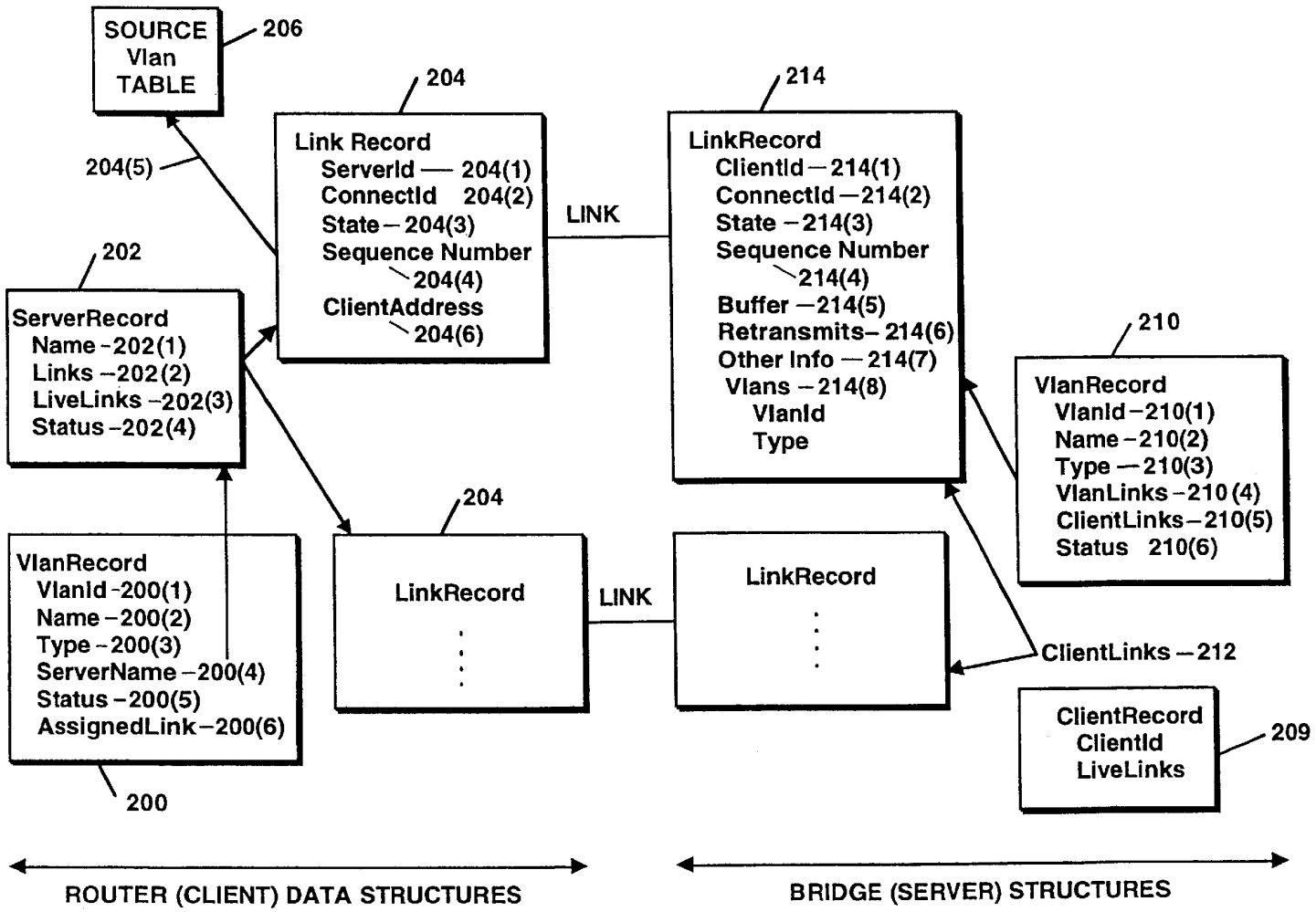
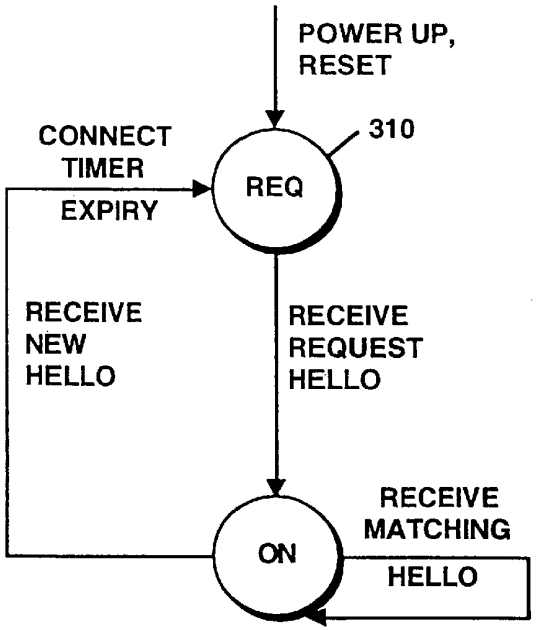
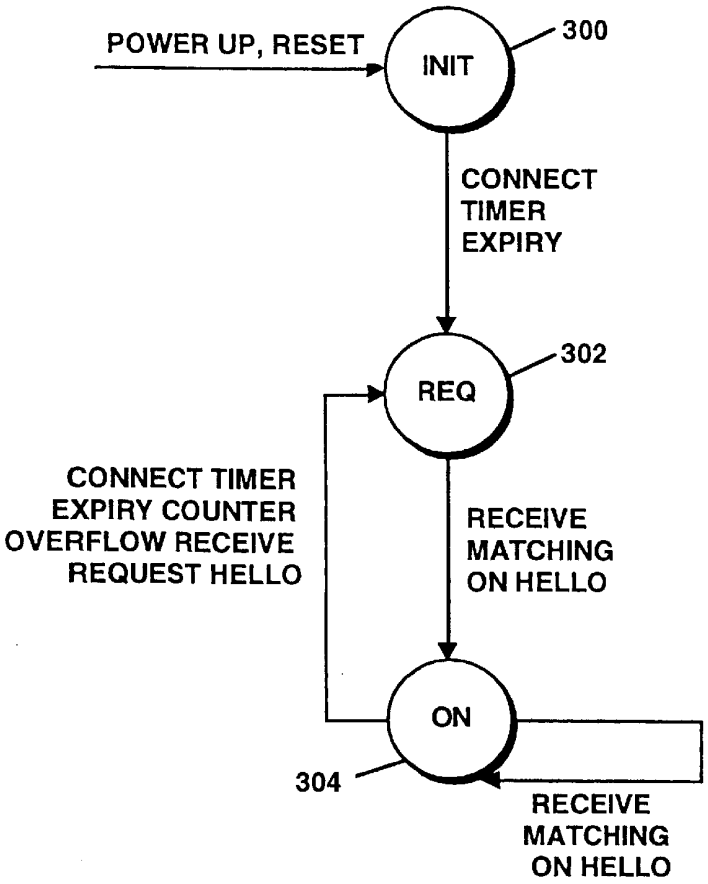

SOURCE Vlan TABLE
206
204
214
Link Record
ServerId — 204(1)
ConnectId 204(2)
State — 204(3)
Sequence Number
204(4)
ClientAddress
204(6)
204(5)
LINK
LinkRecord
ClientId — 214(1)
ConnectId — 214(2)
State — 214(3)
Sequence Number
214(4)
Buffer — 214(5)
Retransmits — 214(6)
Other Info — 214(7)
Vlans — 214(8)
VlanId
Type
202
ServerRecord
Name - 202(1)
Links — 202(2)
LiveLinks — 202(3)
Status - 202(4)
210
VlanRecord
VlanId — 210(1)
Name - 210(2)
Type — 210(3)
VlanLinks - 210 (4)
ClientLinks — 210(5)
Status 210(6)
204
VlanRecord
VlanId - 200(1)
Name - 200(2)
Type - 200(3)
ServerName — 200(4)
Status - 200(5)
AssignedLink — 200(6)
LinkRecord
LINK
LinkRecord
ClientLinks — 212
ClientRecord
ClientId
LiveLinks
209
200
ROUTER (CLIENT) DATA STRUCTURES
BRIDGE (SERVER) STRUCTURES

Figure 6



VML CLIENT CONNECTION
STATE MACHINE

Figure 7a



VML SERVER CONNECTION
STATE MACHINE

Figure 7b

VIRTUAL LANS

This application is a continuation application of prior application Ser. No. of prior application Ser. No. 08/731,905 filed Oct. 22, 1996 which issued as U.S. Pat. No. 5,963,556 on Oct. 5, 1999; which claimed priority and is a continuation to application Ser. No. 08/081,622, filed Jun. 23, 1993, now abandoned.

BACKGROUND OF THE INVENTION

The invention relates to Local Area Networks (LANs) and to bridges and routers that are used on such networks.

Bridges are devices that connect local area networks (LANs) together to form what are referred to as Extended LANs. Large Extended LANs have proven to be difficult to manage because of fault-isolation and addressing problems. The present invention enables a LAN manager to divide a large Extended LAN into smaller virtual LANs that have less overhead and are easier to manage. It further allows the LAN manager to interconnect the virtual LANs with a router.

The recent emergence of large multiport bridges, such as GIGAswitch from Digital Equipment Corporation, which can bridge up to 22 FDDI LANs, enable users to create a large extended LAN. That is, logically it appears that all stations that are bridged together by the switch are on a single LAN. This large configuration is reasonable if the bridge is at the periphery of the extended network and is responsible for bridging together a small number (say 100–250) of stations. However, there are two disadvantages if the bridge is used as the backbone of a large extended LAN. First, implementation and addressing limitations may limit the number of stations that can be present on a single Extended LAN. For example, it is well-known that broadcast traffic used in a LAN does not scale well as the number of LAN stations increases. The second problem is the lack of "firewalls" between the individual LANs that are bridged together by the bridge. An error on one LAN caused by a particular protocol failure can cause all other protocols on the LAN to fail. For example, if a set of stations on a particular LAN get stuck in a loop where they keep generating broadcast traffic, then the entire Extended LAN can fail. Thus some users choose to use a device called a router (as opposed to a bridge) to interconnect LANs.

There are several well-known differences between bridges and routers which make interconnecting LANs with routers more flexible and easier to manage. Routers allow users to construct extremely large and yet manageable networks. Some reasons for this are as follows. First, routers typically do not allow broadcast traffic; if they do, the broadcast traffic can be carefully controlled. By contrast, bridges must allow broadcast to allow LAN protocols to work correctly. Second, routers can be used to break up networks into a hierarchy of manageable subnetworks; bridges cannot. Third, routers have access to more information fields in messages than do bridges; this allows routers to have more discrimination in enforcing security and performance policies.

SUMMARY OF THE INVENTION

This invention provides a way of dividing a large Extended LAN up into multiple "Virtual" LANs (Vlans), which are interconnected by routers. The division is flexible and can be controlled by the manager. The division of the bridge ports into virtual LANs can also be done differently for different protocols.

In general, in one aspect, the invention features a a network device for interconnecting computer networks. The network device includes a bridge having a plurality of ports through which network communications pass to and from said bridge, and it also includes a first interface enabling a user to partition the plurality of bridge ports into a plurality of groups, wherein each group represents a different virtual network. The bridge treats all ports within a given group as part of the virtual network corresponding to that group and the bridge isolates the virtual networks from each other, whereby any communications received at a first bridge port are directly sent by the bridge to another bridge port only if the other bridge port and the first bridge port are part of the same group.

Preferred embodiments include the following features. The bridge also includes a second interface for enabling the user to designate one or more of the plurality of bridge ports as client ports, wherein the bridge sends to the client ports communications that are received from a station on one of said virtual networks and ultimately destined for a station on another of said virtual networks. The network device also includes a router connected to the bridge through the one or more client ports. The router includes a plurality of ports through which network communications pass to and from the router. The router includes an interface enabling the user to designate which one or more of the router ports are connected to the bridge. The router also includes a source table that contains a mapping of source addresses to the virtual networks, the source addresses representing locations of stations that are connected to the virtual networks and that send communications to the bridge. Upon receiving a unicast packet from the bridge, the router uses the source table to identify the virtual network from which the unicast packet came.

Alternatively, in preferred embodiments, the router is assigned a different router address for each of the virtual networks. The router includes a table assigning a different router address to the router for each of the virtual networks. When a unicast packet is sent from a first station on a first virtual network and destined for a second station on a second virtual network, it contains the router address corresponding to the first virtual network. The router identifies the virtual network from which the unicast packet originated by detecting the router address in the unicast packet and through the table determining that the router address corresponds to the first virtual network.

Also in preferred embodiments, the router includes a database identifying each of the virtual networks by a different network identifier. When the router sends to the bridge a multicast packet that is intended for one of the virtual networks, the router adds a network identifier to the multicast packet, the added network identifier being obtained from the database and identifying the virtual network for which the multicast packet is intended. The bridge, upon receipt of the multicast packet sent, removes the network identifier from the multicast packet and then forwards the modified multicast packet to the virtual network identified by the network identifier. The bridge also includes a database mapping the bridge ports to the virtual networks and the bridge uses that database to identify the bridge ports to which the bridge forwards the modified multicast packet. Upon receipt of a multicast packet from any of the virtual networks, the bridge adds source information to the received multicast packet and forwards the resulting multicast packet through one of the client ports to the router. The bridge uses the database to obtain the source information that is added to the multicast packet and it identifies the virtual network from which the multicast packet received.

Preferred embodiments also include the following additional features. The bridge includes a forwarding table which maps addresses of stations to bridge ports. Upon receipt at the bridge of a unicast packet sent by the router and having a destination address located on one of the virtual networks, the bridge determines from the forwarding table through which bridge port that destination address is reachable and then forwards the unicast packet through the identified bridge port. The router includes a memory storing a server record that identifies the bridge to the router, that identifies the one or more designated router ports, and that identifies which of the one or more designated router ports is operational. The router memory also stores a virtual network record for each of the virtual networks. Each of the virtual network records identifies the virtual network with which it is associated and it also identifies a particular one of the one or more designated router ports as the port through which the router sends communications to the virtual network associated with that virtual network record. The bridge also includes a memory storing a virtual network record for each of the virtual networks. Each of the virtual network records in bridge memory identifies the virtual network with which it is associated and it identifies a particular one of the one or more client ports as the client port through which the bridge sends communications to the virtual network associated with that virtual network record.

The invention enables the manager to reconfigure Vlans easily as the needs of the network changes. Reconfiguration of the network is done by setting parameters and not by redeploying cables or boxes. It is also possible to set up Vlans differently for different protocols, thus creating multiple logical networks from the same physical network.

The invention does not rely on any special features of particular implementations though some hardware support can improve efficiency. Thus, the invention can be used with any router and bridge; and it can also be retrofitted into existing routers and bridges, thus preserving user investment.

The bridge forwarding code for unicast packets is not affected by adding Vlan support; whereas the multicast code is increased only slightly to add and remove VlanIds. The router forwarding code for sending packets is only marginally impacted (to add VlanIds for multicast packets). The router forwarding code for receiving packets is only marginally impacted under one approach. Under an alternative approach, a source lookup must be added to the code path, but this can be done efficiently with simple hardware support of the kind used in bridges.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of a router and a bridge which implement virtual LANs;

FIG. **2** is a block diagram of the invention showing port identification numbers on the router and the bridge;

FIG. **3** shows the interfaces for setting up a virtual LAN in the configuration illustrated by FIG. **2**;

FIG. **4** shows a bridge/router configuration that is used to illustrate alternative methods of addressing;

FIG. **5** is a model of a virtual LAN multiplexing protocol;

FIG. **6** shows the data structures at the server and at the client; and

FIGS. **7***a* and **7***b* show the client and server state machines, respectively;

The appendices at the end of the specification include the following:

Appendix I contains a formal description of the data structures that are stored at a client;

Appendix II contains a formal description of the data structures that are stored at a server;

Appendix III lists the basic data types that are used;

Appendix IV presents a logical view of the protocol messaging formats;

Appendix V describes the timers and macros that are used for the transport protocol at the server;

Appendix VI describes the macros that are used to assign Vlans and addresses at the server;

Appendix VII describes the server protocol actions to send and receive hellos;

Appendix VIII describes the client macros for setting up Vlans and transport connections;

Appendix IX describes the client protocol code used to set up Vlans and transport connections;

Appendix X describes the server protocol actions to send updates and receive acks;

Appendix XI describes the code to receive updates and send acks at the client;

Appendix XII describes the server macros for forwarding packets to and from Vlans;

Appendix XIII describes server code for forwarding multicast and unknown destination packets to and from Vlans;

Appendix XIV describes the macros used by the client for forwarding packets to and from Vlans; and

Appendix XV describes the client protocol actions for forwarding packets to and from Vlans.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

In FIG. **1** a router **100** and a bridge **102** are used to create four virtual LANS (identified as Vlan**1** through Vlan**4**) from **12** individual LANS, each represented by a single line on the right of the figure. Router **100** is referred to herein as a VML client and bridge **102** is referred to as a VML server. Bridge **102** is a large switch, such as DEC's GIGAswitch, which has been modified to appear as four virtual bridges **104(1)** through **104(4)**. Virtual bridges **104(1)**–**104(4)** communicate with each other through router **100**. In the described embodiment, the packets sent to router **100** by virtual bridges **104(1)**–**104(4)** are multiplexed over a single line **106** that connects router **100** to bridge **102**.

Filters internal to bridge **102** are used to make the bridge appear as a plurality of virtual bridges. And the multiplexing of packet traffic over the single line connecting bridge **102** to router **100** is accomplished by an extra protocol between a VML client layer at router **100** and a VML server layer at bridge **102**. The following describes both the required bridge filters and the protocol between VML clients and servers in greater detail.

I. Setting up Vlans

The configuration in FIG. **2** of a router **110** and a bridge **112** connected by two links **114(1)** and **114(2)** will be used to illustrate the interfaces that are used to set up a virtual LAN. In that configuration two Vlans are shown, namely Vlan **1** and Vlan **2**. The interfaces used to set up these Vlans are shown in FIG. **3**.

A Vlan is set up using five steps. First, the user uses a management interface at bridge **112** to create Vlan **1** by declaring which bridge ports belong to this Vlan (i.e., ports

8 and 12 in the FIGURE) and giving the Vlan a name, which has purely local significance (step 130). During this first step, the manager also gives the Vlan a VlanId (which in this case is 1). The VlanId is used to coordinate Vlan set up at router 110 and bridge 112. A similar procedure is used to set up Vlan 2 at the bridge.

Also note that each Vlan is given a type which represents the protocols that this Vlan serves. In other words, bridge 112 can appear to be divided into different Vlans for different protocol types. This allows protocols that cannot be connected through a router to be set up so that such protocols "see" bridge 112 as a single LAN. However, any two Vlans with the same type cannot have a common bridge port.

In the next step, the manager specifies the bridge ports that are connected to VML clients (i.e., routers) (step 132). In the illustrated example, bridge ports 17 and 6 are connected to a client router. Note that the manager does not specify which client ports are connected to the same router. The protocol figures this out by receiving "hellos" (to be described later) from client routers.

After the clients have been defined, the next few steps occur at the router. The user creates a VML Server at router 110 by specifying which router ports are connected to the same bridge (in this case, ports 23 and 19) (step 134). If router 110 was connected to another bridge, then the manager would create another VML server for the second bridge. The server is also given a local name.

Next, the user creates Vlans at router 110 by specifying the local name of the server, the routing type, and the VlanId used to identify the Vlan at the server end (step 136). FIG. 3 shows the creation of a Vlan locally called "FOO" at the router and which corresponds to Vlan 1 at bridge 112. The correspondence is made because they both use a common VlanId of 1. The user creates a second Vlan at router 110 corresponding to Vlan 2 at bridge 112.

Finally, for each routing type that is supported, the user creates a circuit corresponding to each Vlan. Thus, for example, a circuit is created corresponding to Vlan 1 and a second circuit can be created corresponding to Vlan 2. The net result is that the router has circuits for each Vlan.

Creating Vlans at both bridge 112 and router 110 is necessary because current router interfaces require all router circuits to be declared in advance (although their status can change to reflect whether the circuit is up or down). Also, an alternative would have been to put all the details of creating a Vlan (i.e., which server, which VlanId etc.) in the circuit creation call at router 110. However, it seemed more desirable to provide a routing layer with a clean abstraction (i.e. a Vlan) that looks very much like a LAN. It also seemed desirable that the routing layer interfaces required to create a Vlan circuit be similar to the interfaces needed to create a LAN circuit. The details of mapping Vlans are present in the VML layer.

II. Multiplexing and Demultiplexing Vlans

When packets arrive at the router from the bridge, the router must be able to tell which Vlan the packet was sent on. Similarly when packets are sent by the router to the bridge on a specific Vlan, the bridge must be able to tell which Vlan the packet was sent on. These capabilities are provided through a mechanism by which information about a Vlan (specifically the VlanId described in the previous section) is embedded in a packet. First, the mechanism will be described and then the manner by which the mechanism is used to provide the capabilities will be described.

A. Encoding a VlanId Field in a Packet:

Consider a data packet P. The system provides a function that adds to P some information (e.g. the VlanId of the Vlan) that describes the Vlan on which P was sent. The system also provides a second function that removes the VlanId field from P.

There are two simple methods by which the VlanId can be added to packet P. The simplest method is to embed P in another packet Q and to add a specially created VlanId field to the header of packet Q. To remove the VlanId field, the system simply extracts P from Q. Another method is to use a redundant field in P. If there is some field in P that contains redundant information that can be derived from other fields in P, then that field can be used to encode the VlanId field. For example, there is a redundant field called the SSAP field in the Data Link Headers of most data packets on a LAN. This field is almost always equal to another field called the DSAP field. Thus, to add the VlanId, the SSAP can be set equal to the VlanId; and to remove the VlanId, the SSAP field can be set equal to the DSAP field.

The first method is more general but the second is more efficient as it does not require the addition of headers to the original packet P.

B. Distinguishing Packets Sent from Router to Bridge:

Referring to FIG. 2, consider a packet P sent from router 110 to bridge 112 on Vlan 1. Two cases must be distinguished, namely, P is either (1) a multicast packet or a unicast packet destined to another router or (2) a unicast packet other than one destined to another router. A multicast packet is defined as a message that is sent on a LAN to a group of stations. The destination address in a multicast packet is a group or multicast address. A unicast packet is a message sent on a LAN to a single station. The destination address in a unicast packet is an individual address.

If P is a multicast packet, P has a destination address that is a group address which identifies a set of stations. In this case, it is crucial that P be sent only to bridge ports corresponding to Vlan 1. Failure to do so can cause routing protocols to break because they use multicast packets to determine which stations are present on a LAN. Thus, multicast packet P sent by router 110 must carry some information so that the bridge can identify which Vlan packet P is to be sent on. Router 110 supplies this information by adding a VlanId field to multicast packets which it sends. The VlanId field identifies the Vlan, in this case Vlan 1.

In a previous section, two options were described for adding a VlanId field to a packet. Both options for adding a VlanId require changing the normal forwarding process of a bridge. However, most bridges process multicast packets in software; thus, the required changes to the forwarding process can easily be made in software.

When bridge 112 receives multicast packet P, it reads the VlanId field in P to find which Vlan P is to be sent on. It then sends P to only the bridge ports corresponding to Vlan 1. Thus in FIG. 2, P is only sent on ports 8 and 12. P is not sent on ports 9 and 15 as these correspond to Vlan 2. However, before bridge 112 sends P on ports 8 and 12, it removes the VlanId field because LAN stations may detect an error if they receive a packet with a VlanId field.

If P is a unicast packet destined for another router 111 (also desingated R2 in FIG. 2), then the VlanId field is added to P before it is sent to the router. Once again, although this is a non-standard packet format, the router is able to receive such packets with an encoded VlanId.

If P is a unicast packet, P has a destination address that is an individual address which identifies a single station.

Router **110** could add a VlanId field to P as was done for multicast packets. However, many bridges forward unicast packets in hardware. Thus, it is hard to add the changes required for VlanId processing without redesigning the hardware. Another solution is to send a unicast packet P from the router without a VlanId field. When the packet P arrives at the bridge there are two possibilities. If the Destination Address DA in packet P is known to the bridge, packet P is sent to the bridge port corresponding to DA. If the Destination Address DA in P is unknown to the bridge, the packet P is sent on all bridge ports. For example, if packet P was destined to a Station A on Vlan **1**, but the address of Station A has not been "learned" by bridge **112**, then P will be sent on all ports, including that of Vlan **2**. Since Station A is only on one bridge port, the other copies will be ignored. Thus, until bridge **112** learns of Station A, packets sent from router **110** to Station A will cause redundant copies to be sent on all ports. This is much the same as normal bridge operation.

C. Distinguishing Packets Sent from Bridge to Router:

Referring again to FIG. **2**, suppose Station A on Vlan **1** sends a packet P that is destined to router **110**. Bridge **112** forwards packet P to router **110**. When packet P arrives at router **110**, router **110** determines which Vlan the packet was sent on. Again, the two cases can be distinguished, one involving the handling of multicast packets and the other involving the handling of unicast packets. If the packet is a multicast packet, as noted above, bridge **112** adds a VlanId field before forwarding the packet. Thus, when router **110** receives the packet, it decodes the VlanId field to yield the Vlan the packet was sent on. If the packet is a unicast packet sent by a bridge, it will not carry a VlanId. Thus, a different approach is necessary. There are at least two different solutions, which shall be referred to as method **1** and method **2**.

According to method **1**, the router uses distinct source addresses for each Vlan. The router is assigned a unique source address for each Vlan it connects to. Thus, for example, referring to FIG. **4**, if a router **140** has two Vlans, Vlan **1** and Vlan **2**, the router is assigned an address X for Vlan **1** and an address Y for Vlan **2**. When router **140** sends any packet P on Vlan **1** to bridge **142**, it uses X as the source address (in the Data Link header of packet P). Similarly, when router **140** sends any packet Q on Vlan **2** to bridge **142** it uses Y as the source address. A station such Station A on Vlan **1** learns the address of router **140** from the source address of packets sent by router **140** to Station A. Thus, stations on Vlan **1**, like Station A and Station B, will learn the router's address as being X. Similarly, stations on Vlan **2**, like Station C and Station D, will learn the router's address as being Y. All unicast packets sent from Vlan **1** to router **140** will be sent to X; similarly all unicast packets sent from Vlan **2** to router **140** will be sent to Y. Router **140** is thus able to distinguish the Vlan on which a packet is sent by looking at the destination address. In the example, packets sent to X are for Vlan **1**, while packets sent to Y are for Vlan **2**.

Method **1** is elegant but has two drawbacks. First, some routing protocols insist that the router uses the same source address on all LANs that the router connects to, making this method inapplicable for such protocols. Second, this method requires that there be multiple addresses for each Vlan. This may be a problem for some implementations.

Method **2** avoids these drawbacks. Referring again to FIG. **4**, according to method **2** the router keeps a Source Vlan Table **144** that associates 48-bit source addresses with Vlans. Received packets are distinguished by finding the Vlan associated with the Source Address of a received packet. For example, the router's table maps the address of Station A to Vlan **1** and the address of Station C to Vlan **2**. Thus, any packet with source address of Station A that is received at the router is assumed to have been sent on Vlan **1**.

Source Vlan Table **144** at router **140** is updated by bridge **142** using the following mechanism. Bridge **142** eventually learns the bridge port corresponding to each source address and enters this information into its forwarding database **146**. Thus, bridge **142** will learn that Station A belongs to bridge port **8** and Station C belongs to bridge port **9**. Whenever there is a change to forwarding database **146**, e.g. either a new entry is learned or a "timed out" entry is deleted, bridge **142** sends this information to router **140** using a reliable transport protocol. Each update sent to router **140** also carries the mapping of ports to Vlans, i.e., bridge **142** also indicates that bridge ports **8** and **12** belong to Vlan **1**, while bridge ports **9** and **15** belong to Vlan **2**. On receiving such an update, router **140** has enough information to update its Source Vlan Table **144**.

Method **2** is general but it requires extra processing for look-up in the Source Vlan table and for updating this table. However, most routers today are brouters, i.e., they implement the bridge forwarding algorithm as well as the normal forwarding code. Since a lookup of the source address is part of the bridge forwarding algorithm, the router often has hardware support for this operation, which makes the operation quite inexpensive. In sum, Method **2** works for all routing protocols and can be efficient with a small amount of hardware support.

D. Other Functions of VML Protocol:

In the description thus far, only LANs have been connected to the bridge. However, there could also be wide area links connected to the bridge. The VML layer makes it appear that the wide-area link is directly connected to one particular router. In other words, if the router does not have an ATM or SONET interface but the bridge does, then the VML layer can provide a LAN "pipe" between the ATM line on the bridge and the router. Conceptually, this is no different from providing virtual LANs except that these lines are typically point-to-point wide area links running protocols like HDLC and SMDS (as opposed to a LAN which has multiple stations whose addresses are unknown). Thus the Source Vlan table required for this kind of "Vlan" is quite small and static, and hence can even be updated manually.

III. The Model of a Virtual LAN Multiplexing Protocol

A model implementation is shown in FIG. **5**. A client **150** (i.e., a router) and a server **152** (i.e., a bridge) are each connected to a link **154** through respective Data Link layers **157** and **159**. In the figure, the solid lines denote the flow of packets (data flow) and the dotted lines denote the major control flow. Thus, an arrow from a process to a database indicates that the process writes the database; an arrow from the database to the process indicates that the process reads the database.

A. Clients

Each client has a single Client Vlan Multiplexing Layer (VML Client Layer) **156** which is responsible for multiplexing Vlans on to physical links to servers. The multiplexing at the client is controlled by two data structures, namely, a ServerList **158** and a VlanList **160**, that are set up by management. Briefly, ServerList **158** contains an entry for every server the client is connected to, and lists the router links that are physically connected to the server. Vlanlist **160**

contains an entry for every Vlan that the client wishes to connect to and lists information like the server on which this Vlan belongs and the VlanId which helps identify the Vlan at the server.

Client VML layer **156** reads ServerList **158** and VlanList **160** and uses this and other state information to multiplex and demultiplex packets. Client VML Layer **156** offers the illusion of multiple Vlans to routing protocols. The routing protocols interface to the Vlans exactly as they do to LANS, i.e., they begin by opening a port, identifying which protocol types they wish to receive, and finally sending and receiving packets on the opened port.

A routing protocol can also (optionally) open a port on what is referred to as an "unknown" Vlan. Suppose a packet is received by Client VML Layer **156** with a protocol type specified by the routing protocol. Suppose also that Client VML layer **156** is unable to decide which Vlan the packet arrived on. Then, Client VML Layer **156** queues the packet on the unknown Vlan port. The routing protocol can then optionally decide to forward or discard the packet. Forwarding packets received on unknown Vlans can help data packets to be forwarded even during periods when Client VML Layer **156** is still learning the information needed to demultiplex packets. Opening a port to the "unknown" Vlan is just a way to model this mechanism.

B. Servers

At the server end, there is a corresponding VML Server Layer **164**. Unlike VML Client Layer **156** at client **150**, VML Server Layer **164** is only involved in setting up Vlans at the server and not in the actual forwarding of data packets. All packets received by Data Link **159** are handed to a Bridge Forwarding Layer **166**. Bridge Forwarding Layer **166** forwards unicast packets to known destinations much as they would be handled in a normal bridge; however, the handling of multicast and unicast packets to an unknown destination is quite different. Bridge Forwarding Layer **166** consults a VlanList **168** at server **152** which contains a record for every Vlan declared at the server. VlanList **168** is used to forward multicast packets received on a Vlan only to the bridge ports corresponding to that Vlan. VlanList **168** is written by management, represented in FIG. **5** by management interface **170**.

VML Server Layer **164** sends Hello messages on every link that is declared to be a link to a client/router. The hellos sent on a link **154** are used to set up a transport connection to the VML Client Layer at the other end of link **154**. If VML Client Layer **156** replies, a transport connection is set up. VML Server Layer **164** then begins to send updates reliably to VML Client Layer **156** on link **154**.

As indicated in FIG. **5**, Bridge Forwarding Layer **166** consults a Forwarding Database **172**, which models the standard forwarding database on a bridge. The learning process in the bridge is modelled by a Learning Process **174** which writes information to Forwarding Database **172**. There is also an interface between Learning Process **14** and VML Server **164**.

Learning Process **174** sends to all VML Clients all updates that it has used to update Bridge Forwarding Database **172**. When a new line to a VML Client comes up, VML Server **164** informs Learning Process **174**. Learning Process **174** then begins sending learning updates (corresponding to the current state of Forwarding Database **172**) to VML Server **164**. VML Server **164** packages this information and sends it to VML client **156**. Finally, VML Client **156** uses the updates to build a Source Vlan Table. Recall that the Source Vlan Table, described in connection with FIG. **4**, is used by VML Client Layer **156** to demultiplex received packets.

As Learning Process **174** gets new information it does not send a complete copy of the new database; instead it only sends incremental updates. Thus, a complete set of updates is only sent when a line to a VML Client first comes up. Later changes are sent as incremental updates. However, the use of incremental updates requires a reliable transport protocol between the server and client on each line. The transport protocol is responsible for retransmitting each update until an ACK is received from the client.

C. Relationship to Bridge Architecture

For the described embodiment, it is assumed that all VML servers are IEEE 802.1 compliant spanning tree bridges. Thus, there are two ways the Spanning Tree protocol can interact with the VML protocol. VML server **164** can implement a single spanning tree for all Vlans or a separate spanning tree for each Vlan.

In the described embodiment, every VML server **164** implements a single spanning protocol for all Vlans. In other words, each VML server implements the spanning tree protocol on all its links. Once the spanning tree has stabilized, the Vlan ports specified by management will define the breakup of the Extended LAN into Vlans. Similarly, VML server **164** builds a single learning database for the entire bridge and not a separate learning database for each Vlan. As in the bridge architecture, the station addresses are unique over the entire Extended LAN (as opposed to only requiring unique addresses for each Vlan).

IV. A More Detailed Protocol Specification

A. The Client Data Structures

The principle data structures stored in memory at both client and server are shown in FIG. **6**. Considering first the data structures at the client end, there is a VlanRecord **200** for each Vlan declared at the client and there is a ServerRecord **202** for each server that the client knows about. Each VlanRecord **200** points to the Server Record **202** corresponding to the server on which this Vlan belongs. Each ServerRecord **202** points to a set of physical link interfaces that connect the client to the server. Each such link interface has a LinkRecord **204** which contains variables required to implement a reliable transport protocol. The transport protocol is used to send information (from the server to the client) that maps source addresses to Vlans. This mapping information is stored in a SourceVlanTable **206** at the client end. There is one SourceVlanTable **206** per link at the client and it is pointed to by the corresponding LinkRecord **204**.

Note that one may use multiple links between the router and the bridge as described in the U.S. Patent Application entitled "SYSTEM FOR ACHIEVING SCALABLE ROUTER PERFORMANCE", by George Varghese, David R. Oran, and Robert E. Thomas, filed on an even date herewith, and which issued as U.S. Pat. No. 5,905,723 on May 18, 1999 and incorporated herein by reference. In that case, there would be multiple LinkRecords.

VlanRecords **200** are linked together in a VlanList. ServerRecords **202** are linked together in a ServerList. And LinkRecords **204** are stored in an array indexed by the link number.

Each of the three records will now be described in greater detail.

1. VlanRecord

Each VlanRecord **200** contains a VlanId **200(1)**, which identifies the corresponding Vlan at the server; a Name **200(2)**, which only has local significance and is set according to the manager's convenience; a type **200(3)**, which identifies the routing type of the Vlan; and a ServerName **200(4)**, which identifies the name of the server on which this Vlan belongs. These variables are set by management. Each

VlanRecord **200** also has two other variables, both of which are set by the protocol, namely, a Status variable **200(5)** and an As signedLink variable **200(6)**.

Status variable **200(5)** indicates if the Vlan is correctly set up and if not, gives an indication of the type of error. The three possible errors are TypeMismatch, if the Vlan type at the client does not match the Vlan type of the corresponding Vlan at the server; IdMismatch, if there is no Vlan at the server with a VlanId equal to that declared at the client end; and ServerFailure, if none of the links to the server are considered operational. AssignedLink variable **200(6)** describes the physical interface assigned to this Vlan at the client.

2. ServerRecord

ServerRecord **202** contains two variables that are set by management. The first is a Name variable **202(1)** that is a local name assigned by management to this server. A VlanRecord V is made to point to a ServerRecord S by setting V.ServerName=S.Name. The second variable set by management is Links **202(2)**, a variable that identifies the set of physical interfaces that connect the client to this particular server. FIG. **6** shows two links between the client and the server. Thus, ServerRecord **202** at the client points to the two corresponding link records.

ServerRecord **202** also contains two other variables that are set by the protocol. First, there is a LiveLinks variable **202(3)** which is the subset of physical links declared in Links that are operational. The traffic from the client to the server must only be split among the set of LiveLinks. As links fail and recover LiveLinks is updated by the protocol. Finally, there is a State variable **202(4)** which describes error conditions. The two possible errors are MultipleServers, if any two links in Links are connected to two distinct servers, and Broadcast, if any link to Links is not a point-to-point link to the server.

3. LinkRecord

Each LinkRecord **204** contains variables required to implement a reliable transport connection with a corresponding link at the server end. Note that if there are multiple links between a client and a server, separate transport connections on each link are set up. ServerId **204(1)** is a 48-bit unique Id of the remote server and ConnectId **204(2)** is a 32 bit connection identifier. State **204(3)** is the state of the connection; the link is considered to be operational when the connection state is ON. SequenceNumber **204(4)** is the number of the last update successfully received from the server. Each LinkRecord contains a pointer to a SourceVlanTable **206** that is used to map source addresses of received unicast packets to the Vlan on which the packet was sent. Finally, a ClientAddresses variable **204(6)** is a list of addresses used by all VML clients on the same server, as reported by the server.

A more comprehensive description of the client data structures can be found in the pseudocode that is presented in Appendix I, attached hereto. The description uses the simple data types that are described Appendix III, also attached hereto.

B. Server Data Structures

As at the client, the server keeps a VlanRecord **210** for every Vlan declared at the server. The server also has a variable called ClientLinks **212** that lists the set of server links that are connected to clients. For each such link, the server keeps a LinkRecord **214** that has more fields than the corresponding LinkRecord **204** kept at the client. Finally, just as the client keeps a ServerRecord **202** for each server, the server keeps a ClientRecord **209** for each client it knows about. The ClientRecord information is useful in implementing hunt groups at the server. For further discussion of hunt groups as they apply to the bridge see U.S. patent application Ser. No. 07/542,856, entitled Fast Arbiter Having Easy Scaling for Large Numbers of Requesters, Large Numbers of Resource Types with Multiple Instances of Each Type, and Selectable Queuing Disciplines, incorporated herein by reference).

These data structures will now be described in greater detail.

1. VlanRecord:

VlanRecord **210** includes VlanId, Name, and Type fields **210(1)**, **210(2)**, and **210(3)** the contents of which are declared by management just as at the client end. However, at the server end, management also specifies VlanLinks **210(4)** which is the set of bridge links that belong to this Vlan. There are also two variables set by the protocol, namely, a ClientLinks variable **210(5)** and a Status variable **210(6)**. ClientLinks **210(5)** is a subset of the server variable ClientLinks that represents the client links to which this Vlan has been assigned. If there are multiple links between a server and a client, the client assigns each Vlan to exactly one operational link. Each client reports the link to which it assigns Vlan V to the server, and the server stores the set of assigned links to ClientLinks **210(5)**. Status variable **210(6)** is identical to Status variable **200(5)** found in a client VlanRecord **200**.

2. LinkRecord

LinkRecord **214** includes a ClientId variable **214(1)**, a ConnectId variable **214(2)**, a State variable **214(3)**, a SequenceNumber variable **214(4)**, all corresponding to previously described similar variables in the client LinkRecord. Server LinkRecord **214** also includes additional variables. First, there is a Buffer variable **214(5)** which is a buffer that stores the current update being transmitted on the link to the server. There is a Retransmits variable **214(6)** that counts the number of times the update stored in Buffer variable **214(5)** has been retransmitted to the client without receiving an acknowledgement. There is an Other Info field **214(7)** that contains various 48-bit addresses that are used by the client. Finally, there is a Vlans variable **214(8)** that identifies the Vlans that are assigned to the link.

3. ClientRecord

Just as in a ServerRecord at the client, each ClientRecord **209** stores a ClientId and a LiveLinks variable which represents the set of links to this client that are considered to be operational.

A formal description of the server data structures can be found in Appendix II, attached hereto.

C. Message Formats

There are four basic types of messages used by the VML protocol. First, servers send ServerHello messages and clients respond with ClientHello messages. These messages are used to set up the transport connections on links and also to coordinate Vlan information between client and server. Next, servers send Update messages to clients containing mapping information that is used by clients to update information in the SourceVlanTable of each link. Each Update message is numbered with a sequence number and clients respond to Updates by sending an ACK.

The relevant fields in each message are shown in Appendix IV, which presents only a logical view of the message formats. For example, in order to encode a variable length sequence (such as a set of VlanRecords), a length field is also needed.

For the most part, the fields of the messages shown in Appendix IV correspond to fields of similar names in the link records at client and server. The relevant state variables

are copied into fields of the same name in the protocol message. Thus, all fields in the client hello (except the PhaseIV Address in the case of the DECnet Phase IV communication protocol) are copied from fields of the same name in the corresponding link record at the client. The PhaseIV Address is copied from a global client variable that represents the 48-bit address corresponding to the PhaseIV address of the client (i.e., Phase IV address prefixed by HI-ORD). If the client has no Phase IV address, this field is set to all 0's.

Similarly, all fields except ClientAddresses in the server hello are copied from fields of the same name in the corresponding link record at the client. The ClientAddresses field is copied from a global server variable that represents all the 48-bit addresses reported by clients.

Each update carries the Server Id, the connection identifier, a sequence number and the actual data. The ACK has the same fields except for the data field.

V. A Protocol for Controlling Vlans

A. Setting Up Transport Protocol Connections Using Hellos

The state machine used for the transport protocol is shown in FIGS. **7***a–b*. State machines attempt to set up a transport connection on each link between a client and a server. Once management declares a ClientLink at the server, the server creates a LinkRecord for the link. On creation and on power up or reset of the link, the LinkRecord is reset by setting the State of the link to INIT **300**. The INIT state causes a wait for a timeout period that is sufficiently long such that by the time the server link exits INIT state: (1) all old control messages sent by the server and the client on this link will have disappeared (any control messages received by the server on a link in INIT state are ignored); and (2) the client will have timed out all old state information it had.

Thus, the timer in INIT state (called ConnectTimer) is set to be large enough to "flush" all old messages and state information. On expiry of the ConnectTimer, the server sets the state of the link to REQ **302** (i.e., requesting a connection) and sends a server hello periodically to the client. All hellos sent by the server list the Vlan Id and type of all Vlans known to the server. All hellos (sent by either the client or server) carry the state of the sender; a Hello with state REQ is called a RequestHello; a Hello with state ON is called an OnHello.

If the client receives a RequestHello while in REQ state, the client transitions to an ON state **304** and sends back an OnHello. While the server periodically sends hellos in the REQ and ON state, the client only sends hellos in response to server hellos. The client is responsible for distributing the Vlans heard from a server among the multiple lines going to the server that are ON. It is also useful for deciding which links multicast traffic for a Vlan will flow on. The client distributes Vlans by setting the variable AssignedLink in a VlanRecord to point to the assigned link. A simple policy would be to distribute the Vlans in roughly equal fashion among all links to the server that are ON.

Having distributed the Vlans among links to the server, the client sends back a hello to the server on the link it received a hello. The client hello lists all Vlans assigned by the client to this link. Notice that the server lists all its Vlans in its hellos, while the client lists the Vlans assigned to the line on which the hello is sent.

Suppose an OnHello comes back on link L to the server. Suppose the hello is received while the server link record is in REQ state and the connection Id in the hello matches the server connectionId. Then the two way handshake to set up the connection is considered to be complete, and the server changes the state of the link to ON state **304**. Also if a Vlan V is mentioned in the hello sent by the client, then the server updates the VlanRecord for V to include L in its list of ClientLinks. The ClientLinks for a Vlan is a subset of the ClientLinks of the server. Basically, when a server has multiple links to a client, the server selects exactly one of these links for each Vlan based on hellos sent by the client. The selected link is used by the server to send multicast traffic for the Vlan to the client.

In normal operation, once both client and server have turned a link ON, the server periodically sends hellos to the client and the client responds with a hello. However, if the client does not hear a hello from the server for a timeout period, the client transitions the link to REQ state **310**. The default value of the timer in INIT state at the server is chosen to be three times as large as the client timeout.

Besides the normal operation, there are two other interesting cases. First, if the client receives a Hello from the server with a new connection Id or server Id (i.e., if the old server is disconnected and a new server plugged in) while in ON state, the client remains in ON state but essentially starts a new connection. If the client crashes and comes up, the client starts the link in REQ state. However, the client will not leave REQ state until the server sends an OnHello to the client and the client responds with a RequestHello. Receipt of the RequestHello causes the server to go into REQ state and restart the connection. This is important because when the client crashed it may have lost all previous updates; thus it must force the server to send it all updates by restarting the connection.

B. Mapping Client 48-bit Addresses

The hellos sent by the client also lists the 48-bit addresses used by the client on the link, including any address derived from the client address. When the server gets the hello it sets up all bridge forwarding Databases such that the address listed in the hello point to the link the hello was received on.

The VMLServer also builds up a list of all client addresses (using a variable ClientAddresses) that it reports back to clients in its server hellos. Note that if a client R sends a packet to another VML Client router S, client R then includes a VlanId in the packet. Client R can distinguish packets sent to other VML Clients by consulting a list of client addresses sent to R by the server.

C. Consistency Checking

Recall that the manager enters information at both client and server to set up Vlans. The code has a few consistency checks on this information.

Recall that the server sends a list of all its Vlans in its hellos with VlanId and Type. Consider a Vlan V declared at the client with VlanId=I. The client will not assign Vlan V to a line unless it finds that the server reports some Vlan with VlanId=I in its hello. Thus, if the manager incorrectly enters the VlanId field for a Vlan at the server and client, the client will not "bring up" the Vlan. Instead the State of Vlan V is reported as IdMismatch.

Suppose by accident, the manager connects a client to two different servers using links L**1** and L**2** but declares L**1** and L**2** to be part of one server record at the client. The client will detect this since it receives hellos with two different Server Ids on both links. The State of the corresponding server record is set to MultipleServers and all Vlans that belong to this server have their State changed to ServerFailure.

If the State of a Vlan is anything other than ON, the client will not send any traffic on this Vlan and will not assign this Vlan to any link.

D. Formal Code for Controlling Vlans at Server

The formal code used by the server to control Vlans is presented in Appendices V, VI, and VII. Appendix V shows

the timers, constants, and macros used by the server transport protocol on each link. The code describes the timers as constants.

When a client hello is received on a link, the information in the link record for that link may change and the macros shown in Appendix VI are used to update information about Vlans, Clients, and addresses. The first macro UpdateClientList keeps track of the client links associated with each distinct client (some clients may be connected to the server with multiple links); if hunt groups are implemented, this information is used to create a single hunt group for all the ON links connected to each client. UpdateVlanStatus is used to choose at most one assigned link per client for each Vlan; the assigned link is the link on which the client reports the Vlan. This information is gathered into a set of client links for each Vlan that is used to forward multicast traffic. (Exactly one copy of a multicast packet sent on a Vlan is sent to each client by sending the packet to the assigned link for that client.)

Finally, UpdateAddresses (in Appendix VI) updates the set of 48-bit addresses reported by clients. It also makes forwarding database entries for these addresses.

In the formal code for the server transport protocol (Appendix VII), when some action on a link L is executed, it typically begins by obtaining the LinkRecord by looking up LinkArray[L]. It is, of course, assumed that there is a check as to whether L is a ClientPort and if L is not, the routine is not executed.

The server code describes actions taken when a hello is received and transport timers expire. It follows the state transitions described earlier.

E. Formal Code for Controlling Vlans at Client

The client code to control Vlans is described in Appendices VIII and IX. Appendix VIII contains a set of useful macros that are called by the client transport protocol. InitializeConnection is called when a connection first begins at a client and initializes transport variables. UpdateServerRecord and UpdateVlanStatus are called whenever the client receives a new server hello.

UpdateServerRecord checks for server errors (such as being connected to two different servers on two links that were declared to be part of one server record) and updates the concerned server record. UpdateVlanStatus similarly checks for Vlan errors (i.e., a Vlan declared at the client does not have a matching Vlan at the server) and also distributes client Vlans among all the ON lines that connect the client and the server.

Finally, the main client code is described in Appendix IX. It has routines to process a received server hello and for handling transport timer expiry. It follows the state machine described earlier.

F. Sending Updates Reliably Over Transport Connections

This section describes how the forwarding tables at the server are sent reliably to the client using the transport connections set up on links.

As was previously described, when the server turns a link ON, the server initializes the SequenceNumber field (for the link) to 1. Similarly when the client starts a connection, the client initializes the SequenceNumber field to 0. The client also initializes its SourceVlanTable to be empty.

In normal operation, the server will send its forwarding table suitable encoded to the client on each line. The client will send ACKs backs separately on each line. This is redundant because if a client has multiple lines to the server it really only needs one copy of the Forwarding table. However, by sending multiple copies parallel processing is possible.

When a connection starts at the server, the server informs the Update Process using a routine called InformUpdateProcess together with a status variable that is set to "new". It is assumed that the Update Process keeps track of the outstanding updates on each client link. When the Update Process gets a status of new for a line, it begins to send the entire Forwarding Database to the client as a sequence of updates. However, the update process sends only one update at a time. Each update is numbered with the server sequence number; when the client receives the update, the client incrementally updates its SourceVlanTable using the update and sends back an ACK. When the ACK is received at the server, the server informs the Update Process using the routine InformUpdateProcess together with a status variable that is set to free. When this happens the Update Process sends the next outstanding update. If the forwarding database changes while the client is being loaded, the UpdateProcess must keep track of the outstanding updates for each client link.

The current update being sent to a client is stored in a variable called Buffer. If an ACK is not received before a RetransmitTimer expires, the update is retransmitted from Buffer. If more than a certain number of retransmissions takes place, the server starts a new connection by going into REQ state and incrementing the connection identifier.

The net result is that on each line, the client will build up a SourceVlanTable that maps source 48-bit addresses to Vlans.

1. Code for Sending Updates Reliably at the Server

The server code for sending updates reliably is described in Appendix X. It uses a macro that is an interface to the update process. It has routines to send a new update, to retransmit an update, and to receive an ACK from the server.

2. Code for Sending Updates Reliably at the Client

The client code for receiving updates reliably and sending updates is described in Appendix XI. It uses a macro that encapsulates the implementation specific method used to update the SourceVlanTable when a new update arrives. The code has only one routine, a routine that receives an update, checks whether it is a duplicate, and sends an ACK back to the server.

G. Forwarding Packets Between Vlans

1. Client Forwarding

Normally the interface to a Data Link is via a port. According to a simplified view of a port interface, a client opens a port and is given a descriptor (much like a Unix file descriptor). The client then enables a certain protocol specifier (includes information on SAPs, Protocol Types, multicast etc.) on the port. The specifier describes the kinds of packets the client wants to receive. Finally, clients can transmit packets to the port; these packets are then sent out on the link. Also received packets of the specified type are queued to the port.

The actual DNA Data Link specifications are slightly different; for instance, the client does not give the port a single protocol specifier but instead separately enables each SAP, protocol type, etc. Also, the interface to receive a packet is typically a polling interface. However, these are just details.

One object is to make a Vlan look just like a LAN to clients and a similar port interface is offered to Vlans except that the ports on a Vlan are called VlanPorts to avoid confusion. Thus a routing protocol (e.g. Phase V DECNET) begins by opening a virtual port on a specified Vlan. Then the routing protocol enables a single protocol specifier on the virtual port which describes all the kinds of packets that the routing protocol wishes to receive. Finally, the routing

protocol transmits to the VlanPort and received packets are queued to the VlanPort.

When a virtual port is opened and given a protocol specifier, the VML layer at the client attempts to open up corresponding ports on all the physical links associated with this Vlan. Thus each Vlan has a server and each server has a set of physical links. The Client VML Layer opens up Data Link ports on each such link and enables each such Data Link port with the specifier of the VlanPort. The client stores the mapping between VlanPorts and Data Link ports in the PortMappingList. This list consists of a record for each association between a VlanPort and a Data Link port. Thus each VlanPort has multiple records in the PortMappingList, one for each Data Link port it is associated with.

The routing protocol can choose to open a port to the unknown Vlan. In this case, the client opens associated physical ports on all server links.

When forwarding a packet sent on VlanPort VP on Vlan V, the client first picks a link L among all the "live" links associated with Vlan V. These are the links associated with V's server that are in ON state. Next, the client searches the PortMapping list to find the Data Link port LP associated with L. If the packet is multicast or is destined to another VML client, a VlanId field is added to the packet. Finally the packet is queued to Data link port LP.

When a packet is received on link L with protocol specifier S, the client attempts to find the Vlan V the packet was sent on by first looking for a VlanId field (if it exists) and then consulting the SourceVlanTable (if packet is unicast). If such a V is found, the client searches the PortMappingList to find the VlanPort VP associated with Vlan V and specifier S; the client then queues the packet to VP. If no such V is found, the client searches the PortMappingList to find the VlanPort VP associated with the "unknown" Vlan and specifier S. If such a port exists, the client queues the packet to VP.

2. Server Forwarding

The server forwarding algorithm is essentially the bridge forwarding algorithm except for small changes to the way a server forwards: a) multicast packets and b) packets sent to an unknown destination address.

The Vlan associated with a received multicast packet P is determined by the type of link P was received on. If P was received on a client link, the Vlan is determined from the VlanId field in P. If P was received on a client link, the Vlan is determined from the VlanId field in P. If P was received on a Vlan link L, the Vlan is determined from the Vlan that link L has been declared to be part of.

Multicast packets sent on a Vlan V are forwarded only to Vlan links associated with V and to client links that report Vlan V in their client hellos. Before a multicast packet is sent on a client link, a VlanId field is added; similarly when a multicast packet is received from a client link, the VlanId field is removed before the packet is sent to Vlan links.

Packets with unknown destination address that are received on client links are sent to all Vlan links that are turned on in the spanning tree. Packets with unknown destination address that are received on a Vlan link are sent only on the Vlan associated with that link. Packets with unknown destination address are never sent on client links. This is because client hellos list all addresses of clients; hence client destination addresses should be unknown only during the (hopefully brief) period when the protocol initializes.

3. Server Code for Forwarding

The server code for forwarding packets is formally described in Appendices XII and XIII. The main server forwarding routines are in Appendix XIII. The two main routines are MULTICAST_FORWARD (which describes how multicast packets are forwarded) and UNKNOWN_FORWARD (which describes how packets with unknown destination addresses are forwarded). Appendix XII describes the macros used by the main code in Appendix XIII. The macros are mostly used to find the links associated with Vlans and to add and remove the VlanId field in the packet.

4. Client Forwarding Protocol

The formal code for the client forwarding protocol is described in Appendices XIV and XV. The major routines in Appendix XV are the TRANSMIT routine (to transmit packets on a Vlan, possibly adding a VlanId to a packet) and the RECEIVE routine (to demultiplex a received Data Link packet using either a VlanId field or the source mapping table). Appendix XIV describes the macros used by the main code in Appendix XV. The macros are mostly used to search the PortMappingList.for mappings between virtual ports, link ports, and protocol specifiers.

The SourceLookup macro finds the Vlan associated with a source address. It uses two architectural constants, UnknownVlanId (VlanId of the unknown Vlan) and VMLRouterId (a second VlanId reserved to denote that this Source Address is a VML Router).

The RECEIVE macro finds the Vlan associated with a packet as follows:

If the packet is multicast, obtain Vlan from the VlanId field in packet.

If not multicast:

Look up the source address in SourceVlanTable.

If the result indicates the packet is from a VML Router then find the Vlan from the VlanId field in the packet.

Otherwise use the Vlan returned by the source lookup.

Other embodiments are within the following claims.

What is claimed is:

1. A method of operating a network bridge having a first plurality of ports through which network communications pass to and from the bridge, the method comprising:

assigning a group identifier to each port of the first plurality of ports, wherein all ports with a same assigned group identifier are in a same group;

receiving a communication on a first port of the bridge; and

if the first port is one of the first plurality of ports and the communication is a multicast packet having a multicast destination address, sending the communication out of the bridge on all other ports of the first plurality of ports having the same assigned group identifier as the first port.

2. The method of claim **1**, further comprising:

identifying a source of the communication received on the first port of the bridge.

3. The method of claim **2**, further comprising:

identifying a destination of the communication;

determining a group identifier assigned to the destination; and

if the group identifier assigned to the destination and the group identifier assigned to the source are different, sending the communication out a client port not within the first plurality of ports.

4. The method of claim 1, wherein the bridge includes a client port not included within the first plurality of ports and the communication is a multicast packet, the method comprising:

receiving the multicast packet on the client port;

identifying a group identifier within the multicast packet; and

sending the multicast packet out on those ports having the same group identifier as the group identifier within the received multicast packet.

5. The method of claim 4, wherein:

the group identifier is removed from the multicast packet before sending the multicast packet out from the bridge.

6. The method of claim 1, further comprising:

assigning a protocol type to each group identifier;

wherein, for each port of first plurality of ports, the protocol type identifies a communication protocol used by a station connected to the port.

7. The method of claim 6, wherein no two distinct group identifiers having a same protocol type are assigned to a same port.

8. The method of claim 1, wherein a port may have more than one group identifier assigned to it.

9. A method of operating a network bridge having a first plurality of ports through which network communications pass to and from the bridge, the method comprising:

assigning a group identifier to each port of the first plurality of ports, wherein all ports with a same assigned group identifier are in a same group;

receiving a communication on a first port of the bridge;

if the first port is one of the first plurality of ports, sending the communication out of the bridge on all other ports of the first plurality of ports having the same assigned group identifier as the first port;

identifying a source of the communication received on the first port of the bridge;

identifying a destination of the communication;

determining a group identifier assigned to the destination; and

if the group identifier assigned to the destination and the group identifier assigned to the source are different, sending the communication out a client port not included within the first plurality of ports and indicating the group identifier assigned to the first port within the communication sent out the client port.

10. The method of claim 9, wherein the step of indicating the group identifier comprises:

replacing a redundant field within the communication with the group identifier.

11. A method of operating a network bridge having a first plurality of ports through which network communications pass to and from the bridge, the method comprising:

assigning a group identifier to each port of the first plurality of ports, wherein all ports with a same assigned group identifier are in a same group;

connecting a router to a client port of the bridge not within the first plurality of ports;

identifying the ports on the router connected to the network bridge;

defining, on the router, a correspondence between the identified ports connected to the network bridge and each distinct group identifier;

receiving a communication on a first port of the bridge;

if the first port is one of the first plurality of ports, sending the communication out of the bridge on all other ports of the bridge having the same assigned group identifier as the first port;

identifying a source of the communication received on the first port of the bridge;

identifying a destination of the communication;

determining a group identifier assigned to the destination; and

if the group identifier assigned to the destination and the group identifier assigned to the source are different, sending the communication out the client port.

12. The method of claim 11, further comprising:

receiving a second communication from the bridge at the router; and

determining a group identifier of the second communication.

13. The method of claim 11, further comprising:

sending a second communication from the router to the bridge,

wherein the second communication comprises a first group identifier, and

wherein the second communication comprises a source address of the router assigned to the first group identifier.

14. The method of claim 11, further comprising:

maintaining, at the router, a record of a source address of each communication from the bridge and a respective group identifier.

15. A computer program product for use with a network device having a computer and a first plurality of ports on which network communications pass to and from the network device, wherein the network device includes a client port not within the first plurality of ports, and the computer program product comprises computer program instructions that when executed by the computer direct the computer to perform a method of directing the network communications, the method comprising:

assigning a group identifier to each port of the first plurality of ports, wherein all ports with a same assigned group identifier are in a same group;

receiving a communication on a first port of the network device;

if the first port is one of the first plurality of ports, sending the communication out of the network device on all other ports of the first plurality of ports having the same assigned group identifier as the first port;

receiving a multicast packet having a multicast destination address on the client port;

identifying a group identifier within the multicast packet; and

sending the multicast packet out on those ports having the same group identifier as the group identifier within the received multicast packet,

wherein the group identifier is removed from the multicast packet before sending the multicast packet out from the network device.

16. The computer program product of claim 15, wherein the method further comprises:

identifying a source of the communication received on the first port.

17. The computer program product of claim 16, wherein the method further comprises:

identifying a destination of the communication;

determining a group identifier assigned to the destination; and

if the group identifier assigned to the destination and a group identifier assigned to the source are different, sending the communication out a client port not within the first plurality of ports.

18. A computer program product for use with a network device having a computer and a first plurality of ports on which network communications pass to and from the network device, the computer program product comprising computer program instructions that when executed by the computer direct the computer to perform a method of directing the network communications, the method comprising:

assigning a group identifier to each port of the first plurality of ports, wherein all ports with a same assigned group identifier are in a same group;

receiving a communication on a first port of the network device;

if the first port is one of the first plurality of ports, sending the communication out of the network device on all of the ports of the first plurality of ports having the same assigned group identifier as the first port;

identifying a source of the communication received on the first port;

identifying a destination of the communication;

determining a group identifier assigned to the destination; and

if the group identifier assigns to the destination and the group identifier assigned to the source are different, sending the communication out a client port not within the first plurality of ports, indicating the group identifier of the first port within the communication sent out the client port and replacing a redundant field within the communication with the group identifier.

* * * * *

Exhibit 7

WordWeb Online

Dictionary and Thesaurus: Search

WordWeb

WordWeb Online

Windows Software

Free Download

For developers

Help Us Improve

Report an error

Missing word/sense

Links

Crossword Software

Crossword Puzzles

Add to Google

WordWeb Pro

Crossword Compiler

Look up words in any Windows program: download the FREE dictionary software.

Verb: prepend

1. (computing) prefix or add to the beginning

Derived forms: prepends, prepending, prepended

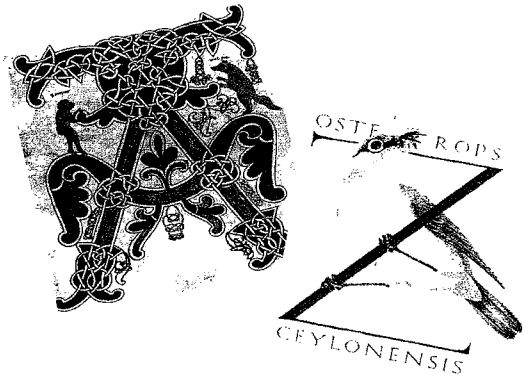
Nearest

preparative
preparatory
preparatory school
prepare
prepare for
prepared
preparedness
preparer
prepay
prepayment
prepend
prepense
prepensely
preponderance
preponderant
preponderantly
preponderate
preponderating
prepone
prepose
preposition

Copyright © 2005-2007

Exhibit 8

The
AMERICAN HERITAGE®
dictionary
of
THE ENGLISH LANGUAGE



fourth edition

Words are included in this Dictionary on the basis of their usage. Words that are known to have current trademark registrations are shown with an initial capital and are also identified as trademarks. No investigation has been made of common-law trademark rights in any word, because such investigation is impracticable. The inclusion of any word in this Dictionary is not, however, an expression of the Publisher's opinion as to whether or not it is subject to proprietary rights. Indeed, no definition in this Dictionary is to be regarded as affecting the validity of any trademark.

American Heritage® and the eagle logo are registered trademarks of Forbes Inc. Their use is pursuant to a license agreement with Forbes Inc.

Copyright © 2000 Houghton Mifflin Company. All rights reserved.

No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, or by any information storage or retrieval system without the prior written permission of Houghton Mifflin Company unless such copying is expressly permitted by federal copyright law. Address inquiries to Reference Permissions, Houghton Mifflin Company, 222 Berkeley Street, Boston, MA 02116.

Visit our Web site: www.hmco.com/trade.

Library of Congress Cataloging-in-Publication Data

The American Heritage dictionary of the English language.–4th ed.
p. cm.
ISBN 0-395-82517-2 (hardcover) — ISBN 0-618-08230-1 (hardcover with CD ROM)
1. English language–Dictionaries
PE1628 .A623 2000
423–dc21

00-025369

Manufactured in the United States of America

Case 1:05-cv-11298-DPW Document 131-9 Filed 08/17/2007 Page 4 of 4

pre•fix (prē′fĭks′) *tr.v.* **-fixed, -fix•ing, -fix•es** **1.** To put or attach before or in front of. **2.** (prē-fĭks′) To settle or arrange in advance. **3.** *Grammar* **a.** To add as a prefix. **b.** To add a prefix to. ❖ *n.* **1.** *Grammar* An affix, such as *dis-* in *disbelieve*, attached to the front of a word to produce a derivative word or an inflected form. **2.** A title placed before a person's name. [Middle English *prefixen*, from Old French *prefixer* : *pre-*, before (from Latin *prae-*; see PRE-) + *fixer*, to place (from Latin *fixus*, past participle of *figere*, to fasten; see **dhīg**^{w}- in Appendix I). N., from New Latin *praefixum*, from neuter sing. of Latin *praefixus*, past participle of *praefigere*, to fix in front : *prae-*, pre- + *figere*, to fasten.] —**pre′fix′al** *adj.* —**pre′fix′al•ly** *adv.* —**pre′fix•a′tion** (-fĭk-sā′shən), **pre•fix′ion** (-fĭk′shən) *n.*

pre•flight (prē′flīt′) *adj.* Preparing for or occurring before flight. ❖ *tr.v.* **-flight•ed, -flight•ing, -flights** To check (an aircraft) for airworthiness before flight. —**pre′flight′** *n.*

pre•form (prē′fôrm′) *tr.v.* **-formed, -form•ing, -forms** **1.** To shape or form beforehand. **2.** To determine the shape or form of beforehand. ❖ *n.* **1.** An object that has been subjected to preliminary, usually incomplete shaping or molding before undergoing complete or final processing. **2.** *Linguistics* A reconstructed ancestral word or form.

pre•for•ma•tion (prē′fôr-mā′shən) *n.* **1.** The act of shaping or forming in advance; prior formation. **2.** A theory popular in the 18th century that all parts of an organism exist completely formed in the germ cell and develop only by increasing in size.

pre•fron•tal (prē-frŭn′tl) *adj.* **1.** Of, relating to, or situated in the anterior part of the frontal lobe. **2.** Situated anterior to the frontal bone.

prefrontal lobotomy *n.* A lobotomy in which the white fibers that connect the thalamus to the prefrontal and frontal lobes of the brain are severed, formerly performed as a treatment for extremely violent behavior.

pre•gan•gli•on•ic (prē-găng′glē-ŏn′ĭk) *adj.* Of, relating to, or being the nerve fibers that supply a ganglion, especially a ganglion of the autonomic nervous system.

preg•na•ble (prĕg′nə-bəl) *adj.* Being such that attack, seizure, or capture is possible; vulnerable or assailable: *a pregnable fortress.* [Middle English *preignable*, *pregnabul*, from Old French *prenable*, *pregnauble*, from *prendre*, to grasp, from Latin *prehendere*, *prēndere*. See **ghend-** in Appendix I.] —**preg′na•bil′i•ty** *n.*

preg•nan•cy (prĕg′nən-sē) *n., pl.* **-cies** **1a.** The condition of being pregnant: *a test for pregnancy.* **b.** An instance of being pregnant: *Her second pregnancy was easy.* **c.** The period during which one is pregnant: *the first trimester of pregnancy.* **2.** The quality or condition of being rich in significance, import, or implication. **3.** Creativity; inventiveness.

preg•nant[1] (prĕg′nənt) *adj.* **1.** Carrying developing offspring within the body. **2a.** Weighty or significant; full of meaning: *a conversation occasionally punctuated by pregnant pauses.* **b.** Of great or potentially great import, implication, or moment: *"It was a politically pregnant time in Poland"* (New York). **3.** Filled or fraught; replete: *"This was, from the Party's point of view, both deplorable in itself and pregnant with danger for the future"* (Robert Conquest). **4.** Having a profusion of ideas; creative or inventive. **5.** Producing results; fruitful: *a pregnant decision.* [Middle English, from Old French, from Latin *praegnāns*, *praegnant-*, variant of *praegnās*. See **genə-** in Appendix I.] —**preg′nant•ly** *adv.*

preg•nant[2] (prĕg′nənt) *adj. Archaic* Convincing; cogent. Used of an argument or a proof. [Middle English, probably from Old French *preignant*, present participle of *prembre*, to press, from Latin *premere*. See **per-**[4] in Appendix I.]

pre•heat (prē-hēt′) *tr.v.* **-heat•ed, -heat•ing, -heats** To heat (an oven, for example) beforehand. —**pre•heat′er** *n.*

pre•hen•sile (prē-hĕn′səl, -sīl′) *adj.* **1.** Adapted for seizing, grasping, or holding, especially by wrapping around an object: *a monkey's prehensile tail.* **2.** Having keen intellect; insightful. **3.** Greedy; grasping. [French *préhensile*, from Latin *prehēnsus*, past participle of *prehendere*, to grasp. See **ghend-** in Appendix I.] —**pre′hen•sil′i•ty** (-sĭl′ĭ-tē) *n.*

pre•hen•sion (prē-hĕn′shən) *n.* **1.** The act of grasping or seizing. **2a.** Apprehension by the senses. **b.** Understanding. [Latin *prehēnsiō*, *prehēnsiōn-*, from *prehēnsus*, past participle of *prehendere*, to seize. See **ghend-** in Appendix I.]

pre•his•tor•ic (prē′hī-stôr′ĭk, -stŏr′-) also **pre•his•tor•i•cal** (-ĭ-kəl) *adj.* **1.** Of, relating to, or belonging to the era before recorded history. **2.** Of or relating to a language before it is first recorded in writing. —**pre′his•tor′i•cal•ly** *adv.*

pre•his•to•ry (prē-hĭs′tə-rē) *n., pl.* **-ries** **1.** History of humankind in the period before recorded history. **2.** The circumstances or developments leading up to or surrounding a situation, event, or development; background: *"[He] then told me the curious prehistory of his obsessive interest in the seduction theory"* (Janet Malcolm). —**pre′his•tor′i•an** (-hī-stôr′ē-ən, -stŏr′-) *n.*

pre•hom•i•nid (prē-hŏm′ə-nĭd) *n.* Any of several extinct primates regarded as an immediate ancestor of the hominids. ❖ *adj.* Of or relating to these extinct primates.

pre•ig•ni•tion (prē′ĭg-nĭsh′ən) *n.* The ignition of fuel in an internal-combustion engine before the spark passes through the fuel, resulting from a hot spot in the cylinder or from too great a compression ratio for the fuel.

pre•in•dus•tri•al (prē′ĭn-dŭs′trē-əl) *adj.* Of, relating to, or being a society or an economic system that is not or has not yet become industrialized.

pre•judge (prē-jŭj′) *tr.v.* **-judged, -judg•ing, -judg•es** To judge beforehand without possessing adequate evidence. —**pre•judg′er** *n.* —**pre•judg′ment, pre•judge′ment** *n.*

prej•u•dice (prĕj′ə-dĭs) *n.* **1a.** An adverse judgment or opinion formed beforehand or without knowledge or examination of the facts. **b.** A preconceived preference or idea. **2.** The act or state of holding unreasonable preconceived judgments or convictions. See synonyms at **predilection**. **3.** Irrational suspicion or hatred of a particular group, race, or religion. **4.** Detriment or injury caused to a person by the preconceived, unfavorable conviction of another or others. ❖ *tr.v.* **-diced, -dic•ing, -dic•es** **1.** To cause (someone) to judge prematurely and irrationally. See synonyms at **bias**. **2.** To affect injuriously or detrimentally by a judgment or an act. [Middle English, from Old French, from Latin *praeiūdicium* : *prae-*, pre- + *iūdicium*, judgment (from *iūdex*, *iūdic-*, judge; see **deik-** in Appendix I).]

prej•u•di•cial (prĕj′ə-dĭsh′əl) *adj.* **1.** Detrimental; injurious. **2.** Causing or tending to preconceived judgment or convictions: *Jurors were told not to read the newspapers to avoid being exposed to prejudicial publicity for the defendant.* —**prej′u•di′cial•ly** *adv.* —**prej′u•di′cial•ness** *n.*

prej•u•di•cious (prĕj′ə-dĭsh′əs) *adj.* Prejudicial. —**prej′u•di′cious•ly** *adv.*

prel•a•cy (prĕl′ə-sē) *n., pl.* **-cies** **1a.** The office or station of a prelate. **b.** Prelates considered as a group. Also called *prelature*. **2.** Church government administrated by prelates.

pre•lap•sar•i•an (prē′lăp-sâr′ē ən) *adj.* Of or relating to the period before the fall of Adam and Eve. [PRE- + Latin *lāpsus*, fall; see LAPSE + -ARIAN.]

prel•ate (prĕl′ĭt) *n.* A high-ranking member of the clergy, especially a bishop. [Middle English *prelat*, from Old French, from Medieval Latin *praelātus*, from Latin, past participle of *praeferre*, to carry before, to prefer : *prae-*, pre- + *lātus*, brought; see **telə-** in Appendix I.] —**pre•lat′ic** (prĭ-lăt′ĭk) *adj.*

prelate nul•li•us (nŏō-lē′əs) *n., pl.* **prelates nullius** *Roman Catholic Church* A prelate, usually a titular bishop, who has jurisdiction over a territory not in a diocese but subject directly to the Holy See. [PRELATE + New Latin *nullīus (dīvecēsis)*, of no (diocese) (from Latin, genitive sing. of *nullus*, none).]

prel•a•ture (prĕl′ə-chər, -chŏōr′) *n.* See **prelacy** (sense 1).

pre•launch (prē′lônch′, -lŏnch′) *adj.* Preparatory or preliminary to launch, especially of a spacecraft or missile.

pre•law (prē′lô′) *adj.* Of, relating to, or being the studies that prepare one for the study of law.

pre•lect (prĭ-lĕkt′) *intr.v.* **-lect•ed, -lect•ing, -lects** To lecture or discourse in public. [Latin *praelegere*, *praelect-* : *prae-*, pre- + *legere*, to read; see **leg-** in Appendix I.] —**pre•lec′tion** *n.* —**pre•lec′tor** *n.*

pre•li•ba•tion (prē′lī-bā′shən) *n.* A foretaste. [Latin *praelībātiō*, *praelībātiōn-*, from *praelībātus*, past participle of *praelībāre*, to taste beforehand : *prae-*, pre- + *lībāre*, pour out, to taste.]

pre•lim (prē′lĭm′, prĭ-lĭm′) *n. Informal* A preliminary: *teams that were eliminated in the prelims; studied hard for my prelims.*

pre•lim•i•nar•y (prĭ-lĭm′ə-nĕr′ē) *adj.* Prior to or preparing for the main matter, action, or business; introductory or prefatory. ❖ *n., pl.* **-ies** **1.** Something that precedes, prepares for, or introduces the main matter, action, or business. **2.** An academic test or examination that is preparatory to one that is longer, more complex, or more important. **3.** *Sports* A contest to determine the finalists in a competition. **4.** *Sports* An event that precedes the main event of a particular program, especially in boxing or wrestling. **5.** *Printing* The front matter of a book. Often used in the plural. [From New Latin *praelīmināris* : Latin *prae-*, pre- + Latin *līmen*, *limin-*, threshold.] —**pre•lim′i•nar′i•ly** (-nâr′ə-lē) *adv.*

Synonyms *preliminary, introductory, prefatory, preparatory* These adjectives mean going before and preparing the way for something else: *a preliminary investigation; introductory remarks; an author's prefatory notes; preparatory steps.*

pre•lit•er•ate (prē-lĭt′ər-ĭt) *adj.* Of, relating to, or being a culture not having a written language. ❖ *n.* A person belonging to such a culture.

Pre•log (prĕl′ôg′), **Vladimir** 1906–1998. Bosnian-born Swiss chemist. He shared a 1975 Nobel Prize for research on the structure of biological molecules.

prel•ude (prĕl′yōōd′, prā′lōōd′, prĕl′-) *n.* **1.** An introductory performance, event, or action preceding a more important one; a preliminary or preface. **2.** *Music* **a.** A piece or movement that serves as an introduction to another section or composition and establishes the key, such as one that precedes a fugue, opens a suite, or precedes a church service. **b.** A similar but independent composition for the piano. **c.** The overture to an oratorio, opera, or act of an opera. **d.** A short composition of the 15th and early 16th centuries written in a free style, usually for keyboard. ❖ *v.* **-ud•ed, -ud•ing, -udes** —*tr.* **1.** To serve as a prelude to. **2.** To introduce with or as if with a prelude. —*intr.* To serve as a prelude or introduction. [Medieval Latin *praelūdium*, from Latin *praelūdere*, to play beforehand : *prae-*, pre- + *lūdere*, to play; see **leid-** in Appendix I.] —**prel′ud′er** *n.* —**pre•lu′di•al** (prĭ-lōō′dē-əl) *adj.*

pre•lu•sion (prĭ-lōō′zhən) *n.* A prelude or introduction. [Latin *praelūsiō*, *praelūsiōn-*, from *praelūsus*, past participle of *praelūdere*, to play beforehand. See PRELUDE.]

pre•lu•sive (prĭ-lōō′sĭv) *adj.* Of or serving as a prelude; introductory. —**pre•lu′sive•ly** *adv.*

pre•ma•lig•nant (prē′mə-lĭg′nənt) *adj.* Precancerous.

pre•mar•i•tal (prē-măr′ĭ-tl) *adj.* Taking place or existing before marriage. —**pre•mar′i•tal•ly** *adv.*

pre•ma•ture (prē′mə-tyŏōr′, -tŏōr′, -chŏōr′) *adj.* **1.** Occurring, growing, or existing before the customary, correct, or assigned time: un-



prehensile
kinkajou hanging from a tree limb

Exhibit 9

Network Working Group
Request for Comments: 994

ANSI X3S3.3 86-80
ISO TC97/SC6/N 3998
March 1986

I S O
INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE DE NORMALISATION

ISO/TC 97/SC 6
TELECOMMUNICATIONS AND INFORMATION
EXCHANGE BETWEEN SYSTEMS
Secretariat: USA (ANSI)

Title: Final Text of DIS 8473, Protocol for Providing the Connectionless-mode Network Service

Source: DIS 8473 Editor

Contents

1 Scope and Field of Application 6

2 References 7

SECTION ONE. GENERAL 9

3 Definitions 9
3.1 Reference Model Definitions 9
3.2 Service Conventions Definitions 9
3.3 Network Layer Architecture Definitions 9
3.4 Network Layer Addressing Definitions 10
3.5 Additional Definitions 10

4 Symbols and Abbreviations 11
4.1 Data Units 11
4.2 Protocol Data Units 11
4.3 Protocol Data Unit Fields 11
4.4 Parameters 11
4.5 Miscellaneous 11

5 Overview of the Protocol 12
5.1 Internal Organization of the Network Layer 12
5.2 Subsets of the Protocol 12
5.3 Addresses and Titles 13
5.3.1 Addresses 13
5.3.2 Network-entity Titles 13
5.4 Service Provided by the Network Layer 14
5.5 Underlying Service Assumed by the Protocol 14
5.5.1 Subnetwork Points of Attachment 15
5.5.2 Subnetwork Quality of Service 15
5.5.3 Subnetwork User Data 16
5.5.4 Subnetwork Dependent Convergence Functions 16
5.6 Service Assumed from Local Environment 16

SECTION TWO. SPECIFICATION OF THE PROTOCOL 18

6 Protocol Functions 18
6.1 PDU Composition Function 18
6.2 PDU Decomposition Function 19
6.3 Header Format Analysis Function 19

6.4 PDU Lifetime Control Function 20
6.5 Route PDU Function 20
6.6 Forward PDU Function 21
6.7 Segmentation Function 21
6.8 Reassembly Function 22
6.9 Discard PDU Function 23
6.10 Error Reporting Function 24
6.10.1 Overview 24
6.10.2 Requirements 25
6.10.3 Processing of Error Reports 25
6.10.4 Relationship of Data PDU Options to Error Reports . . 26
6.11 PDU Header Error Detection 27
6.12 Padding Function 28
6.13 Security 28
6.14 Source Routing Function 28
6.15 Record Route Function 29
6.16 Quality of Service Maintenance Function 30
6.17 Priority Function 31
6.18 Congestion Notification Function 31
6.19 Classification of Functions 31

Structure and Encoding of PDUs 33
7.1 Structure 33
7.2 Fixed Part 34
7.2.1 General 34
7.2.2 Network Layer Protocol Identifier 34
7.2.3 Length Indicator 35
7.2.4 Version/Protocol Identifier Extension 35
7.2.5 PDU Lifetime 35
7.2.6 Flags 35
7.2.6.1 Segmentation Permitted 35
7.2.6.2 More Segments 35
7.2.6.3 Error Report 36
7.2.7 Type Code 36
7.2.8 PDU Segment Length 36
7.2.9 PDU Checksum 36
7.3 Address Part 37
7.3.1 General 37
7.3.1.1 Destination and Source Addresses 37
7.4 Segmentation Part 38
7.4.1 Data Unit Identifier 38
7.4.2 Segment Offset 38
7.4.3 PDU Total Length 39
7.5 Options Part 39
7.5.1 General 39
7.5.2 Padding 40
7.5.3 Security 40
7.5.3.1 Source Address Specific 41
7.5.3.2 Destination Address Specific 41
7.5.3.3 Globally Unique Security 41
7.5.4 Source Routing 41

7.5.5 Recording of Route 42

7.5.6 Quality of Service Maintenance 43
7.5.6.1 Source Address Specific 43
7.5.6.2 Destination Address Specific 43
7.5.6.3 Globally Unique QoS 43
7.5.7 Priority 44
7.6 Data Part 45
7.7 Data (DT) PDU 46
7.7.1 Structure 46
7.7.1.1 Fixed Part 47
7.7.1.2 Addresses 47
7.7.1.3 Segmentation 47
7.7.1.4 Options 47
7.7.1.5 Data 47
7.8 Inactive Network Layer Protocol 47
7.8.1 Network Layer Protocol Id 47
7.8.2 Data Field 47
7.9 Error Report PDU (ER) 48
7.9.1 Structure 48
7.9.1.1 Fixed Part 49
7.9.1.2 Addresses 49
7.9.1.3 Options 49
7.9.1.4 Reason for Discard 50
7.9.1.5 Error Report Data Field 51

8 Conformance 51
8.1 Provision of Functions for Conformance 51

List of Tables

1 Service Primitives for Underlying Service 14
2 Service Primitives for Underlying Service 14
3 Timer Primitives 14
4 Categorization of Protocol Functions 32
5 Valid PDU Types 36
6 Encoding of Option Parameters 39
7 Reason for Discard 50
8 Categorization of Functions 52

List of Figures

1 Interrelationship of Standards 6
2 PDU Structure 34
3 PDU Header -- Fixed Part 34
4 PDU Header -- Address Part 37
5 Address Parameters 38
6 PDU Header -- Segmentation Part 38
7 PDU Header -- Options Part 39
8 PDU Header -- Data Field 45

9 DT PDU 46
10 Inactive Network Layer Protocol 47

11 Error Report PDU 48

0 Introduction

This Protocol Standard is one of a set of International Standards

produced to facilitate the interconnection of open systems. The set of standards covers the services and protocols required to achieve such interconnection.

This Protocol Standard is positioned with respect to other related standards by the layers defined in the Reference Model for Open Systems Interconnection (ISO 7498). In particular, it is a protocol of the Network Layer. This Protocol may be used between network-entities in end systems or in Network Layer relay systems (or both). It provides the Connectionless-mode Network Service as defined in Addendum 1 to the Network Service Definition Covering Connectionless-mode Transmission (ISO 8348/AD1).

The interrelationship of these standards is illustrated in Figure 1 below:

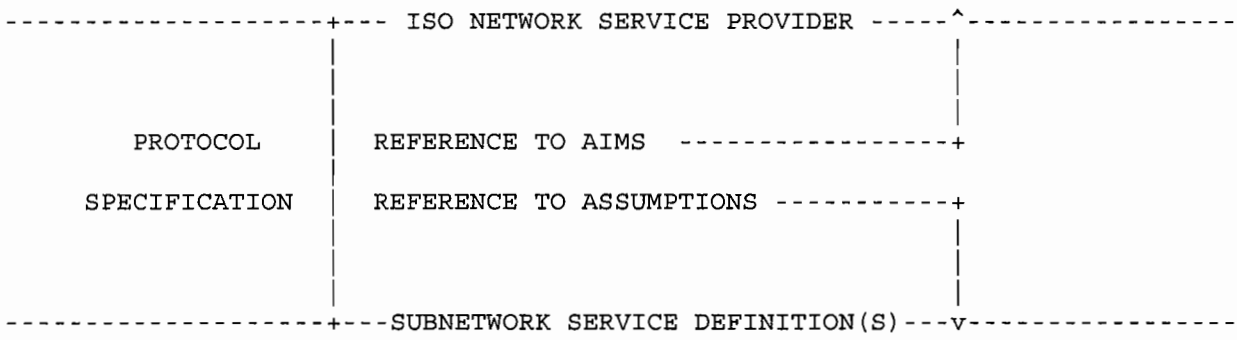


Figure 1: Interrelationship of Standards

1 Scope and Field of Application

This International Standard specifies a protocol which is used to provide the Connectionless-mode Network Service as described in Addendum 1 to the Network Service Definition Covering Connectionless-mode Transmission. The protocol relies upon the provision of an underlying connectionless-mode service by real subnetworks and/or data links. The underlying connectionless-mode service assumed by the protocol may be obtained either directly, from a connectionless-mode real subnetwork, or indirectly, through the operation of an appropriate Subnetwork Dependent Convergence Function (SNDCF) or Protocol (SNDCP) over a connection-mode real subnetwork as described in ISO 8648, Internal Organization of the Network Layer.

This Standard specifies:

a) procedures for the connectionless transmission of data and control information from one network-entity to a peer

network-entity;

b) the encoding of the protocol data units (PDUs) used for the transmission of data and control information, comprising a variable-length protocol header format;

c) procedures for the correct interpretation of protocol control information; and

d) the functional requirements for implementations claiming conformance to the Standard.

The procedures are defined in terms of:

a) the interactions among peer network-entities through the exchange of protocol data units;

b) the interactions between a network-entity and a Network Service user through the exchange of Network Service primitives; and

c) the interactions between a network-entity and an underlying service provider through the exchange of service primitives.

2 References

ISO 7498, Information Processing Systems --- Open Systems Interconnection --- Basic Reference Model

DIS 7498/AD1, Information Processing Systems --- Open Systems Interconnection --- Addendum to ISO 7498 Covering Connectionless-mode Transmission

ISO 8348, Information Processing Systems --- Telecommunications and Information Exchange between Systems --- Network Service Definition

ISO 8348/AD1, Information Processing Systems --- Telecommunications and Information Exchange between Systems --- Addendum to the Network Service Definition Covering Connectionless-mode Transmission

ISO 8348/AD2, Information Processing Systems --- Telecommunications and Information Exchange between Systems --- Addendum to the Network Service Definition Covering Network Layer Addressing*

DIS 8648, Information Processing Systems --- Telecommunications and Information Exchange between Systems --- Internal Organization of the Network Layer

ISO 8509, Technical Report --- OSI Service Conventions

ISO 9074, A Formal Description Technique based on an Extended State Transition Model

*At present, at the stage of Draft; publication anticipated in due course.

SECTION ONE. GENERAL

3 Definitions

3.1 Reference Model Definitions

This document makes use of the following concepts defined in ISO 7498:

(a) End system

(b) Network entity

(c) Network layer

(d) Network protocol

(e) Network protocol data unit

(f) Network relay

(g) Network service

(h) Network service access point

(i) Network service access point address

(j) Routing

(k) Service

(l) Service data unit

3.2 Service Conventions Definitions

This Protocol Standard makes use of the following terms from the OSI Service Conventions Technical Report (ISO TR 8509):

(a) Service provider

(b) Service user

3.3 Network Layer Architecture Definitions

This Protocol Standard makes use of the following terms from the Internal Organization of the Network Layer (ISO 8648):

(a) Intermediate system

(b) Relay system

(c) Subnetwork

3.4 Network Layer Addressing Definitions

This Protocol Standard makes use of the following terms from ISO 8348/AD2, Addendum to the Network Service Definition Covering Network Layer addressing:

(a) Network addressing domain

(b) Network protocol address information

(c) Subnetwork point of attachment

3.5 Additional Definitions

For the purposes of this Protocol Standard, the following definitions apply:

(a) derived PDU --- a protocol data unit whose fields are identical to those of an initial PDU, except that it carries only a segment of the user data from an N-UNITDATA request.

(b) initial PDU --- a protocol data unit carrying the whole of the userq data from an N-UNITDATA request.

(c) local matter --- a decision made by a system concerning its behavior in the Network Layer that is not prescribed or constrained by this Protocol Standard.

(d) network-entity title --- an identifier for a network-entity which has the same abstract syntax as an NSAP address, and which can be used to unambiguously identify a network-entity in an end or intermediate system.

(e) reassembly --- the act of regenerating an initial PDU from two or more derived PDUs.

(f) segment --- a distinct unit of data consisting of part or all of the user data provided in the N-UNITDATA request and delivered in the N-UNITDATA indication.

(g) segmentation --- the act of generating two or more derived PDUs from an initial or derived PDU. The derived PDUs together carry the entire user data of the initial or derived PDU from which they were generated.

Note:
It is possible that such an initial PDU will never actually be generated for a particular N-UNITDATA request, owing to the immediate application of segmentation.

4 Symbols and Abbreviations

4.1 Data Units

NSDU	Network Service Data Unit
PDU	Protocol Data Unit
SNSDU	Subnetwork Service Data Unit

4.2 Protocol Data Units

DT PDU	Data Protocol Data Unit
ER PDU	Error Report Protocol Data Unit

4.3 Protocol Data Unit Fields

CS	Checksum
DA	Destination Address
DAL	Destination Address Length
DUID	Data Unit Identifier
E/R	Error Report Flag
LI	Length Indicator
LT	Lifetime
MS	More Segments Flag
NLPID	Network Layer Protocol Identifier
SA	Source Address
SAL	Source Address Length
SL	Segment Length
SO	Segment Offset
SP	Segmentation Permitted Flag
TL	Total Lengt
TP	Type
V/P	Version/Protocol Identifier Extension

4.4 Parameters

DA	Destination Address
QOS	Quality of Service
SA	Source Address

4.5 Miscellaneous

CLNP	Connectionless-mode Network Protocol
NS	Network Service
NPAI	Network Protocol Address Information
NSAP	Network Service Access Point
SDU	Service Data Uni
SN	Subnetwork
SNDCF	Subnetwork Dependent Convergence Function
SNDCP	Subnetwork Dependent Convergence Protocol
SNICP	Subnetwork Independent Convergence Protocol
SNPA	Subnetwork Point of Attachment

5 Overview of the Protocol

5.1 Internal Organization of the Network Layer

The architectural organization of the Network Layer is described in a separate document, Internal Organization of the Network Layer (ISO 8648). ISO 8648 identifies and categorizes the way in which functions can be performed within the Network Layer by Network Layer protocols, thus providing a uniform framework for describing how protocols

operating either individually or cooperatively in the Network Layer can be used to provide the OSI Network Service. This protocol is designed to be used in the context of the internetworking protocol approach to the provision of the Connectionless-mode Network Service defined in that Standard.

This protocol is intended for use in the Subnetwork Independent Convergence Protocol (SNICP) role. A protocol which fulfills the SNICP role operates to construct the OSI Network Service over a defined set of underlying services, performing functions which are necessary to support the uniform appearance of the OSI Connectionless-mode Network Service over a homogeneous or heterogeneous set of interconnected subnetworks. This protocol is defined to accommodate variability where Subnetwork Dependent Convergence Protocols and/or Subnetwork Access Protocols do not provide all of the functions necessary to support the Connectionless-mode Network Service over all or part of the path from one NSAP to another.

As described in ISO 8648, a protocol at the Network Layer may fulfill different roles in different configurations. Although this protocol is designed particularly to be suitable for a SNICP role in the context of the internetworking protocol approach to the provision of the Connectionless-mode Network Service, it may also be used to fulfill other roles and may therefore be used in the context of other approaches to subnetwork interconnection.

The specification of this protocol begins with a definition of the underlying service which it assumes. This service is made available by the operation of other Network Layer protocols or through provision of the Data Link Service. The underlying service assumed by this protocol is described in Clause 5.5.

5.2 Subsets of the Protocol

Two proper subsets of the full protocol are defined which permit the use of known subnetwork characteristics and are therefore not subnetwork independent.

The Inactive Network Layer protocol subset is a null-function subset which can be used when it is known that the source and destination end-systems are connected by a single subnetwork, and when none of the functions performed by the full protocol is required to provide

the Connectionless-mode Network Service between any pair of end-systems.

The Non-segmenting protocol subset permits simplification of the header where it is known that the source and destination end-systems are connected by subnetworks whose service data unit sizes are greater than or equal to a known bound which is large enough so that segmentation is not required. This subset is selected by setting the Segmentation Permitted flag to zero.

5.3 Addresses and Titles

The following Clauses describe the addresses and titles used by this Protocol.

5.3.1 Addresses

The Source Address and Destination Address parameters referred to in Clause 7.3 of this International Standard are OSI Network Service Access Point Addresses. The syntax and semantics of an OSI Network Service Access Point Address are described in a separate document, ISO 8348/AD2, Addendum to the Network Service Definition Covering Network Layer Addressing.

The encoding used by this protocol to convey NSAP Addresses shall be the preferred binary encoding specified in ISO 8348/AD2; the entire NSAP address, taken as a whole, is represented explicitly as a string of binary octets. This string is conveyed in its entirety in the address fields described in Clause 7.3. The rules governing the generation of the preferred binary encoding are described in ISO 8348/AD2.

5.3.2 Network-entity Titles

A network-entity title is an identifier for a network-entity in an endsystem or intermediate-system. Network-entity titles are allocated from the same name space as NSAP addresses, and the determination of whether an address is an NSAP address or a network-entity title depends on the context in which the address is interpreted. The entries in the Source Routing and Recording of Route parameters defined in Clauses 7.5.4 and 7.5.5 are network-entity titles. The Source Address and Destination Address parameters in the Error Report PDU defined in Clause 7.9.1.2 are also network-entity titles.

The encoding used by this protocol to convey network-entity titles shall also be the preferred binary encoding; again, the entire network-entity title, taken as a whole, is represented explicitly as a string of binary octets. This string is conveyed in its entirety in the fields described in Clauses 7.5.4, 7.5.5, and 7.9.1.2.

5.4 Service Provided by the Network Layer

The service provided by this protocol is the Connectionless-mode Network Service described in ISO 8348/AD1, Addendum to the Network Service Definition Covering Connectionless-mode Transmission. The Network Service primitives provided are summarized in Table 1:

PRIMITIVES		PARAMETERS
N_UNITDATA	.Request	N_Source_Address,

	.Indication	N_Destination_Address, N_Quality_of_Service, N_Userdata

Table 1: Service Primitives for Underlying Service

The Addendum to the Network Service Definition Covering Connectionless-mode Transmission (ISO 8348/AD1) states that the maximum size of a connectionless-mode Network-service-data-unit (NSDU) is limited to 64512 octets.

5.5 Underlying Service Assumed by the Protocol

The underlying service required to support this protocol is defined by the following primitives:

PRIMITIVES		PARAMETERS
SN_UNITDATA	.Request	SN_Source_Address,
	.Indication	SN_Destination_Address, SN_Quality_of_Service, SN_Userdata

Table 2: Service Primitives for Underlying Service

Note:
These service primitives are used to describe the abstract interface which exists between the ISO 8473 protocol machine and an underlying real subnetwork or a Subnetwork Dependent Convergence Function which operates over a real subnetwork or real data link to provide the required underlying service.

5.5.1 Subnetwork Points of Attachment

The source and destination addresses specify the points of attachment to a public or private subnetwork(s) involved in the transmission. Subnetwork Point of Attachment addresses (SNPAs) are defined by each individual subnetwork authority.

The syntax and semantics of SNPAs are not defined in this Standard.

5.5.2 Subnetwork Quality of Service

Subnetwork Quality of Service describes aspects of an underlying

connectionless-mode service which are attributable solely to the underlying service.

Associated with each connectionless-mode transmission, certain measures of Quality of Service are requested when the primitive action is initiated. These requested measures (or parameter values and options) are based on a priori knowledge of the service(s) made available to it by the subnetwork. Knowledge of the nature and type of service available is typically obtained prior to an invocation of the underlying connectionless-mode service.

The Quality of Service parameters identified for the underlying connectionless-mode service may in some circumstances be directly derivable from or mappable onto those identified in the Connectionless-mode Network Service. The following parameters as defined in ISO 8348/AD1, Addendum to the Network Service Definition Covering Connectionlessmode Transmission, may be employed:

(a) transit delay;

(b) protection against unauthorized access;

(c) cost determinants;

(d) priority; and

(e) residual error probability.

Note:
For those subnetworks which do not inherently provide Quality of Service as a parameter when the primitive action is initiated, it is a local matter as to how the semantics of the service requested might be preserved. In particular, there may be instances in which the Quality of Service requested cannot be maintained. In such circumstances, an attempt shall be made to deliver the protocol data unit at whatever Quality of Service is available.

5.5.3 Subnetwork User Data

The SN-Userdata is an ordered multiple of octets, and is transferred transparently between the specified subnetwork points of attachment.

The underlying service assumed by the CLNP is required to support a service data unit size of at least 512 octets.

If the minimum service data unit sizes supported by all of the subnetworks involved in the transmission of a particular PDU are known to be large enough that segmentation is not required, then the Non-segmenting protocol subset may be used.

5.5.4 Subnetwork Dependent Convergence Functions

Subnetwork Dependent Convergence Functions may be performed to provide an underlying connectionless-mode service in the case where a real subnetwork does not inherently provide the connectionless-mode service assumed by the protocol. If a subnetwork inherently provides a connection-mode service, a Subnetwork Dependent Convergence Function provides a mapping into the required underlying service. Subnetwork Dependent Convergence Functions may also be required in those cases where functions assumed from the underlying service are not performed. In some cases, this may require the operation of an explicit protocol (i.e., a protocol involving explicit exchanges of protocol control information between peer network-entities) in the Subnetwork Dependent Convergence Protocol (SNDCP) role. However, there may also be cases where the functionality required to fulfill the SNDCP role consists simply of a set of rules for manipulating the underlying service.

5.6 Service Assumed from Local Environment

A timer service must be provided to allow the protocol entity to schedule events.

There are three primitives associated with the S-TIMER service:

1. the S--TIMER Request,
2. the S--TIMER Response, and
3. the S--TIMER Cancel.

The S--TIMER Request primitive indicates to the local environment that it should initiate a timer of the specified name and subscript and maintain it for the duration specified by the time parameter.

The S--TIMER Response primitive is initiated by the local environment to indicate that the delay requested by the corresponding S-TIMER Request primitive has elapsed.

The S--TIMER Cancel primitive is an indication to the local environment that the specified timer(s) should be canceled. If the subscript parameter is not specified, then all timers with the specified name are canceled; otherwise, the timer of the given name and subscript is cancelled. If no timers correspond to the parameters specified, the local environment takes no action.

The parameters of the S--TIMER service primitives are specified in Table 3.

PRIMITIVES		PARAMETERS
S--TIMER	.Request	S-Time,

		S-Name,
		S-Subscript
	.Response	S-Name,
		S-Subscript

Table 3: Timer Primitives

The time parameter indicates the time duration of the specified timer. An identifiying label is associated with a timer by means of the name parameter. The subscript parameter specifies a value to distinguish timers with the same name. The name and subscript taken together constitute a unique reference to the timer.

Timers used in association with a specific protocol funtion are defined under that protocol function.

Note:

This International Standard does not define specific values for the timers. Any derivations described in this Standard are not mandatory. Timer values should be chosen so that the requested Quality of Service can be provided, given the known characteristics of the underlying service.

SECTION TWO. SPECIFICATION OF THE PROTOCOL

6 Protocol Functions

This Clause describes the functions performed as part of the Protocol.

Not all of the functions must be performed by every implementation. Clause 6.17 specifies which functions may be omitted, and the correct behavior when requested functions are not implemented.

6.1 PDU Composition Function

This function is responsible for the construction of a protocol data

unit according to the rules governing the encoding of PDUs given in Clause 7. Protocol Control Information required for delivering the data unit to its destination is determined from current state and local information and from the parameters associated with the N-UNITDATA Request.

Network Protocol Address Information (NPAI) for the Source Address and Destination Address fields of the PDU header is derived from the NS-Source-Address and NS-Destination-Address parameters. The NS-Destination-Address and NS-Quality-of-Service parameters, together with current state and local information, are used to determine which optional functions are to be selected. User data passed from the Network Service User (NS-Userdata) forms the Data field of the protocol data unit.

During the composition of the protocol data unit, a Data Unit Identifier is assigned to distinguish this request to transmit NS-Userdata to a particular destination NS User from other such requests. The originator of the PDU must choose the Data Unit Identifier so that it remains unique (for this Source and Destination address pair) for the maximum lifetime of the Initial PDU in the network; this rule applies for any PDUs derived from the Initial PDU as a result of the application of the Segmentation Function (see Clause 6.7). Derived PDUs are considered to correspond to the same Initial PDU, and hence the same N-UNITDATA Request, if they have the same Source Address, Destination Address, and Data Unit Identifier.

The Data Unit Identifier is also available for ancillary functions such as error reporting (see Clause 6.10).

The total length of the PDU in octets is determined by the originator and placed in the Total Length field of the PDU header. This field is not changed in any Derived PDU for the lifetime of the protocol data unit.

When the Non-segmenting protocol subset is employed, neither the Total Length field nor the Data Unit Identifier field is present. The rules governing the PDU composition function are modified in this case as follows. During the composition of the protocol data unit, the total length of the PDU in octets is determined by the originator and placed in the Segment Length field of the PDU header. This field is not changed for the lifetime of the PDU. No Data Unit Identification is provided.

6.2 PDU Decomposition Function

This function is responsible for removing the Protocol Control Information from the protocol data unit. During this process, information pertinent to the generation of the N-UNITDATA Indication is determined as follows. The NS-Source-Address and NS-Destination-Address parameters of the N-UNITDATA Indication are recovered from the NPAI

in the Source and Destination Address fields of the PDU header. The data field of the PDU received is reserved until all segments of the original service data unit have been received; collectively, these form the NS-Userdata parameter of the N-UNITDATA Indication. Information relating to the Quality of Service provided during the transmission of the PDU is determined from the Quality of Service and other information contained in the Options Part of the PDU header. This information constitutes the NS-Quality-of-Service parameter of the N-UNITDATA Indication.

6.3 Header Format Analysis Function

This function determines whether the full protocol described in this Standard is employed, or one of the defined proper subsets thereof. If the protocol data unit has a Network Layer Protocol Identifier indicating that this is a standard version of the Protocol, this function determines whether a received PDU has reached its destination, using the Destination Address provided in the PDU. If the Destination Address provided in the PDU identifies an NSAP served by this network-entity, then the PDU has reached its destination; if not, it must be forwarded.

If the protocol data unit has a Network Layer Protocol Identifier indicating that the Inactive Network Layer Protocol subset is in use, then no further analysis of the PDU header is required. The network-entity in this case determines that either the Subnetwork Point of Attachment address encoded as network protocol address information in the supporting subnetwork protocol corresponds directly to an NSAP address serviced by this network-entity or that an error has occurred. If the subnetwork protocol data unit has been delivered correctly, then the PDU may be decomposed according to the procedures described for that particular subnetwork protocol.

6.4 PDU Lifetime Control Function

This function is used to enforce the maximum PDU lifetime. It is closely associated with the Header Format Analysis function. This function determines whether a PDU received may be forwarded or whether its assigned lifetime has expired, in which case it must be discarded.

The operation of the PDU Lifetime Control function depends upon the Lifetime field in the PDU header. This field contains, at any time, the remaining lifetime of the PDU (represented in units of 500 milliseconds). The Lifetime of the Initial PDU is determined by the originating network-entity, and placed in the Lifetime field of the PDU. When the Segmentation function is applied to a PDU, the value of the Lifetime field of the Initial PDU is copied into all of the Derived PDUs.

The Lifetime of the PDU is decremented by every network-entity which processes the PDU. When a network-entity processes a PDU, it decrements the PDU Lifetime by at least one. The value of the PDU Lifetime field shall be decremented by more than one if the sum of:

1. the transit delay in the underlying service from which the PDU was received; and

2. the delay within the system processing the PDU

exceeds or is estimated to exceed 500 milliseconds. In this case, the lifetime field should be decremented by one for each additional 500 milliseconds of delay. The determination of delay need not be precise, but where a precise value cannot be ascertained, the value used shall be an overestimate, not an underestimate.

If the Lifetime field reaches a value of zero before the PDU is delivered to the destination, the PDU must be discarded. The Error Reporting function shall be invoked as described in Clause 6.10, Error Reporting Function, and may result in the generation of an Error Report PDU. It is a local matter whether the destination network-entity performs the Lifetime Control function.

6.5 Route PDU Function

This function determines the network-entity to which a protocol data unit should be forwarded and the underlying service that must be used to reach that network-entity, using the Destination Address and the total length of the PDU. Where segmentation is required, the Route PDU function further determines over which underlying service Derived PDUs/segments must be sent in order to reach that network-entity. The results of the Route PDU function are passed to the Forward PDU function (along with the PDU itself) for further processing. Selection of the underlying service that must be used to reach the "next" sys-

tem in the route is initially influenced by the NS-Quality-of- Service parameter of the N-UNITDATA Request, which specifies the QoS requested by the sending NS User. Whether this QoS is to be provided directly by the CLNP, through the selection of the Quality of Service Maintenance parameter and other optional parameters, or through the QoS facilities offered by each of the underlying services is determined prior to invocation of the Forward PDU function. Route selection by intermediate systems may subsequently be influenced by the values of the Quality of Service Maintenance parameter (if present), and other optional parameters (if present).

6.6 Forward PDU Function

This function issues an SN-UNITDATA Request primitive (see Clause 5.5), supplying the subnetwork or SNDCF identified by the Route PDU function with the protocol data unit as user data to be transmitted, the address information required by that subnetwork or SNDCF to identify the "next" system within the subnetwork-specific addressing

domain (this may be an intermediate-system or the destination end-system), and Quality of Service constraints (if any) to be considered in the processing of the user data.

When the PDU to be forwarded is longer than the maximum service data user size provided by the underlying service, the Segmentation function is applied (See Clause 6.7, which follows).

6.7 Segmentation Function

Segmentation is performed when the size of the protocol data unit is greater than the maximum service data unit size supported by the underlying service to be used to transmit the PDU.

Segmentation consists of composing two or more new PDUs (Derived PDUs) from the PDU received. The PDU received may be the Initial PDU, or it may be a Derived PDU. All of the header information from the PDU to be segmented, with the exception of the segment length and checksum fields of the fixed part, and the segment offset of the segmentation part, is duplicated in each Derived PDU, including all of the address part, the data unit identifier and total length of the segmentation part, and the options part (if present).

Note:

The rules for forwarding and segmentation guarantee that the header length is the same for all segments (Derived PDUs) of the Initial PDU, and is the same as the header length of the Initial PDU. The size of a PDU header will not change due to operation of any protocol function.

The user data encapsulated within the PDU received are divided such that the Derived PDUs satisfy the size requirements of the user data parameter field of the primitive used to access the underlying ser-

vice.

Derived PDUs are identified as being from the same Initial PDU by means of

(a) the source address,

(b) the destination address, and

(c) the data unit identifier.

Segmentation shall not result in the generation of a Derived PDU containing less than eight (8) octets of user data.

The following fields of the PDU header are used in conjunction with the Segmentation function:

(a) Segment Offset --- identifies, with respect to the start of the Initial PDU, the octet at which the segment begins;

(b) Segment Length --- specifies the number of octets in the Derived PDU, including both header and data;

(c) More Segments Flag --- is set to one if this Derived PDU does not contain, as its final octet of user data, the final octet of the Initial PDU; and

(d) Total Length --- specifies the entire length of the Initial PDU, including both header and data.

Derived PDUs may be further segmented without constraining the routing of the individual Derived PDUs. The Segmentation Permitted flag is set to one to indicate that segmentation is permitted. If the Initial PDU is not to be segmented at any point during its lifetime in the network, the flag is set to zero by the source network-entity. The setting of the Segmentation Permitted flag cannot be changed by any other network-entity for the lifetime of the Initial PDU and any Derived PDUs.

6.8 Reassembly Function

The Reassembly function reconstructs the Initial PDU from the Derived PDUs generated by the operation of the Segmentation Function on the Initial PDU (and, recursively, on subsequent Derived PDUs). A bound on the time during which segments (Derived PDUs) of an Initial PDU will be held at a reassembly point before being discarded is provided, so that reassembly resources may be released when it is no longer expected that any outstanding segments of the Initial PDU will arrive at the reassembly point. Upon reception of a Derived PDU, a reassembly timer is initiated with a value which indicates the amount of

time which must elapse before any outstanding segments of the Initial PDU shall be assumed to be lost. When this timer expires, all segments (Derived PDUs) of the Initial PDU held at the reassembly point are discarded, the resources allocated for those segments are freed, and if selected, an Error Report is generated (see Clause 6.10). While the exact relationship between reassembly lifetime and PDU lifetime is a local matter, the Reassembly Function must preserve the intent of the PDU lifetime. Consequently, the reassembly function must discard PDUs whose lifetime would otherwise have expired had they not been under the control of the reassembly function.

Note:

1. Methods of bounding reassembly lifetime are discussed in Annex B.

2. The Segmentation and Reassembly functions are intended to be used in such a way that the fewest possible segments are generated at each segmentation point and reassembly takes place at the final destination of a PDU. However, other

schemes which

(a) interact with the routing algorithm to favor paths on which fewer segments are generated;

(b) generate more segments than absolutely required in order to avoid additional segmentation at some subsequent point; or

(c) allow partial or full reassembly at some intermediate point along the route

are not precluded. The information necessary to enable the use of one of these alternative strategies may be made available through the operation of a Network Layer Management function or by other means.

3. The originator of the Initial PDU determines the value of the Segmentation Permitted flag in the Initial PDU and all Derived PDUs (if any). Partial or full reassembly in an intermediate system (Note 2 (c) above) cannot change this value in the Initial PDU or any PDU derived from it, and cannot therefore add or remove the segmentation part of the header.

6.9 Discard PDU Function

This function performs all of the actions necessary to free the resources reserved by the network-entity when any of the following situations is encountered (Note: the list is not exhaustive):

(a) A violation of protocol procedure has occurred.

(b) A PDU is received whose checksum is inconsistent with its contents.

(c) A PDU is received, but due to local congestion, it cannot be processed.

(d) A PDU is received whose header cannot be analyzed.

(e) A PDU is received which cannot be segmented and cannot be forwarded because its length exceeds the maximum service data unit size supported by any underlying service available for transmission of the PDU to the next network-entity on the chosen route.

(f) A PDU is received whose destination address is unreachable or unknown.

(g) Incorrect or invalid source routing was specified. This may include a syntax error in the source routing field, an unknown or unreachable address in the source routing field, or a path which is not acceptable for other reasons.

(h) A PDU is received whose PDU lifetime has expired or whose lifetime expires during reassembly.

(i) A PDU is received which contains an unsupported option.

6.10 Error Reporting Function

6.10.1 Overview

This function causes an attempt to return an Error Report PDU to the source network-entity when a protocol data unit is discarded in accordance with Clause 6.9.

The Error Report PDU identifies the discarded PDU, specifies the type of error detected, and identifies the location in the header of the discarded PDU at which the error was detected. At least the entire header of the Discarded PDU (and, at the discretion of the originator of the Error Report PDU none, all, or part of the data field) is placed in the data field of the Error Report PDU.

The originator of a Data PDU may control the generation of Error Report PDUs. An Error Report flag in the original PDU is set by the source network-entity to indicate that an Error Report PDU is to be returned if the Initial PDU or any PDUs derived from it are discarded; if the flag is not set, Error Reports are to be suppressed.

Note:

1. The suppression of Error Report PDUs is controlled by the

originating network-entity and not by the NS User. Care should be exercised by the originator with regard to suppressing ER PDUs so that error reporting is not suppressed for every PDU generated.

2. Non-receipt of an Error Report PDU does not imply correct delivery of a PDU issued by a source network-entity.

6.10.2 Requirements

An Error Report PDU shall not be generated to report the discard of an Error Report PDU.

An Error Report PDU shall not be generated to report the discard of a Data PDU unless that PDU has the Error Report flag set to allow Error Reports.

If a Data PDU is discarded, and the Error Report flag has been set to allow Error Reports, an Error Report PDU shall be generated if the reason for discard is one of the reasons for discard enumerated in Clause 6.9, subject to the conditions described in Clause 6.10.4.

Note:

If a Data PDU with the E/R flag set to allow Error Reports is discarded for any other reason, an ER PDU may be generated (as an implementation option).

6.10.3 Processing of Error Reports

An Error Report PDU is composed from information contained in the header of the discarded Data PDU to which the Error Report refers. The contents of the Source Address field of the discarded Data PDU are used as the Destination Address of the Error Report PDU. This value, which in the context of the Data PDU was used as an NSAP Address, is used in the context of the Error Report PDU as the network-entity title of the network-entity that originated the Data PDU. The network- entity title of the originator of the Error Report PDU is conveyed in the Source Address field of the header of the Error Report PDU. The value of the Lifetime field is determined in accordance with Clause 6.4. Optional parameters are selected in accordance with Clause 6.10.4.

Segmentation of Error Report PDUs is not permitted; hence, no Segmentation Part is present. The total length of the ER PDU in octets is placed in the Segment Length field of the ER PDU header. This field is not changed during the lifetime of the ER PDU. If the originator of the ER PDU determines that the size of the ER PDU exceeds the maximum service data unit size of the underlying service, the ER PDU shall be truncated to the maximum service data unit size (see Clause 5.5.3) and forwarded with no other change. Error Report PDUs are routed and forwarded by intermediate-system network-entities in the

same way as Data PDUs.

Note:

The requirement that the underlying service assumed by the CLNP must be capable of supporting a service data unit size of at least 512 octets guarantees that the entire header of the discarded Data PDU can be conveyed in the data field of any ER PDU.

When an ER PDU is decomposed upon reaching its destination, information that may be used to interpret and act upon the Error Report is obtained as follows. The network-entity title recovered from the NPAI in the Source Address field of the ER PDU header is used to identify the network-entity which generated the Error Report. The reason for generating the Error Report is extracted from the Options Part of the PDU header. The entire header of the discarded Data PDU (and part or all of the original user data) is extracted from the data field of the ER PDU to assist in determining the nature of the error.

6.10.4 Relationship of Data PDU Options to Error Reports

The generation of an Error Report is affected by options that are present in the corresponding Data PDU. The presence of options in the original Data PDU that are not supported by the system which has dis-

carded that PDU may cause the suppression of an Error Report even if the original Data PDU indicated that an Error Report should be generated in the event of a discard.

The processing of an Error Report is also affected by options which are present in the corresponding Data PDU. In particular, options selected for the original Data PDU affect which options are included in the corresponding Error Report PDU. The selection of options for an Error Report PDU is governed by the following requirements:

(a) If the Priority Option or the QoS Maintenance Option is selected in the original Data PDU, and the system generating the Error Report PDU supports the option, then the Error Report PDU shall specify the option.

(b) If the Security Option is selected in the Data PDU, and the system generating the Error Report supports this option, then the Error Report PDU shall specify the option using the value that was specified in the original Data PDU. If the system does not support the Security Option, an Error Report must not be generated for a Data PDU that selects the Security Option.

(c) If the Complete Source Route Option is selected in the original Data PDU, and the system generating the Error Report PDU supports this option, then the error Report shall specify the Complete Source Route option. The Source Route parameter value is obtained by extracting from the original Data PDU that portion of the complete source route that has already been traversed, and reversing the

order of network-entity titles which comprise the list. If the system does not support the Complete Source Route Option, an Error Report must not be generated for a Data PDU that selects the Complete Source Route option.

(d) The Padding, Partial Source Routing, and Record Route Options, if supported, may be specified in the Error Report PDU.

Note:
The values of the optional parameters in (d) above may be derived as a local matter, or they may be based upon the corresponding values in the original Data PDU.

6.11 PDU Header Error Detection

The PDU Header Error Detection function protects against failure of intermediate or end-system network-entities due to the processing of erroneous information in the PDU header. The function is realized by a checksum computed on the entire PDU header. The checksum is verified at each point at which the PDU header is processed. If the checksum calculation fails, the PDU must be discarded. If PDU header fields are modified (for example, due to operation of the lifetime function), then the checksum is modified so that the checksum remains valid.

The use of the Header Error Detection function is optional, and is selected by the originating network-entity. If the function is not used, the checksum field of the PDU header is set to zero.

If the function is selected by the originating network-entity, the value of the checksum field causes the following formulae to be satisfied:

(The Sum from i=1 to L of a(i)) (mod 255) = 0

(The Sum from i=1 to L of (L - i + 1) * a(i)) (mod 255) = 0

where L = the number of octets in the PDU header, and a(i) = the value of the octet at position i. The first octet in the PDU header is considered to occupy position i = 0.

When the function is in use, neither octet of the checksum field may be set to zero.

Note:

1. To ensure that inadvertent modification of a header while a PDU is being processed by an intermediate system (for example, due to a memory fault) may still be detected by the PDU Header Error function, an intermediate system network-

entity must not recompute the checksum for the entire header, even if fields are modified.

2. Annex C contains descriptions of algorithms which may be used to calculate the correct value of the checksum field when the PDU is created, and to update the value of the checksum field when the header is modified.

6.12 Padding Function

The padding function is provided to allow space to be reserved in the PDU header which is not used to support any other function. Octet alignment must be maintained.

Note:

An example of the use of this function is to cause the data field of a PDU to begin on a convenient boundary for the originating network-entity, such as a computer word boundary.

6.13 Security

The provision of protection services (e.g., data origin authentication, data confidentiality, and data integrity of a single connectionless-mode NSDU) is performed by the Security Function.

The Security Function is related to the Protection from Unauthorized Access Quality of Service parameter described in ISO 8348/AD1, Addendum to the Network Service Definition Covering Connectionless-mode Transmission. The function is realized through selection of the security parameter in the options part of the PDU header.

This Standard does not specify the way in which protection services are to be provided; it only provides for the encoding of security information in the PDU header. To facilitate interoperation between end-systems and network relay-systems by avoiding different interpretations of the same encoding, a means to distinguish user-defined security encodings from standardized security encodings is described in Clause 7.5.3.

Note:

As an implementation consideration, data origin authentication may be provided through the use of a cryptographically generated or enciphered checksum (unique from the PDU Header Error Detection mechanism); data confidentiality and data integrity may be provided via route control mechanisms.

6.14 Source Routing Function

The Source Routing function allows the originator to specify the path a generated PDU must take. Source routing may only be selected by the

originator of a PDU. Source Routing is accomplished using a list of network-entity titles held in a parameter within the options part of the PDU header. The length of this parameter is determined by the originating network-entity, and does not change as the PDU traverses the network.

The Source Route parameter includes information used by the originating end-system when determining the initial route of the PDU. Only the titles of intermediate system network-entities are included in the list; the network-entity title of the destination of the PDU is not included in the list.

Associated with the list of network-entity titles is an indicator which identifies the next entry in the list to be used; this indicator is advanced by the receiver of the PDU when the next title in the list matches its own. The indicator is updated as the PDU is forwarded so as to identify the appropriate entry at each stage of relaying.

Two forms of the Source Routing function are provided. The first form, referred to as Complete Source Routing, requires that the specified path must be taken; that is, only those systems identified in the list may be visited by the PDU while en route to the destination, and each system must be visited in the order specified. If the specified path cannot be taken, the PDU must be discarded. Clause 6.10 describes the circumstances in which an attempt shall be made to inform the originator of the discard using the Error Reporting func-

tion.

The second form is referred to as Partial Source Routing. Again, each system identified in the list must be visited in the order specified while en route to the destination. However, with this form of source routing the PDU may take any path necessary to arrive at the next intermediate system in the list, which may include visiting intermediate systems that are not identified in the list. The PDU will not be discarded (for source routing related reasons) unless one of the systems specified cannot be reached by any available route.

6.15 Record Route Function

The Record Route function records the path(s) taken by a PDU as it traverses a series of intermediate systems. A recorded route consists of a list of network-entity titles held in a parameter within the options part of the PDU header. The length of this parameter is determined by the originating network-entity, and does not change as the PDU traverses the network.

The list is constructed as the PDU is forwarded along a path towards its destination. Only the titles of intermediate system network-entities are included in the recorded route. The network-entity title of the originator of the PDU is not recorded in the list.

When an intermediate system network-entity processes a PDU containing the Record Route parameter, the system adds its own networkentity title at the end of the list of recorded network-entity titles. An indicator is maintained to identify the next available octet to be used for recording of route. This indicator is updated as entries are added to the list as follows. The length of the entry to be added to the list is added to the value of the next available octet indicator, and this sum is compared with the length of the Record Route parameter. If the addition of the entry to the list would exceed the size of the parameter, the next available octet indicator is set to indicate that route recording has been terminated. The network-entity title is not added to the list. The PDU may still be forwarded to its final destination, without further addition of network-entity titles.

If the addition of the entry would not exceed the size of the Record Route parameter, the next available octet indicator is updated with the new value, and the network-entity title is added to the head of the list after the other entries have been moved.

Two forms of the Record Route function are provided. The first form is referred to as Complete Route Recording. It requires that the list of network-entity titles be a complete and accurate record of all intermediate systems visited by a PDU (including Derived PDUs), except when a shortage of space in the record route option field causes termination of recording of route, as described above. When Complete Route Recording is selected, PDU reassembly at intermediate systems is performed only when the Derived PDUs that are reassembled

all took the same route; otherwise, the PDU is discarded, and if selected, an Error Report is generated (see Clause 6.10).

The second form is referred to as Partial Route Recording. It also requires a record of intermediate systems visited by a PDU. When Partial Route Recording is selected, PDU reassembly at intermediate systems is always permitted. When reassembly is performed at an intermediate system, the route recorded in any of the Derived PDUs may be placed in the PDU resulting from the reassembly.

Note:

The Record Route function is intended to be used in the diagnosis of subnetwork problems and/or to provide a return path that could be used as a source route in a subsequent PDU.

6.16 Quality of Service Maintenance Function

The Quality of Service Maintenance function provides information to network-entities in intermediate systems which may be used to make routing decisions where such decisions affect the overall QoS provided to NS users. This information is conveyed to intermediate system network- entities in a parameter in the options part of the PDU header.

In those instances where the QoS requested cannot be maintained, intermediate system network-entities shall attempt to deliver the PDU at a QoS different from the QoS requested. Intermediate system network-entities do not necessarily provide a notification of failure to meet the requested Quality of Service.

6.17 Priority Function

The Priority function allows a PDU with a numerically higher priority value to be processed preferentially with respect to other PDUs with numerically lower priority values. The function is realized through selection of a parameter in the options part of the PDU header.

The lowest priority value is zero; a source network-entity that does not support the Priority function must set the Priority value to zero. The Priority function provides a means whereby the resources of end and intermediate system network-entities, such as outgoing transmission queues and buffers, can be used preferentially to process higher-priority PDUs ahead of lower-priority PDUs. The specific action taken by an individual network-entity to support the Priority function is a local matter.

6.18 Congestion Notification Function

To allow NS Users to take appropriate action when congestion is experienced within the NS provider, intermediate systems may inform the destination network-entity of congestion through the use of a flag in

the QoS Maintenance parameter in the options part of the PDU header. The value of this flag is initially set to zero (0) by the originator of the PDU and may be set to one (1) by any intermediate system which processes the PDU to indicate that it is experiencing congestion. The criteria for determining when this action is to be taken are a local matter.

Note:

Congestion typically corresponds to inavailability of buffer space to maintain output queues. An appropriate policy for indicating congestion may be based upon the depth of the output queue selected for a PDU (according to its destination address or other routing information). When the depth of a particular output queue exceeds a certain proportion of the depth of that queue, an intermediate system will start to discard PDUs. The intermediate system will set the Congestion Experienced flag in the next PDU to be forwarded and may continue to do so until the condition is alleviated.

6.19 Classification of Functions

Implementations are not required to support all of the functions described in Clauses 6.1 through 6.18. Functions are divided into three categories:

Type 1: These functions must be supported.

Type 2: These functions may or may not be supported.
If an implementation does not support a Type 2 function, and the function is selected in a PDU, then that PDU must be discarded, and an Error Report PDU must be generated and forwarded to the originating network-entity, providing that the Error Report flag is set and the conditions of Clause 6.10.4 are satisfied.

Type 3: These functions may or may not be supported.
If an implementation does not support a Type 3 function, and the function is selected in a PDU, then the function is not performed, and the PDU is processed exactly as though the function had not been selected. The protocol data unit shall not be discarded for this reason.

Table 4 shows how the functions are divided into these three categories:

FUNCTION	FULL PROTOCOL	NON SEGMENTING SUBSET	INACTIVE SUBSET
PDU Composition	1	1	1
PDU Composition	1	1	1
Header Format Analysis	1	1	1
PDU Lifetime Control	1	1	N/A
Route PDU	1	1	N/A
Forward PDU	1	1	N/A

Segment PDU	1	N/A	N/A
Reassemble PDU	1	N/A	N/A
Discard PDU	1	1	N/A
Error Reporting (Note 1)	1	1	N/A
Header Error Detection (Note 1)	1	1	N/A
Security	1	2	N/A
Complete Source Routing	1	2	N/A
Complete Route Recording	2	2	N/A
Partial Source Routing	3	3	N/A
Partial Route Recording	3	3	N/A
Priority	3	3	N/A
QoS Maintenance	3	3	N/A
Congestion Notification	3	3	N/A
Padding	3	3	N/A

Table 4: Categorization of Protocol Functions

Note:

1. While the Error Reporting and Header Error Detection functions must be provided, they are provided only when selected by the sending Network Service user.

2. The rationale for the inclusion of type 3 functions is that in the case of some functions it is more important to forward the PDUs between intermediate systems or deliver them to an end-system than it is to support the functions. Type 3 functions should be used in those cases where they are of an advisory nature; they cannot cause a PDU to be discarded when they are not supported.

7 Structure and Encoding of PDUs

7.1 Structure

All Protocol Data Units shall contain an integral number of octets. The octets in a PDU are numbered starting from one (1) and increasing in the order in which they are submitted to the underlying service. The bits in an octet are numbered from one (1) to eight (8), where bit one (1) is the low-order (least significant) bit.

When consecutive octets are used to represent a binary number, the lower octet number has the most significant value.

Any implementation supporting this protocol is required to state in its specification the way in which octets are transferred, using the terms "most significant bit" and "least significant bit". The PDUs of

this protocol are defined using the terms "most significant bit" and "least significant bit".

Note:

When the encoding of a PDU is represented using a diagram in this Clause the following representation is used:

a) octets are shown with the lowest numbered octet to the left, higher number octets being further to the right;
b) within an octet, bits are shown with bit eight (8) to the left and bit one (1) to the right.

PDUs shall contain, in the following order:

1. the fixed part;
2. the address part;
3. the segmentation part, if present;
4. the Options part, if present;

and the data field, if present. This structure is illustrated in Figure 2:

7.2 Fixed Part

7.2.1 General

The fixed part of the PDU header contains frequently occurring parameters including the type code (DT or ER) of the protocol data unit. The length and the structure of the fixed part are defined by the PDU code.

The fixed part has the following format:

Part	Described in
Fixed Part	Section 7.2
Address Part	Section 7.3
Segmentation Part	Section 7.4
Options Part	Section 7.5
Data	Section 7.6

Figure 2: PDU Structure

	Octet
Network Layer Protocol Identifier	1

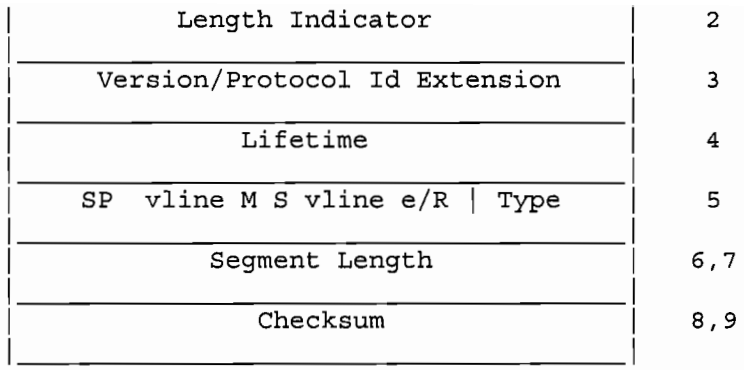


Figure 3: PDU Header -- Fixed Part

7.2.2 Network Layer Protocol Identifier

The value of this field is set to binary 1000 0001 to identify this Network Layer protocol as ISO 8473, Protocol for Providing the Connectionless- mode Network Service. The value of this field is set

to binary 0000 0000 to identify the Inactive Network Layer protocol subset.

7.2.3 Length Indicator

The length is indicated by a binary number, with a maximum value of 254 (1111 1110). The length indicated is the length in octets of the header, as described in Clause 7.1. The value 255 (1111 1111) is reserved for possible future extensions.

Note:
The rules for forwarding and segmentation guarantee that the header length is the same for all segments (Derived PDUs) of the Initial PDU, and is the same as the header length of the Initial PDU. The size of a PDU header will not change due to operation of any protocol function.

7.2.4 Version/Protocol Identifier Extension

The value of this field is binary 0000 0001, which identifies the standard Version 1 of ISO 8473, Protocol for Providing the Connectionless-mode Network Service.

7.2.5 PDU Lifetime

The PDU Lifetime field is encoded as a binary number representing the remaining lifetime of the PDU, in units of 500 milliseconds.

7.2.6 Flags

7.2.6.1 Segmentation Permitted

The Segmentation Permitted flag indicates whether segmentation is permitted. Its value is determined by the originator of the PDU and cannot be changed by any other network-entity for the lifetime of the Initial PDU and any Derived PDUs.

A value of one (1) indicates that segmentation is permitted. A value of zero (0) indicates that the non-segmenting protocol subset is employed. When the value of zero is selected, the segmentation part of the PDU header is not present, and the Segment Length field serves as the Total Length field (see Clause 7.2.8).

7.2.6.2 More Segments

The More Segments flag indicates whether the data segment in this PDU contains (as its last octet) the last octet of the User Data in the NSDU. When the More Segments flag is set to one (1), segmentation has taken place and the last octet of the NSDU is not contained in this PDU. The More Segments flag cannot be set to one (1) if the Segmentation Permitted flag is not set to one (1).

When the More Segments flag is set to zero (0), the last octet of the Data Part of the PDU is the last octet of the NSDU.

7.2.6.3 Error Report

When the Error Report flag is set to one, the rules in Clause 6.10 are used to determine whether to generate an Error Report PDU if it is necessary to discard this Data PDU.

When the Error Report flag is set to zero, discard of the Data PDU will not cause the generation of an Error Report PDU.

7.2.7 Type Code

The Type code field identifies the type of the protocol data unit. Allowed values are given in Table 5:

	Bits	5	4	3	2	1
DT PDU		1	1	1	0	0
ER PDU		0	0	0	0	1

Table 5: Valid PDU Types

7.2.8 PDU Segment Length

The Segment Length field specifies the entire length, in octets, of the Derived PDU, including both header and data (if present). When the full protocol is employed and a PDU is not segmented, the value of this field is identical to the value of the Total Length field lo-

cated in the Segmentation Part of the header.

When the non-segmenting protocol subset is employed, no segmentation part is present in the header. In this subset, the Segment Length field specifies the entire length of the Initial PDU, including both header and data (if present). The Segment Length field is not changed for the lifetime of the PDU.

7.2.9 PDU Checksum

The checksum is computed on the entire PDU header. For the Data PDU, this includes the segmentation and options parts (if present). For the Error Report PDU, this includes the reason for discard field as well.

A checksum value of zero is reserved to indicate that the checksum is to be ignored. The operation of the PDU Header Error Detection function (Clause 6.11) ensures that the value zero does not represent a

valid checksum. A non-zero value indicates that the checksum must be processed. If the checksum calculation fails, the PDU must be discarded.

7.3 Address Part

7.3.1 General

Address parameters are distinguished by their location, immediately following the fixed part of the PDU header. The address part is illustrated Figure 4:

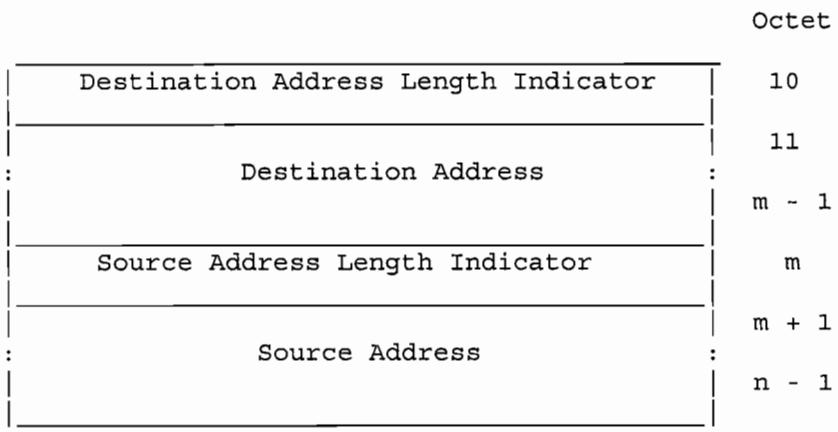


Figure 4: PDU Header -- Address Part

7.3.1.1 Destination and Source Addresses

The Destination and Source addresses used by this protocol are Network Service Access Point addresses as defined in ISO 8348/AD2, Ad-

dendum to the Network Service Definition Covering Network Layer Addressing.

The Destination and Source Addresses are variable length. The Destination and Source Address fields are encoded as Network Protocol Address Information using the Preferred Binary Encoding defined in Clause 8.3.1 of ISO 8348/AD2.

The Destination Address Length Indicator field specifies the length of the Destination Address in octets. The Destination Address field follows the Destination Address Length Indicator field.

The Source Address Length Indicator field specifies the length of the Source Address in octets. The Source Address Length Indicator field follows the Destination Address field. The Source Address field follows the Source Address Length Indicator field.

Each address parameter is encoded as illustrated in Table 5:

Octet n	Address parameter Length Indicator (e.g., 'm')
Octets n + 1 thru n + m	Address Parameter Value

Figure 5: Address Parameters

7.4 Segmentation Part

If the Segmentation Permitted Flag in the Fixed Part of the PDU Header (Octet 4, Bit 8) is set to one, the segmentation part of the header, illustrated in Figure 6, must be present:

If the Segmentation Permitted flag is set to zero, the non-segmenting protocol subset is in use.

	Octet
Data Unit Identifier	n, n + 1
Segment Offset	n + 2, n + 3
Total Length	n + 4, n + 5

Figure 6: PDU Header -- Segmentation Part

7.4.1 Data Unit Identifier

The Data Unit Identifier identifies an Initial PDU (and hence, its Derived PDUs) so that a segmented data unit may be correctly reassembled. The Data Unit Identifier size is two octets.

7.4.2 Segment Offset

For each Derived PDU, the Segment Offset field specifies the relative position of the segment contained in the data field of the Derived PDU with respect to the start of the data field of the Initial PDU. The offset is measured in units of octets. The offset of the first segment (and hence, the Initial PDU) is zero; an unsegmented (Initial PDU) has a segment offset value of zero (0). The value of this field shall be a multiple of eight 8).

7.4.3 PDU Total Length

The Total Length field specifies the entire length of the Initial PDU, including both the header and data. This field is not changed for the lifetime of the Initial PDU (and hence, its Derived PDUs).

7.5 Options Part

7.5.1 General

The options part is used to convey optional parameters. The options part of the PDU header is illustrated below:

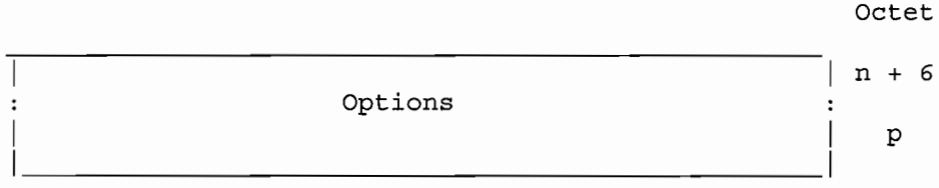


Figure 7: PDU Header -- Options Part

If the options part is present, it may contain one or more parameters. The number of parameters that may be contained in the options part is constrained by the length of the options part, which is determined by the following formula:

PDU Header Length -(length of fixed part+length of address part+length of segmentation part)

and by the length of the individual optional parameters.

Parameters defined in the options part may appear in any order. Duplication of options is not permitted. Receipt of a Protocol Data Unit with an option duplicated should be treated as a protocol error. The rules governing the treatment of protocol errors are described in

Clause 6.10, Error Reporting Function.

The encoding of parameters contained within the options part of the PDU header is illustrated in Table 6:

Octets

n	Parameter Code
n + 1	Parameter Length (e.g.m)
n + 2 to n + m + 1	Parameter Value

Table 6: Encoding of Parameters

The parameter code field is coded in binary and, without extensions, provides a maximum of 255 different parameters. No parameter codes use bits 8 and 7 with the value 00, so the actual maximum number of parameters is lower. A parameter code of 255 (binary 1111 1111) is reserved for possible future extensions.

The parameter length field indicates the length, in octets, of the parameter value field. The length is indicated by a positive binary number, m, with a theoretical maximum value of 254. The practical maximum value of m is lower. For example, in the case of a single parameter contained within the options part, two octets are required for the parameter code and the parameter length indicators. Thus, the value of m is limited to:

m = 252-(length of fixed part +length of address part +length of segmentation part)

For each succeeding parameter the maximum value of m decreases. The parameter value field contains the value of the parameter identified in the parameter code field.

The following parameters are permitted in the options part.

7.5.2 Padding

The padding parameter is used to lengthen the PDU header to a convenient size (See Clause 6.12).

Parameter Code: 1100 1100

Parameter Length: variable

Parameter Value: any value is allowed

7.5.3 Security

This parameter allows a unique and unambiguous security level to be assigned to a protocol data unit.

Parameter Code: 1100 0101

Parameter Length: variable

Parameter Value: The high order two bits of the first octet specify the Security Format Code, where:

Security Format Code	Type of Security Field:
00	Reserved
01	Source Address Specific
10	Destination Address Specific
11	Globally Unique

The rest of the first octet is reserved and must be zero. The remainder of the Parameter Value field specifies the security level as described in the following Clauses.

7.5.3.1 Source Address Specific

The Security Format Code value of binary "01" indicates that the remaining octets of the parameter value field specify a security level which is unique and unambiguous in the context of the security classification system employed by the authority responsible for assigning the source NSAP Address.

7.5.3.2 Destination Address Specific

The Security Format Code value of binary "10" indicates that the remaining octets of the parameter value field specify a security level which is unique and unambiguous in the context of the security classification system employed by the authority responsible for assigning the destination NSAP Address.

7.5.3.3 Globally Unique Security

The Security Format Code value of binary "11" indicates that the remaining octets of the parameter value field specify a globally unique and unambiguous security level. This security classification system is not specified in this Standard.

7.5.4 Source Routing

The source routing parameter specifies, either completely or partially, the route to be taken from Source Network Address to Destination Network Address.

Parameter Code: 1100 0101

Parameter Length: variable

Parameter Value: 2 octets of control information succeeded by a concatenation of ordered network-entity title entries (ordered from source to destination)

The first octet of the parameter value is the type code, and has the following significance:

```
0000 0000    partial source routing
0000 0001    complete source routing
             <all other values reserved>
```


The second octet indicates the octet offset of the next network-entity title entry to be processed in the list. It is relative to the start of the parameter, such that a value of three (3) indicates that the next network-entity title entry begins immediately after this control octet. Successive octets are indicated by correspondingly larger values of this indicator.

The third octet begins the network-entity title list. The list consists of variable length network-entity title entries. The first octet of entry identifies the length of the network-entity title which comprises the re- mainder of the entry.

7.5.5 Recording of Route

The recording of route parameter identifies the route of intermediate systems traversed by the PDU.

Parameter Code: 1100 1011

Parameter Length: variable

Parameter Value: 2 octets of control information succeeded by a con catenation of ordered network-entity title entries (ordered from destination to source)

The first octet of the parameter value is the type code, and has the following significance:

```
0000 0000    Partial Recording of Route in progress
0000 0001    Complete Recording of Route in progress
             <all other values reserved>
```

The second octet identifies the first octet not currently used for a recorded network-entity title, and therefore also the end of the list. It is encoded relative to the start of the parameter value, such that a value of three (3) indicates that no network-entity titles have yet been recorded. A value of all ones is used to indicate that route recording has been terminated.

The third octet begins the network-entity title list. The list consists of variable length network-entity title entries. The first octet of each entry specifies the length of the network-entity title comprising the remainder of the entry. Network-entity title entries are always added to the beginning of the list; that is, the most recently added entry will begin in the third octet of the parameter value.

Note:
The length of the Record Route parameter is determined by the originator of the PDU and is not changed during the lifetime of the PDU; hence, the operation of the Record Route function does

not affect the length of the header.

7.5.6 Quality of Service Maintenance

The Quality of Service parameter conveys information about the quality of service requested by the originating Network Service user. Network-entities in intermediate systems may (but are not required to) make use of this information as an aid in selecting a route when more than one route satisfying other routing criteria is available and the available routes are known to differ with respect to Quality of Service see Clause 6.16).

Parameter Code: 1100 0011
Parameter Length: variable
Parameter Value: The high order two bits of the first octet specify the QoS Format Code, where:

QoS Format Code	Type of QoS Field
00	Reserved
01	Source Address Specific
10	Destination Address Specific
11	Globally Unique

The rest of the first octet is reserved and must be zero. The remainder of the Parameter Value field specifies the QoS as described in the following Clauses.

7.5.6.1 Source Address Specific

The QoS Format Code value of binary "01" indicates that the remaining octets of the parameter value field specify a QoS which is unique and unambiguous in the context of the QoS Maintenance system employed by the authority responsible for assigning the source NSAP Address.

7.5.6.2 Destination Address Specific

The QoS Format Code value of binary "10" indicates that the remaining octets of the parameter value field specify a QoS which is unique and unambiguous in the context of the QoS Maintenance system employed by

the authority responsible for assigning the destination NSAP Address.

7.5.6.3 Globally Unique QoS

The QoS Format Code value of binary "11" indicates that the remainder of the parameter value field specifies a globally unique QoS Maintenance field. When the globally unique QoS Maintenance function is employed, the parameter value field must have a total length of one octet, which is assigned the following values:

Bits 8 and 7: QoS Format Code of binary "11"

Bit 6: Reserved
Bit 5: sequencing vs. transit delay
Bit 4: congestion experienced
Bit 3: transit delay vs. cost
Bit 2: residual error probability vs. transit delay
Bit 1: residual error probability vs. cost

Bit 5 is set to one to indicate that, where possible, routing decisions should favor sending all PDUs to the specified destination NSAP address over a single path (in order to maintain sequence) over minimizing transit delay. A value of zero (0) indicates that, where possible, routing decisions should favor low transit delay over sequence preservation.

Bit 4 is set to zero by the network-entity which originates the protocol data unit. It is set to one by an intermiediate system to indicate that this PDU has visited a congested intermediate system, and appropriate action should be taken by the destination network-entity. Once the congestion experienced bit is set by an intermediate system, it may not be reset by any intermediate system traversed by the PDU further along the path towards the destination.

Bit 3 is set to one to indicate that, where possible, routing decisions should favor low transit delay over low cost. A value of 0 indicates that routing decisions should favor low cost over low transit delay.

Bit 2 set to one to indicate that, where possible, routing decisions should favor low residual error probability over low transit delay. A value of zero indicates that routing decisions should favor low transit delay over low residual error probability.

Bit 1 is set to one to indicate that, where possible, routing decisions should favor low residual error probability over low cost. A value of 0 indicates that routing decisions should favor low cost over low residual error probability.

7.5.7 Priority

The value of the Priority parameter indicates the relative priority of the protocol data unit. Intermediate systems that support this

option shall make use of this information in routing and in ordering PDUs for transmission.

Parameter Code: 1100 1101

Parameter Length: one octet

Parameter Value: 0000 0000 - Normal (Default) through
0000 1110 - Highest
<all other values reserved>

The values 0000 0001 through 0000 1110 are to be used for higher priority protocol data units. If an intermediate system does not support this option, all PDUs shall be treated as if the field had the value 0000 0000.

7.6 Data Part

The Data part of the PDU is structured as an ordered multiple of octets, which is identical to the same ordered multiple of octets specified for the NS-Userdata parameter of the N-UNITDATA Request and Indication primitives. The data field is illustrated in Figure 8:

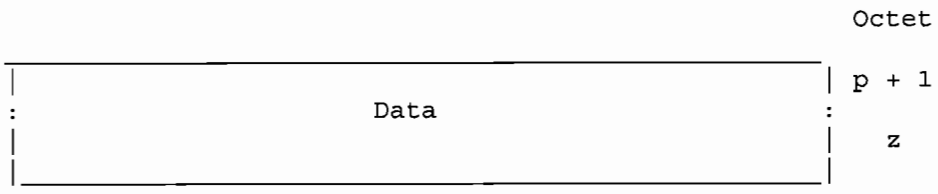
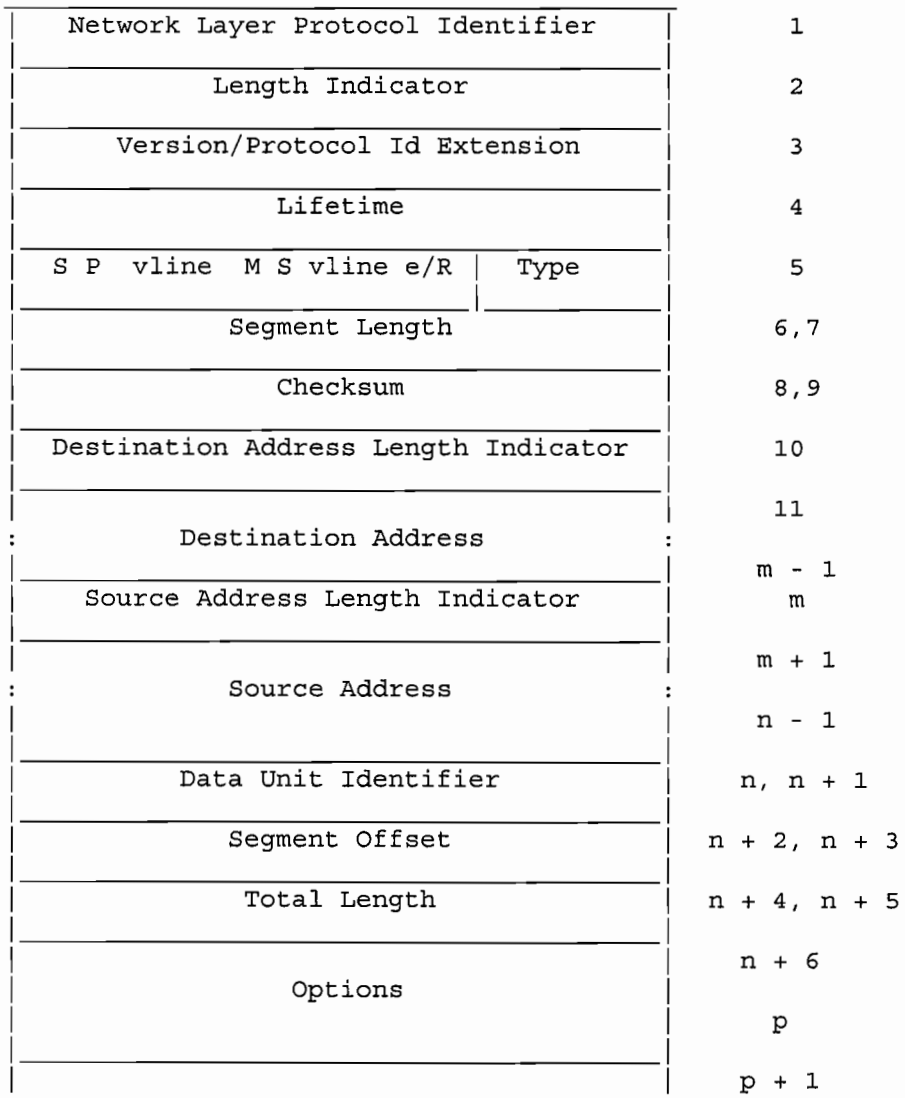


Figure 8: PDU Header -- Data Field

7.7 Data (DT) PDU

7.7.1 Structure

The DT PDU has the following format:



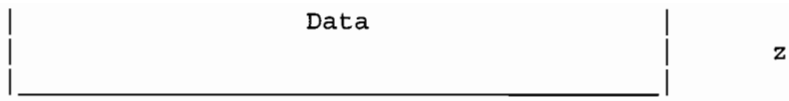


Figure 9: DT PDU

7.7.1.1 Fixed Part

1)	Network Layer Protocol Identifier	See Clause 7.2.2
2)	Length Indicator	See Clause 7.2.3
3)	Version/Protocol Id Extension	See Clause 7.2.4
4)	Lifetime	See Clause 7.2.5
5)	SP, MS, E/R	See Clause 7.2.6
6)	Type Code	See Clause 7.2.7
7)	Segment Length	See Clause 7.2.8
8)	Checksum	See Clause 7.2.9

7.7.1.2 Addresses

See Clause 7.3.

7.7.1.3 Segmentation

See Clause 7.4.

7.7.1.4 Options

See Clause 7.5.

7.7.1.5 Data

See Clause 7.7.

7.8 Inactive Network Layer Protocol

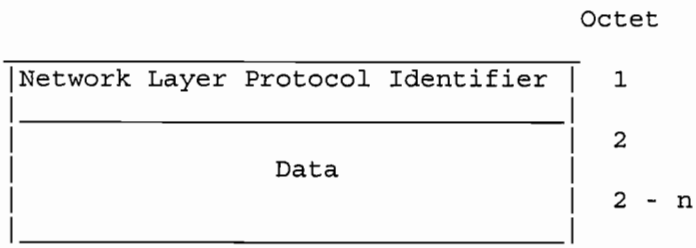


Figure 10: Inactive Network Layer Protocol

7.8.1 Network Layer Protocol Id

The value of the Network Layer Protocol Identifier field is binary

zero (0000 0000).

7.8.2 Data Field

The length of the NS-Userdata parameter is constrained to be less than or equal to the value of the length of the SN-Userdata parameter minus one (see Clause 7.7).

7.9 Error Report PDU (ER)

7.9.1 Structure

The ER PDU has the following format:

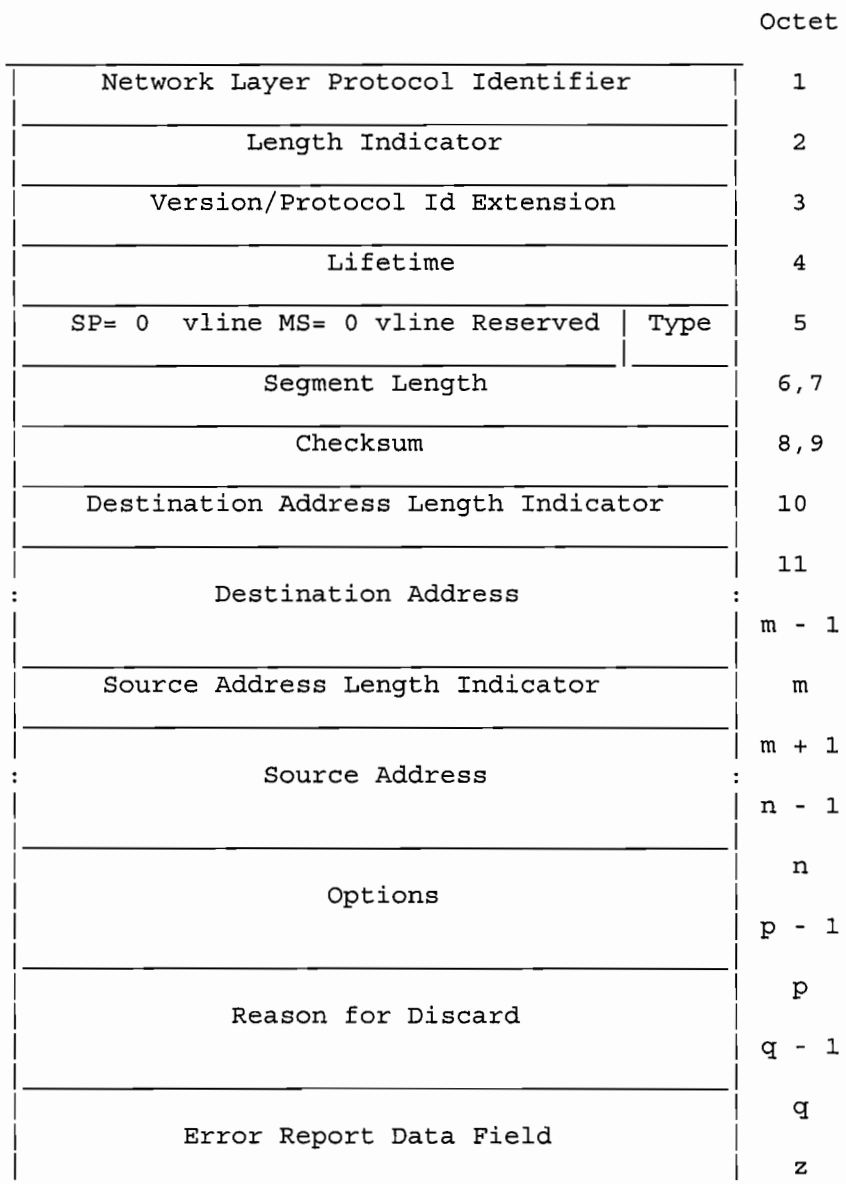




Figure 11: Error Report PDU

7.9.1.1 Fixed Part

The fixed part of the Error Report Protocol Data Unit is composed in the same way as a new (Initial) Data PDU. References are provided to previous Clauses describing the encoding of the fields comprising the fixed part:

1)	Network Layer Protocol Identifier	See Clause 7.2.2
2)	Length Indicator	See Clause 7.2.3
3)	Version/Protocol Id Extension	See Clause 7.2.4
4)	Lifetime	See Clause 7.2.5
5)	SP, MS, E/R	Always set to zero, (See Clause 6.10)
6)	Type Code	See Clause 7.2.7
7)	Segment Length	See Clause 7.2.8
8)	Checksum	See Clause 7.2.9

7.9.1.2 Addresses

See Clause 7.3.

The Destination Address specifies the network-entity title of the originator of the discarded PDU. The Source Address specifies the title of the intermediate-system or end-system network-entity initiating the Error Report PDU.

7.9.1.3 Options

See Clause 7.5.

7.9.1.4 Reason for Discard

This parameter is valid only for the Error Report PDU.

Parameter Code: 1100 0001
Parameter Length: two octets
Parameter Value: type of error encoded in binary. Values are listed in Table 7:

Parameter Value Octet 1	Octet 2	Class of Error	Meaning
0000	0000	General	Reason not specified
	0001		Protocol Procedure Error
	0010		Incorrect Checksum
	0011		PDU Discarded due to Congestion
	0100		Header Syntax Error (cannot be parsed)
	0101		Segmentation needed but not permitted
	0110		Incomplete PDU Received
	0111		Duplicate Option
1000	0000	Address	Destination Address Unreachable
	0001		Destination Address Unknown
1001	0000	Source Routing	Unspecified Source Routing Error
	0001		Syntax Error in Source Routing Field
	0010		Unknown Address in Source Routing Field
	0011		Path not Acceptable
1010	0000	Lifetime	Lifetime Expired while Data Unit in Transit
	0001		Lifetime Expired during Reassembly
1011	0000	PDU Discarded	Unsupported Option not Specified
	0001		Unsupported Protocol Version
	0010		Unsupported Security Option
	0011		Unsupported Source Routing Option
	0100		Unsupported Recording of Route Option
1100	0000	Reassembly	Reassembly interference

Table 7: Reasons for Discard

The first octet of the parameter value contains an error type code. If the error in the discarded Data PDU can be localized to a particular field, the number of the first octet of that field is stored in

the second octet of the reason for discard parameter field. If the error cannot be localized to a particular field, or if the error is a checksum error, then the value zero is stored in the second octet of the reason for discard parameter field.

7.9.1.5 Error Report Data Field

This field contains the entire header of the discarded Data PDU, and may contain some or all of the data field of the discarded PDU.

8 Conformance

For conformance to this International Standard, the ability to originate, manipulate, and receive PDUs in accordance with the full protocol (as opposed to the non-segmenting or Inactive Network Layer Protocol subsets) is required.

Additionally, conformance to the Standard requires provision of the protocol functions described in Clause 6. Provision of the optional functions described in Clause 6.18 and enumerated in Table 9-1 must meet the requirements described therein. Exceptions to this requirement are described in Clause 8.1 below.

Additionally, conformance to the Standard requires adherence to the structure and encoding of PDUs of Clause 7.

If and only if the above requirements are met is there conformance to this International Standard.

8.1 Provision of Functions for Conformance

The following table categorizes the functions in Clause 6 with respect to the type of system providing the function:

Note:

1. The support of the PDU Composition and Forward PDU functions is necessary for the generation of Error Report PDUs.

2. The Segment PDU function is in general mandatory for an intermediate system. However, a system which is to be connected only to subnetworks all offering the same maximum SDU size (such as identical Local Area Networks) will not need to perform this function and therefore does not need to implement it.

   If this function is not implemented, this shall be stated as part of the specification of the implementation.

3. The correct treatment of the padding function requires no processing. A conforming implementation shall support the function, to the extent of ignoring this parameter wherever it may appear.

4. This function may or may not be supported. If an implementation does not support this function, and the

function is selected in a PDU, then the PDU shall be discarded, and an ER PDU shall be generated and forwarded to the originating network-entity if the Error Report flag is set and the conditions of Clause 6.10.4 are satisfied.

5. This function may or may not be supported. If an implementation does not support this function, and the function is selected in a PDU, then the function is not performed and the PDU is processed exactly as though the function had not been selected. The PDU shall not be discarded for this reason.

Function	Send	Forward	Receive
PDU Composition	M	(Note 1)	(Note 1)
PDU Decomposition	M	-	M
Header Format Analysis	-	M	M
PDU Lifetime Control		M	I
Route PDU	-	M	-
Forward PDU	M	M	(Note 1)
Segment PDU	M	(Note 2)	-
Reassemble PDU	-	I	M
Discard PDU	-	M	M
Error Reporting	M	M	M
Header Error Detection	(Note 3)	M	M
Security	-	(Note 3)	(Note 4)
Complete Source Routing	-	(Note 4)	-
Complete Route Recording	-	(Note 4)	-
Partial Source Routing	-	(Note 5)	-
Partial Route Recording	-	(Note 5)	-
Priority	-	(Note 5)	-
QoS Maintenance	-	(Note 5)	-
Congestion Notification	-	(Note 5)	-
Padding	-	(Note 5)	(Note 3)

Table 8: Categorization of Functions
Key:

M: Mandatory Function; this function must be implemented.

-: Not applicable.

I: Implementation option, as described in the text.

NOTE: See notes above

Exhibit 10

RFC: 791

INTERNET PROTOCOL

DARPA INTERNET PROGRAM

PROTOCOL SPECIFICATION

September 1981

prepared for

Defense Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Boulevard
Arlington, Virginia 22209

by

Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, California 90291

TABLE OF CONTENTS

PREFACE iii

1. INTRODUCTION 1

1.1 Motivation 1
1.2 Scope 1
1.3 Interfaces 1
1.4 Operation 2

2. OVERVIEW 5

2.1 Relation to Other Protocols 9
2.2 Model of Operation 5
2.3 Function Description 7
2.4 Gateways 9

3. SPECIFICATION 11

3.1 Internet Header Format 11
3.2 Discussion 23
3.3 Interfaces 31

APPENDIX A: Examples & Scenarios 34
APPENDIX B: Data Transmission Order 39

GLOSSARY 41

REFERENCES 45

PREFACE

This document specifies the DoD Standard Internet Protocol. This document is based on six earlier editions of the ARPA Internet Protocol Specification, and the present text draws heavily from them. There have been many contributors to this work both in terms of concepts and in terms of text. This edition revises aspects of addressing, error handling, option codes, and the security, precedence, compartments, and handling restriction features of the internet protocol.

Jon Postel

Editor

September 1981

RFC: 791
Replaces: RFC 760
IENs 128, 123, 111,
80, 54, 44, 41, 28, 26

DARPA INTERNET PROGRAM
PROTOCOL SPECIFICATION

1. INTRODUCTION

1.1. Motivation

The Internet Protocol is designed for use in interconnected systems of packet-switched computer communication networks. Such a system has been called a "catenet" [1]. The internet protocol provides for transmitting blocks of data called datagrams from sources to destinations, where sources and destinations are hosts identified by fixed length addresses. The internet protocol also provides for fragmentation and reassembly of long datagrams, if necessary, for transmission through "small packet" networks.

1.2. Scope

The internet protocol is specifically limited in scope to provide the functions necessary to deliver a package of bits (an internet datagram) from a source to a destination over an interconnected system of networks. There are no mechanisms to augment end-to-end data reliability, flow control, sequencing, or other services commonly found in host-to-host protocols. The internet protocol can capitalize on the services of its supporting networks to provide various types and qualities of service.

1.3. Interfaces

This protocol is called on by host-to-host protocols in an internet environment. This protocol calls on local network protocols to carry the internet datagram to the next gateway or destination host.

For example, a TCP module would call on the internet module to take a TCP segment (including the TCP header and user data) as the data portion of an internet datagram. The TCP module would provide the addresses and other parameters in the internet header to the internet module as arguments of the call. The internet module would then create an internet datagram and call on the local network interface to transmit the internet datagram.

In the ARPANET case, for example, the internet module would call on a

local net module which would add the 1822 leader [2] to the internet datagram creating an ARPANET message to transmit to the IMP. The ARPANET address would be derived from the internet address by the local network interface and would be the address of some host in the

ARPANET, that host might be a gateway to other networks.

1.4. Operation

The internet protocol implements two basic functions: addressing and fragmentation.

The internet modules use the addresses carried in the internet header to transmit internet datagrams toward their destinations. The selection of a path for transmission is called routing.

The internet modules use fields in the internet header to fragment and reassemble internet datagrams when necessary for transmission through "small packet" networks.

The model of operation is that an internet module resides in each host engaged in internet communication and in each gateway that interconnects networks. These modules share common rules for interpreting address fields and for fragmenting and assembling internet datagrams. In addition, these modules (especially in gateways) have procedures for making routing decisions and other functions.

The internet protocol treats each internet datagram as an independent entity unrelated to any other internet datagram. There are no connections or logical circuits (virtual or otherwise).

The internet protocol uses four key mechanisms in providing its service: Type of Service, Time to Live, Options, and Header Checksum.

The Type of Service is used to indicate the quality of the service desired. The type of service is an abstract or generalized set of parameters which characterize the service choices provided in the networks that make up the internet. This type of service indication is to be used by gateways to select the actual transmission parameters for a particular network, the network to be used for the next hop, or the next gateway when routing an internet datagram.

The Time to Live is an indication of an upper bound on the lifetime of an internet datagram. It is set by the sender of the datagram and reduced at the points along the route where it is processed. If the time to live reaches zero before the internet datagram reaches its destination, the internet datagram is destroyed. The time to live can be thought of as a self destruct time limit.

The Options provide for control functions needed or useful in some situations but unnecessary for the most common communications. The options include provisions for timestamps, security, and special routing.

The Header Checksum provides a verification that the information used in processing internet datagram has been transmitted correctly. The data may contain errors. If the header checksum fails, the internet datagram is discarded at once by the entity which detects the error.

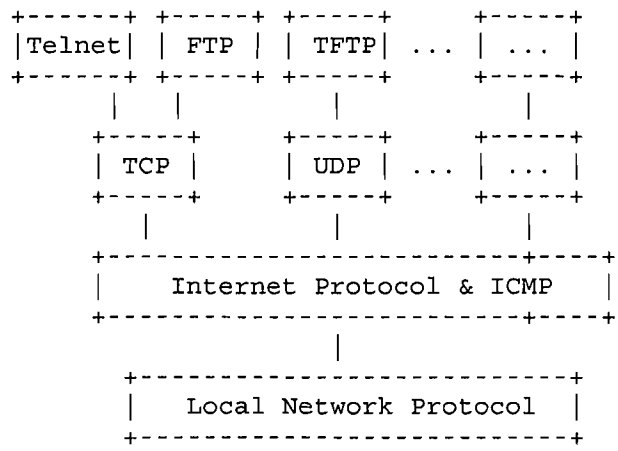
The internet protocol does not provide a reliable communication facility. There are no acknowledgments either end-to-end or hop-by-hop. There is no error control for data, only a header checksum. There are no retransmissions. There is no flow control.

Errors detected may be reported via the Internet Control Message Protocol (ICMP) [3] which is implemented in the internet protocol module.

2. OVERVIEW

2.1. Relation to Other Protocols

The following diagram illustrates the place of the internet protocol in the protocol hierarchy:



Protocol Relationships

Figure 1.

Internet protocol interfaces on one side to the higher level host-to-host protocols and on the other side to the local network protocol. In this context a "local network" may be a small network in a building or a large network such as the ARPANET.

2.2. Model of Operation

The model of operation for transmitting a datagram from one application program to another is illustrated by the following scenario:

We suppose that this transmission will involve one intermediate gateway.

The sending application program prepares its data and calls on its local internet module to send that data as a datagram and passes the destination address and other parameters as arguments of the call.

The internet module prepares a datagram header and attaches the data to it. The internet module determines a local network address for this internet address, in this case it is the address of a gateway.

It sends this datagram and the local network address to the local network interface.

The local network interface creates a local network header, and attaches the datagram to it, then sends the result via the local network.

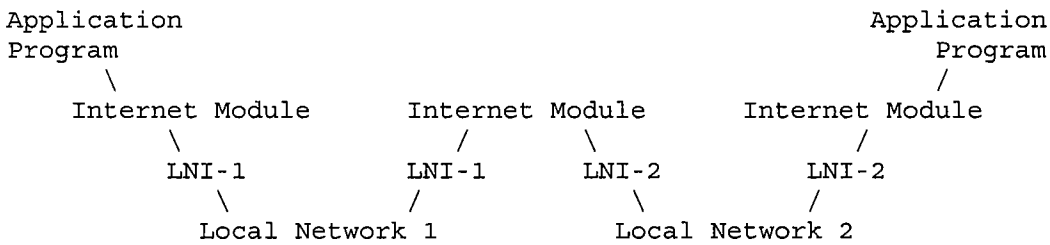
The datagram arrives at a gateway host wrapped in the local network

header, the local network interface strips off this header, and turns the datagram over to the internet module. The internet module determines from the internet address that the datagram is to be forwarded to another host in a second network. The internet module determines a local net address for the destination host. It calls on the local network interface for that network to send the datagram.

This local network interface creates a local network header and attaches the datagram sending the result to the destination host.

At this destination host the datagram is stripped of the local net header by the local network interface and handed to the internet module.

The internet module determines that the datagram is for an application program in this host. It passes the data to the application program in response to a system call, passing the source address and other parameters as results of the call.



Transmission Path

Figure 2

2.3. Function Description

The function or purpose of Internet Protocol is to move datagrams through an interconnected set of networks. This is done by passing the datagrams from one internet module to another until the destination is reached. The internet modules reside in hosts and gateways in the internet system. The datagrams are routed from one internet module to another through individual networks based on the interpretation of an internet address. Thus, one important mechanism

of the internet protocol is the internet address.

In the routing of messages from one internet module to another, datagrams may need to traverse a network whose maximum packet size is smaller than the size of the datagram. To overcome this difficulty, a fragmentation mechanism is provided in the internet protocol.

Addressing

A distinction is made between names, addresses, and routes [4]. A name indicates what we seek. An address indicates where it is. A route indicates how to get there. The internet protocol deals primarily with addresses. It is the task of higher level (i.e., host-to-host or application) protocols to make the mapping from names to addresses. The internet module maps internet addresses to local net addresses. It is the task of lower level (i.e., local net or gateways) procedures to make the mapping from local net addresses to routes.

Addresses are fixed length of four octets (32 bits). An address begins with a network number, followed by local address (called the "rest" field). There are three formats or classes of internet addresses: in class a, the high order bit is zero, the next 7 bits are the network, and the last 24 bits are the local address; in class b, the high order two bits are one-zero, the next 14 bits are the network and the last 16 bits are the local address; in class c, the high order three bits are one-one-zero, the next 21 bits are the network and the last 8 bits are the local address.

Care must be taken in mapping internet addresses to local net addresses; a single physical host must be able to act as if it were several distinct hosts to the extent of using several distinct internet addresses. Some hosts will also have several physical interfaces (multi-homing).

That is, provision must be made for a host to have several physical interfaces to the network with each having several logical internet addresses.

Examples of address mappings may be found in "Address Mappings" [5].

Fragmentation

Fragmentation of an internet datagram is necessary when it originates in a local net that allows a large packet size and must traverse a local net that limits packets to a smaller size to reach its destination.

An internet datagram can be marked "don't fragment." Any internet

datagram so marked is not to be internet fragmented under any circumstances. If internet datagram marked don't fragment cannot be delivered to its destination without fragmenting it, it is to be discarded instead.

Fragmentation, transmission and reassembly across a local network which is invisible to the internet protocol module is called intranet fragmentation and may be used [6].

The internet fragmentation and reassembly procedure needs to be able to break a datagram into an almost arbitrary number of pieces that can be later reassembled. The receiver of the fragments uses the identification field to ensure that fragments of different datagrams are not mixed. The fragment offset field tells the receiver the position of a fragment in the original datagram. The fragment offset and length determine the portion of the original datagram covered by this fragment. The more-fragments flag indicates (by being reset) the last fragment. These fields provide sufficient information to reassemble datagrams.

The identification field is used to distinguish the fragments of one datagram from those of another. The originating protocol module of an internet datagram sets the identification field to a value that must be unique for that source-destination pair and protocol for the time the datagram will be active in the internet system. The originating protocol module of a complete datagram sets the more-fragments flag to zero and the fragment offset to zero.

To fragment a long internet datagram, an internet protocol module (for example, in a gateway), creates two new internet datagrams and copies the contents of the internet header fields from the long datagram into both new internet headers. The data of the long datagram is divided into two portions on a 8 octet (64 bit) boundary (the second portion might not be an integral multiple of 8 octets, but the first must be). Call the number of 8 octet blocks in the first portion NFB (for Number of Fragment Blocks). The first portion of the data is placed in the first new internet datagram, and the total length field is set to the length of the first

datagram. The more-fragments flag is set to one. The second portion of the data is placed in the second new internet datagram, and the total length field is set to the length of the second datagram. The more-fragments flag carries the same value as the long datagram. The fragment offset field of the second new internet datagram is set to the value of that field in the long datagram plus NFB.

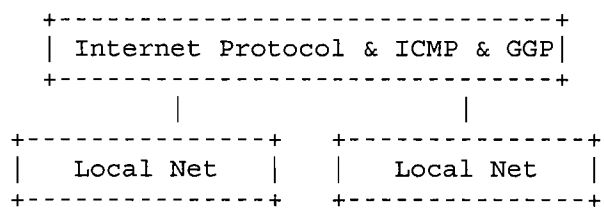
This procedure can be generalized for an n-way split, rather than the two-way split described.

To assemble the fragments of an internet datagram, an internet protocol module (for example at a destination host) combines internet datagrams that all have the same value for the four fields: identification, source, destination, and protocol. The combination is done by placing the data portion of each fragment in the relative position indicated by the fragment offset in that fragment's internet header. The first fragment will have the fragment offset zero, and the last fragment will have the more-fragments flag reset to zero.

2.4. Gateways

Gateways implement internet protocol to forward datagrams between networks. Gateways also implement the Gateway to Gateway Protocol (GGP) [7] to coordinate routing and other internet control information.

In a gateway the higher level protocols need not be implemented and the GGP functions are added to the IP module.



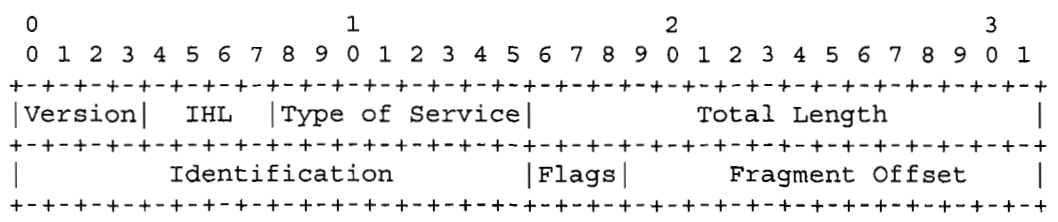
Gateway Protocols

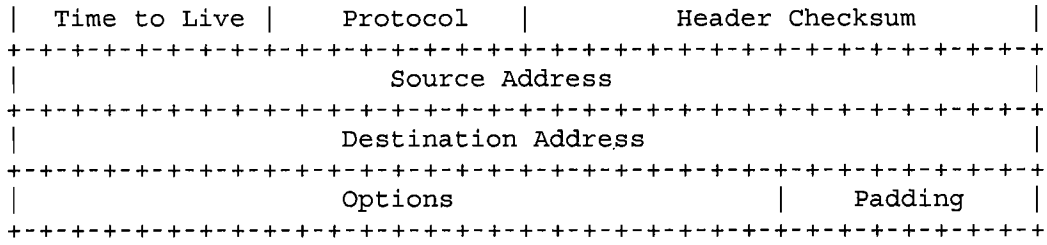
Figure 3.

3. SPECIFICATION

3.1. Internet Header Format

A summary of the contents of the internet header follows:





Example Internet Datagram Header

Figure 4.

Note that each tick mark represents one bit position.

Version: 4 bits

The Version field indicates the format of the internet header. This document describes version 4.

IHL: 4 bits

Internet Header Length is the length of the internet header in 32 bit words, and thus points to the beginning of the data. Note that the minimum value for a correct header is 5.

Type of Service: 8 bits

The Type of Service provides an indication of the abstract parameters of the quality of service desired. These parameters are to be used to guide the selection of the actual service parameters when transmitting a datagram through a particular network. Several networks offer service precedence, which somehow treats high precedence traffic as more important than other traffic (generally by accepting only traffic above a certain precedence at time of high load). The major choice is a three way tradeoff between low-delay, high-reliability, and high-throughput.

Bits 0-2: Precedence.
Bit 3: 0 = Normal Delay, 1 = Low Delay.

```
Bits   4:  0 = Normal Throughput, 1 = High Throughput.
Bits   5:  0 = Normal Relibility, 1 = High Relibility.
Bit  6-7:  Reserved for Future Use.

   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|                 |     |     |     |     |     |
|   PRECEDENCE    |  D  |  T  |  R  |  0  |  0  |
|                 |     |     |     |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

Precedence

111 - Network Control
110 - Internetwork Control
101 - CRITIC/ECP
100 - Flash Override
011 - Flash
010 - Immediate
001 - Priority
000 - Routine

The use of the Delay, Throughput, and Reliability indications may increase the cost (in some sense) of the service. In many networks better performance for one of these parameters is coupled with worse performance on another. Except for very unusual cases at most two of these three indications should be set.

The type of service is used to specify the treatment of the datagram during its transmission through the internet system. Example mappings of the internet type of service to the actual service provided on networks such as AUTODIN II, ARPANET, SATNET, and PRNET is given in "Service Mappings" [8].

The Network Control precedence designation is intended to be used within a network only. The actual use and control of that designation is up to each network. The Internetwork Control designation is intended for use by gateway control originators only. If the actual use of these precedence designations is of concern to a particular network, it is the responsibility of that network to control the access to, and use of, those precedence designations.

Total Length: 16 bits

Total Length is the length of the datagram, measured in octets, including internet header and data. This field allows the length of a datagram to be up to 65,535 octets. Such long datagrams are impractical for most hosts and networks. All hosts must be prepared to accept datagrams of up to 576 octets (whether they arrive whole

or in fragments). It is recommended that hosts only send datagrams larger than 576 octets if they have assurance that the destination is prepared to accept the larger datagrams.

The number 576 is selected to allow a reasonable sized data block to be transmitted in addition to the required header information. For example, this size allows a data block of 512 octets plus 64 header octets to fit in a datagram. The maximal internet header is 60 octets, and a typical internet header is 20 octets, allowing a margin for headers of higher level protocols.

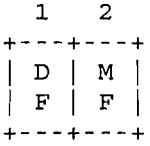
Identification: 16 bits

An identifying value assigned by the sender to aid in assembling the fragments of a datagram.

Flags: 3 bits

Various Control Flags.

Bit 0: reserved, must be zero
Bit 1: (DF) 0 = May Fragment, 1 = Don't Fragment.
Bit 2: (MF) 0 = Last Fragment, 1 = More Fragments.



Fragment Offset: 13 bits

This field indicates where in the datagram this fragment belongs.

The fragment offset is measured in units of 8 octets (64 bits). The first fragment has offset zero.

Time to Live: 8 bits

This field indicates the maximum time the datagram is allowed to remain in the internet system. If this field contains the value zero, then the datagram must be destroyed. This field is modified in internet header processing. The time is measured in units of seconds, but since every module that processes a datagram must decrease the TTL by at least one even if it process the datagram in less than a second, the TTL must be thought of only as an upper bound on the time a datagram may exist. The intention is to cause undeliverable datagrams to be discarded, and to bound the maximum datagram lifetime.

Protocol: 8 bits

This field indicates the next level protocol used in the data portion of the internet datagram. The values for various protocols are specified in "Assigned Numbers" [9].

Header Checksum: 16 bits

A checksum on the header only. Since some header fields change (e.g., time to live), this is recomputed and verified at each point that the internet header is processed.

The checksum algorithm is:

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.

This is a simple to compute checksum and experimental evidence indicates it is adequate, but it is provisional and may be replaced by a CRC procedure, depending on further experience.

Source Address: 32 bits

The source address. See section 3.2.

Destination Address: 32 bits

The destination address. See section 3.2.

Options: variable

The options may appear or not in datagrams. They must be implemented by all IP modules (host and gateways). What is optional is their transmission in any particular datagram, not their implementation.

In some environments the security option may be required in all datagrams.

The option field is variable in length. There may be zero or more options. There are two cases for the format of an option:

Case 1: A single octet of option-type.

Case 2: An option-type octet, an option-length octet, and the actual option-data octets.

The option-length octet counts the option-type octet and the option-length octet as well as the option-data octets.

The option-type octet is viewed as having 3 fields:

1 bit copied flag,
2 bits option class,
5 bits option number.

The copied flag indicates that this option is copied into all fragments on fragmentation.

0 = not copied
1 = copied

The option classes are:

0 = control
1 = reserved for future use
2 = debugging and measurement
3 = reserved for future use

The following internet options are defined:

CLASS	NUMBER	LENGTH	DESCRIPTION
0	0	-	End of Option list. This option occupies only 1 octet; it has no length octet.
0	1	-	No Operation. This option occupies only 1 octet; it has no length octet.
0	2	11	Security. Used to carry Security, Compartmentation, User Group (TCC), and Handling Restriction Codes compatible with DOD requirements.
0	3	var.	Loose Source Routing. Used to route the internet datagram based on information supplied by the source.
0	9	var.	Strict Source Routing. Used to route the internet datagram based on information supplied by the source.

0	7	var.	Record Route. Used to trace the route an internet datagram takes.
0	8	4	Stream ID. Used to carry the stream identifier.
2	4	var.	Internet Timestamp.

Specific Option Definitions

End of Option List

```
+--------+
|00000000|
+--------+
  Type=0
```

This option indicates the end of the option list. This might not coincide with the end of the internet header according to the internet header length. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the internet header.

May be copied, introduced, or deleted on fragmentation, or for any other reason.

No Operation

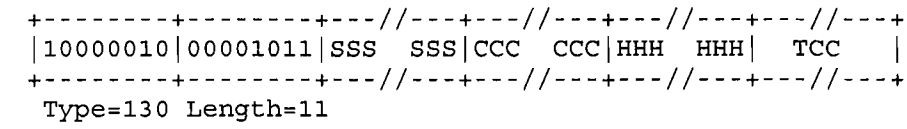
```
+--------+
|00000001|
+--------+
  Type=1
```

This option may be used between options, for example, to align the beginning of a subsequent option on a 32 bit boundary.

May be copied, introduced, or deleted on fragmentation, or for any other reason.

Security

This option provides a way for hosts to send security, compartmentation, handling restrictions, and TCC (closed user group) parameters. The format for this option is as follows:



Security (S field): 16 bits

Specifies one of 16 levels of security (eight of which are reserved for future use).

00000000 00000000 - Unclassified
11110001 00110101 - Confidential
01111000 10011010 - EFTO
10111100 01001101 - MMMM
01011110 00100110 - PROG
10101111 00010011 - Restricted
11010111 10001000 - Secret
01101011 11000101 - Top Secret
00110101 11100010 - (Reserved for future use)
10011010 11110001 - (Reserved for future use)
01001101 01111000 - (Reserved for future use)
00100100 10111101 - (Reserved for future use)
00010011 01011110 - (Reserved for future use)
10001001 10101111 - (Reserved for future use)
11000100 11010110 - (Reserved for future use)
11100010 01101011 - (Reserved for future use)

Compartments (C field): 16 bits

An all zero value is used when the information transmitted is not compartmented. Other values for the compartments field may be obtained from the Defense Intelligence Agency.

Handling Restrictions (H field): 16 bits

The values for the control and release markings are alphanumeric digraphs and are defined in the Defense Intelligence Agency Manual DIAM 65-19, "Standard Security Markings".

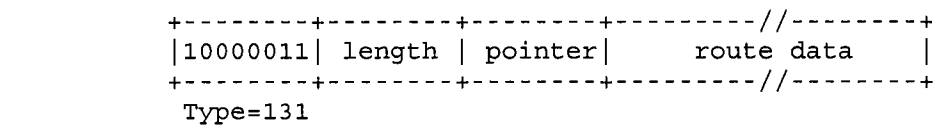
Transmission Control Code (TCC field): 24 bits

Provides a means to segregate traffic and define controlled communities of interest among subscribers. The TCC values are trigraphs, and are available from HQ DCA Code 530.

Must be copied on fragmentation. This option appears at most

once in a datagram.

Loose Source and Record Route



The loose source and record route (LSRR) option provides a means for the source of an internet datagram to supply routing information to be used by the gateways in forwarding the datagram to the destination, and to record the route information.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next source address to be processed. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A route data is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is greater than the length, the source route is empty (and the recorded route full) and the routing is to be based on the destination address field.

If the address in destination address field has been reached and the pointer is not greater than the length, the next address in the source route replaces the address in the destination address field, and the recorded route address replaces the source address just used, and pointer is increased by four.

The recorded route address is the internet module's own internet address as known in the environment into which this datagram is being forwarded.

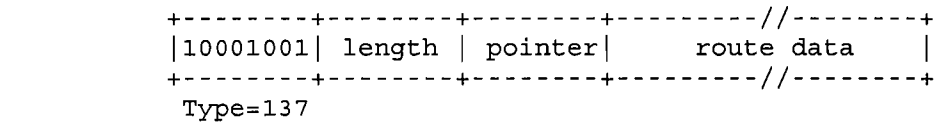
This procedure of replacing the source route with the recorded route (though it is in the reverse of the order it must be in to be used as a source route) means the option (and the IP header as a whole) remains a constant length as the datagram progresses through the internet.

This option is a loose source route because the gateway or host IP is allowed to use any route of any number of other intermediate gateways to reach the next address in the route.

Must be copied on fragmentation. Appears at most once in a

datagram.

Strict Source and Record Route



The strict source and record route (SSRR) option provides a means for the source of an internet datagram to supply routing information to be used by the gateways in forwarding the datagram to the destination, and to record the route information.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next source address to be processed. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A route data is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is greater than the length, the source route is empty (and the

recorded route full) and the routing is to be based on the destination address field.

If the address in destination address field has been reached and the pointer is not greater than the length, the next address in the source route replaces the address in the destination address field, and the recorded route address replaces the source address just used, and pointer is increased by four.

The recorded route address is the internet module's own internet address as known in the environment into which this datagram is being forwarded.

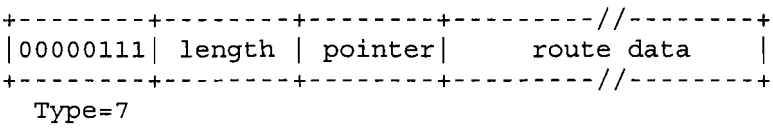
This procedure of replacing the source route with the recorded route (though it is in the reverse of the order it must be in to be used as a source route) means the option (and the IP header as a whole) remains a constant length as the datagram progresses through the internet.

This option is a strict source route because the gateway or host IP must send the datagram directly to the next address in the source route through only the directly connected network

indicated in the next address to reach the next gateway or host specified in the route.

Must be copied on fragmentation. Appears at most once in a datagram.

Record Route



The record route option provides a means to record the route of an internet datagram.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next area to store a route address. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A recorded route is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is

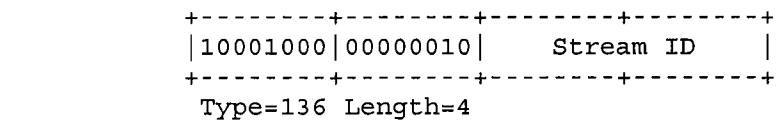
greater than the length, the recorded route data area is full. The originating host must compose this option with a large enough route data area to hold all the address expected. The size of the option does not change due to adding addresses. The intitial contents of the route data area must be zero.

When an internet module routes a datagram it checks to see if the record route option is present. If it is, it inserts its own internet address as known in the environment into which this datagram is being forwarded into the recorded route begining at the octet indicated by the pointer, and increments the pointer by four.

If the route data area is already full (the pointer exceeds the length) the datagram is forwarded without inserting the address into the recorded route. If there is some room but not enough room for a full address to be inserted, the original datagram is considered to be in error and is discarded. In either case an ICMP parameter problem message may be sent to the source host [3].

Not copied on fragmentation, goes in first fragment only. Appears at most once in a datagram.

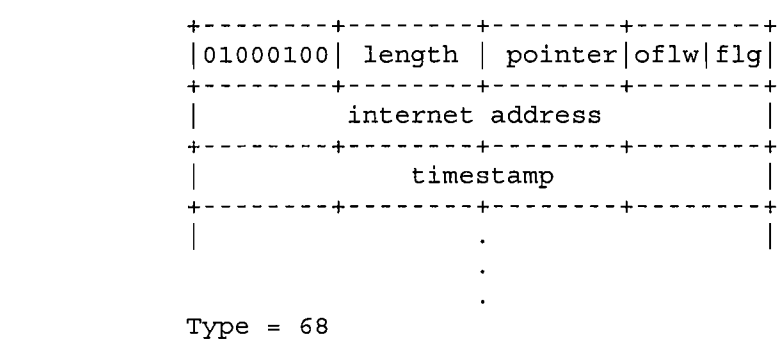
Stream Identifier



This option provides a way for the 16-bit SATNET stream identifier to be carried through networks that do not support the stream concept.

Must be copied on fragmentation. Appears at most once in a datagram.

Internet Timestamp



Type = 68

The Option Length is the number of octets in the option counting the type, length, pointer, and overflow/flag octets (maximum length 40).

The Pointer is the number of octets from the beginning of this option to the end of timestamps plus one (i.e., it points to the octet beginning the space for next timestamp). The smallest legal value is 5. The timestamp area is full when the pointer is greater than the length.

The Overflow (oflw) [4 bits] is the number of IP modules that cannot register timestamps due to lack of space.

The Flag (flg) [4 bits] values are

0 -- time stamps only, stored in consecutive 32-bit words,

1 -- each timestamp is preceded with internet address of the registering entity,

3 -- the internet address fields are prespecified. An IP module only registers its timestamp if it matches its own address with the next specified internet address.

The Timestamp is a right-justified, 32-bit timestamp in milliseconds since midnight UT. If the time is not available in milliseconds or cannot be provided with respect to midnight UT then any time may be inserted as a timestamp provided the high order bit of the timestamp field is set to one to indicate the use of a non-standard value.

The originating host must compose this option with a large enough timestamp data area to hold all the timestamp information expected. The size of the option does not change due to adding

timestamps. The intitial contents of the timestamp data area must be zero or internet address/zero pairs.

If the timestamp data area is already full (the pointer exceeds the length) the datagram is forwarded without inserting the timestamp, but the overflow count is incremented by one.

If there is some room but not enough room for a full timestamp to be inserted, or the overflow count itself overflows, the original datagram is considered to be in error and is discarded. In either case an ICMP parameter problem message may be sent to the source host [3].

The timestamp option is not copied upon fragmentation. It is carried in the first fragment. Appears at most once in a datagram.

Padding: variable

The internet header padding is used to ensure that the internet header ends on a 32 bit boundary. The padding is zero.

3.2. Discussion

The implementation of a protocol must be robust. Each implementation

must expect to interoperate with others created by different individuals. While the goal of this specification is to be explicit about the protocol there is the possibility of differing interpretations. In general, an implementation must be conservative in its sending behavior, and liberal in its receiving behavior. That is, it must be careful to send well-formed datagrams, but must accept any datagram that it can interpret (e.g., not object to technical errors where the meaning is still clear).

The basic internet service is datagram oriented and provides for the fragmentation of datagrams at gateways, with reassembly taking place at the destination internet protocol module in the destination host. Of course, fragmentation and reassembly of datagrams within a network or by private agreement between the gateways of a network is also allowed since this is transparent to the internet protocols and the higher-level protocols. This transparent type of fragmentation and reassembly is termed "network-dependent" (or intranet) fragmentation and is not discussed further here.

Internet addresses distinguish sources and destinations to the host level and provide a protocol field as well. It is assumed that each protocol will provide for whatever multiplexing is necessary within a host.

Addressing

To provide for flexibility in assigning address to networks and allow for the large number of small to intermediate sized networks the interpretation of the address field is coded to specify a small number of networks with a large number of host, a moderate number of networks with a moderate number of hosts, and a large number of networks with a small number of hosts. In addition there is an escape code for extended addressing mode.

Address Formats:

High Order Bits	Format	Class
0	7 bits of net, 24 bits of host	a
10	14 bits of net, 16 bits of host	b
110	21 bits of net, 8 bits of host	c
111	escape to extended addressing mode	

A value of zero in the network field means this network. This is only used in certain ICMP messages. The extended addressing mode is undefined. Both of these features are reserved for future use.

The actual values assigned for network addresses is given in "Assigned Numbers" [9].

The local address, assigned by the local network, must allow for a single physical host to act as several distinct internet hosts. That is, there must be a mapping between internet host addresses and network/host interfaces that allows several internet addresses to correspond to one interface. It must also be allowed for a host to have several physical interfaces and to treat the datagrams from several of them as if they were all addressed to a single host.

Address mappings between internet addresses and addresses for ARPANET, SATNET, PRNET, and other networks are described in "Address Mappings" [5].

Fragmentation and Reassembly.

The internet identification field (ID) is used together with the source and destination address, and the protocol fields, to identify datagram fragments for reassembly.

The More Fragments flag bit (MF) is set if the datagram is not the last fragment. The Fragment Offset field identifies the fragment location, relative to the beginning of the original unfragmented datagram. Fragments are counted in units of 8 octets. The

fragmentation strategy is designed so than an unfragmented datagram has all zero fragmentation information (MF = 0, fragment offset = 0). If an internet datagram is fragmented, its data portion must be broken on 8 octet boundaries.

This format allows 2**13 = 8192 fragments of 8 octets each for a total of 65,536 octets. Note that this is consistent with the the datagram total length field (of course, the header is counted in the total length and not in the fragments).

When fragmentation occurs, some options are copied, but others remain with the first fragment only.

Every internet module must be able to forward a datagram of 68 octets without further fragmentation. This is because an internet header may be up to 60 octets, and the minimum fragment is 8 octets.

Every internet destination must be able to receive a datagram of 576 octets either in one piece or in fragments to be reassembled.

The fields which may be affected by fragmentation include:

(1) options field
(2) more fragments flag
(3) fragment offset
(4) internet header length field
(5) total length field

(6) header checksum

If the Don't Fragment flag (DF) bit is set, then internet fragmentation of this datagram is NOT permitted, although it may be discarded. This can be used to prohibit fragmentation in cases where the receiving host does not have sufficient resources to reassemble internet fragments.

One example of use of the Don't Fragment feature is to down line load a small host. A small host could have a boot strap program that accepts a datagram stores it in memory and then executes it.

The fragmentation and reassembly procedures are most easily described by examples. The following procedures are example implementations.

General notation in the following pseudo programs: "=<" means "less than or equal", "#" means "not equal", "=" means "equal", "<-" means "is set to". Also, "x to y" includes x and excludes y; for example, "4 to 7" would include 4, 5, and 6 (but not 7).

An Example Fragmentation Procedure

The maximum sized datagram that can be transmitted through the next network is called the maximum transmission unit (MTU).

If the total length is less than or equal the maximum transmission unit then submit this datagram to the next step in datagram processing; otherwise cut the datagram into two fragments, the first fragment being the maximum size, and the second fragment being the rest of the datagram. The first fragment is submitted to the next step in datagram processing, while the second fragment is submitted to this procedure in case it is still too large.

Notation:

FO	-	Fragment Offset
IHL	-	Internet Header Length
DF	-	Don't Fragment flag
MF	-	More Fragments flag
TL	-	Total Length
OFO	-	Old Fragment Offset
OIHL	-	Old Internet Header Length
OMF	-	Old More Fragments flag
OTL	-	Old Total Length
NFB	-	Number of Fragment Blocks
MTU	-	Maximum Transmission Unit

Procedure:

```
   IF TL =< MTU THEN Submit this datagram to the next step
        in datagram processing ELSE IF DF = 1 THEN discard the
   datagram ELSE
   To produce the first fragment:
   (1)  Copy the original internet header;
   (2)  OIHL <- IHL; OTL <- TL; OFO <- FO; OMF <- MF;
   (3)  NFB <- (MTU-IHL*4)/8;
   (4)  Attach the first NFB*8 data octets;
   (5)  Correct the header:
        MF <- 1;  TL <- (IHL*4)+(NFB*8);
        Recompute Checksum;
   (6)  Submit this fragment to the next step in
        datagram processing;
   To produce the second fragment:
   (7)  Selectively copy the internet header (some options
        are not copied, see option definitions);
   (8)  Append the remaining data;
   (9)  Correct the header:
        IHL <- (((OIHL*4)-(length of options not copied))+3)/4;
```



```
        TL <- OTL - NFB*8 - (OIHL-IHL)*4);
        FO <- OFO + NFB;  MF <- OMF;  Recompute Checksum;
   (10) Submit this fragment to the fragmentation test; DONE.
```

In the above procedure each fragment (except the last) was made the maximum allowable size. An alternative might produce less than the maximum size datagrams. For example, one could implement a fragmentation procedure that repeatly divided large datagrams in half until the resulting fragments were less than the maximum transmission unit size.

An Example Reassembly Procedure

For each datagram the buffer identifier is computed as the concatenation of the source, destination, protocol, and identification fields. If this is a whole datagram (that is both the fragment offset and the more fragments fields are zero), then any reassembly resources associated with this buffer identifier are released and the datagram is forwarded to the next step in datagram processing.

If no other fragment with this buffer identifier is on hand then reassembly resources are allocated. The reassembly resources consist of a data buffer, a header buffer, a fragment block bit table, a total data length field, and a timer. The data from the fragment is placed in the data buffer according to its fragment offset and length, and bits are set in the fragment block bit table corresponding to the fragment blocks received.

If this is the first fragment (that is the fragment offset is zero) this header is placed in the header buffer. If this is the last fragment (that is the more fragments field is zero) the total data length is computed. If this fragment completes the datagram (tested by checking the bits set in the fragment block table), then the datagram is sent to the next step in datagram processing; otherwise the timer is set to the maximum of the current timer value and the value of the time to live field from this fragment; and the reassembly routine gives up control.

If the timer runs out, the all reassembly resources for this buffer identifier are released. The initial setting of the timer is a lower bound on the reassembly waiting time. This is because the waiting time will be increased if the Time to Live in the arriving fragment is greater than the current timer value but will not be decreased if it is less. The maximum this timer value could reach is the maximum time to live (approximately 4.25 minutes). The current recommendation for the initial timer setting is 15 seconds. This may be changed as experience with

this protocol accumulates. Note that the choice of this parameter value is related to the buffer capacity available and the data rate of the transmission medium; that is, data rate times timer value equals buffer size (e.g., 10Kb/s X 15s = 150Kb).

Notation:

```
  FO    -  Fragment Offset
  IHL   -  Internet Header Length
  MF    -  More Fragments flag
  TTL   -  Time To Live
  NFB   -  Number of Fragment Blocks
  TL    -  Total Length
  TDL   -  Total Data Length
  BUFID -  Buffer Identifier
  RCVBT -  Fragment Received Bit Table
  TLB   -  Timer Lower Bound
```

Procedure:

```
  (1)  BUFID <- source|destination|protocol|identification;
  (2)  IF FO = 0 AND MF = 0
  (3)     THEN IF buffer with BUFID is allocated
  (4)             THEN flush all reassembly for this BUFID;
  (5)          Submit datagram to next step; DONE.
  (6)     ELSE IF no buffer with BUFID is allocated
  (7)             THEN allocate reassembly resources
                       with BUFID;
                       TIMER <- TLB; TDL <- 0;
  (8)          put data from fragment into data buffer with
```

```
                BUFID from octet FO*8 to
                                    octet (TL-(IHL*4))+FO*8;
(9)             set RCVBT bits from FO
                                   to FO+((TL-(IHL*4)+7)/8);
(10)            IF MF = 0 THEN TDL <- TL-(IHL*4)+(FO*8)
(11)            IF FO = 0 THEN put header in header buffer
(12)            IF TDL # 0
(13)             AND all RCVBT bits from 0
                                        to (TDL+7)/8 are set
(14)               THEN TL <- TDL+(IHL*4)
(15)                    Submit datagram to next step;
(16)                    free all reassembly resources
                        for this BUFID; DONE.
(17)            TIMER <- MAX(TIMER,TTL);
(18)            give up until next fragment or timer expires;
(19) timer expires: flush all reassembly with this BUFID; DONE.
```

In the case that two or more fragments contain the same data

either identically or through a partial overlap, this procedure will use the more recently arrived copy in the data buffer and datagram delivered.

Identification

The choice of the Identifier for a datagram is based on the need to provide a way to uniquely identify the fragments of a particular datagram. The protocol module assembling fragments judges fragments to belong to the same datagram if they have the same source, destination, protocol, and Identifier. Thus, the sender must choose the Identifier to be unique for this source, destination pair and protocol for the time the datagram (or any fragment of it) could be alive in the internet.

It seems then that a sending protocol module needs to keep a table of Identifiers, one entry for each destination it has communicated with in the last maximum packet lifetime for the internet.

However, since the Identifier field allows 65,536 different values, some host may be able to simply use unique identifiers independent of destination.

It is appropriate for some higher level protocols to choose the identifier. For example, TCP protocol modules may retransmit an identical TCP segment, and the probability for correct reception would be enhanced if the retransmission carried the same identifier as the original transmission since fragments of either datagram could be used to construct a correct TCP segment.

Type of Service

The type of service (TOS) is for internet service quality selection. The type of service is specified along the abstract parameters precedence, delay, throughput, and reliability. These abstract parameters are to be mapped into the actual service parameters of the particular networks the datagram traverses.

Precedence. An independent measure of the importance of this datagram.

Delay. Prompt delivery is important for datagrams with this indication.

Throughput. High data rate is important for datagrams with this indication.

Reliability. A higher level of effort to ensure delivery is important for datagrams with this indication.

For example, the ARPANET has a priority bit, and a choice between "standard" messages (type 0) and "uncontrolled" messages (type 3), (the choice between single packet and multipacket messages can also be considered a service parameter). The uncontrolled messages tend to be less reliably delivered and suffer less delay. Suppose an internet datagram is to be sent through the ARPANET. Let the internet type of service be given as:

Precedence:	5
Delay:	0
Throughput:	1
Reliability:	1

In this example, the mapping of these parameters to those available for the ARPANET would be to set the ARPANET priority bit on since the Internet precedence is in the upper half of its range, to select standard messages since the throughput and reliability requirements are indicated and delay is not. More details are given on service mappings in "Service Mappings" [8].

Time to Live

The time to live is set by the sender to the maximum time the datagram is allowed to be in the internet system. If the datagram is in the internet system longer than the time to live, then the datagram must be destroyed.

This field must be decreased at each point that the internet header is processed to reflect the time spent processing the datagram.

Even if no local information is available on the time actually spent, the field must be decremented by 1. The time is measured in units of seconds (i.e. the value 1 means one second). Thus, the maximum time to live is 255 seconds or 4.25 minutes. Since every module that processes a datagram must decrease the TTL by at least one even if it process the datagram in less than a second, the TTL must be thought of only as an upper bound on the time a datagram may exist. The intention is to cause undeliverable datagrams to be discarded, and to bound the maximum datagram lifetime.

Some higher level reliable connection protocols are based on assumptions that old duplicate datagrams will not arrive after a certain time elapses. The TTL is a way for such protocols to have an assurance that their assumption is met.

Options

The options are optional in each datagram, but required in implementations. That is, the presence or absence of an option is the choice of the sender, but each internet module must be able to parse every option. There can be several options present in the option field.

The options might not end on a 32-bit boundary. The internet header must be filled out with octets of zeros. The first of these would be interpreted as the end-of-options option, and the remainder as internet header padding.

Every internet module must be able to act on every option. The Security Option is required if classified, restricted, or compartmented traffic is to be passed.

Checksum

The internet header checksum is recomputed if the internet header is changed. For example, a reduction of the time to live, additions or changes to internet options, or due to fragmentation. This checksum at the internet level is intended to protect the internet header fields from transmission errors.

There are some applications where a few data bit errors are acceptable while retransmission delays are not. If the internet protocol enforced data correctness such applications could not be supported.

Errors

Internet protocol errors may be reported via the ICMP messages [3].

3.3. Interfaces

The functional description of user interfaces to the IP is, at best, fictional, since every operating system will have different facilities. Consequently, we must warn readers that different IP implementations may have different user interfaces. However, all IPs must provide a certain minimum set of services to guarantee that all IP implementations can support the same protocol hierarchy. This section specifies the functional interfaces required of all IP implementations.

Internet protocol interfaces on one side to the local network and on the other side to either a higher level protocol or an application program. In the following, the higher level protocol or application

program (or even a gateway program) will be called the "user" since it is using the internet module. Since internet protocol is a datagram protocol, there is minimal memory or state maintained between datagram transmissions, and each call on the internet protocol module by the user supplies all information necessary for the IP to perform the service requested.

An Example Upper Level Interface

The following two example calls satisfy the requirements for the user to internet protocol module communication ("=>" means returns):

```
SEND (src, dst, prot, TOS, TTL, BufPTR, len, Id, DF, opt => result)

  where:

    src = source address
    dst = destination address
    prot = protocol
    TOS = type of service
    TTL = time to live
    BufPTR = buffer pointer
    len = length of buffer
    Id  = Identifier
    DF = Don't Fragment
    opt = option data
    result = response
      OK = datagram sent ok
      Error = error in arguments or local network error

  Note that the precedence is included in the TOS and the
  security/compartment is passed as an option.

RECV (BufPTR, prot, => result, src, dst, TOS, len, opt)
```

where:

BufPTR = buffer pointer
prot = protocol
result = response
OK = datagram received ok
Error = error in arguments
len = length of buffer
src = source address
dst = destination address
TOS = type of service
opt = option data

When the user sends a datagram, it executes the SEND call supplying all the arguments. The internet protocol module, on receiving this call, checks the arguments and prepares and sends the message. If the arguments are good and the datagram is accepted by the local network, the call returns successfully. If either the arguments are bad, or the datagram is not accepted by the local network, the call returns unsuccessfully. On unsuccessful returns, a reasonable report must be made as to the cause of the problem, but the details of such reports are up to individual implementations.

When a datagram arrives at the internet protocol module from the local network, either there is a pending RECV call from the user addressed or there is not. In the first case, the pending call is satisfied by passing the information from the datagram to the user. In the second case, the user addressed is notified of a pending datagram. If the user addressed does not exist, an ICMP error message is returned to the sender, and the data is discarded.

The notification of a user may be via a pseudo interrupt or similar mechanism, as appropriate in the particular operating system environment of the implementation.

A user's RECV call may then either be immediately satisfied by a pending datagram, or the call may be pending until a datagram arrives.

The source address is included in the send call in case the sending host has several addresses (multiple physical connections or logical addresses). The internet module must check to see that the source address is one of the legal address for this host.

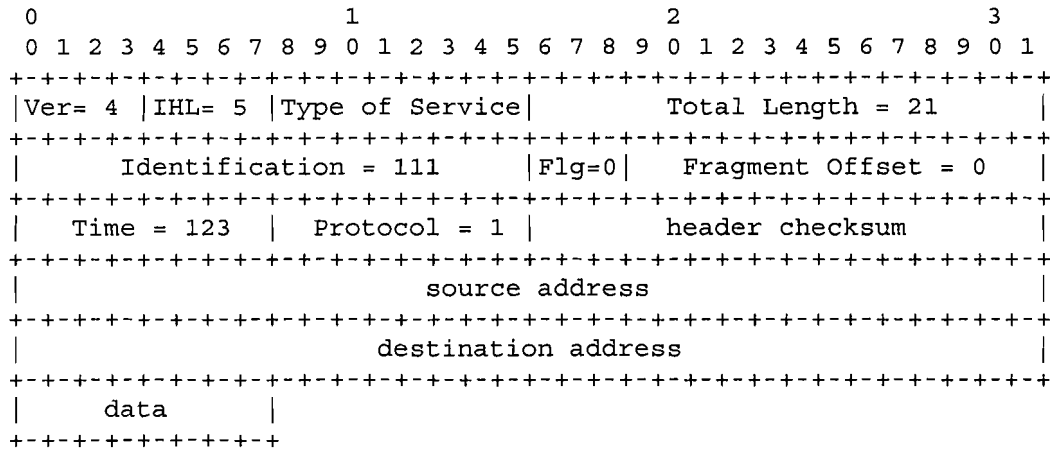
An implementation may also allow or require a call to the internet module to indicate interest in or reserve exclusive use of a class of datagrams (e.g., all those with a certain value in the protocol field).

This section functionally characterizes a USER/IP interface. The notation used is similar to most procedure of function calls in high level languages, but this usage is not meant to rule out trap type service calls (e.g., SVCs, UUOs, EMTs), or any other form of interprocess communication.

APPENDIX A: Examples & Scenarios

Example 1:

This is an example of the minimal data carrying internet datagram:



Example Internet Datagram

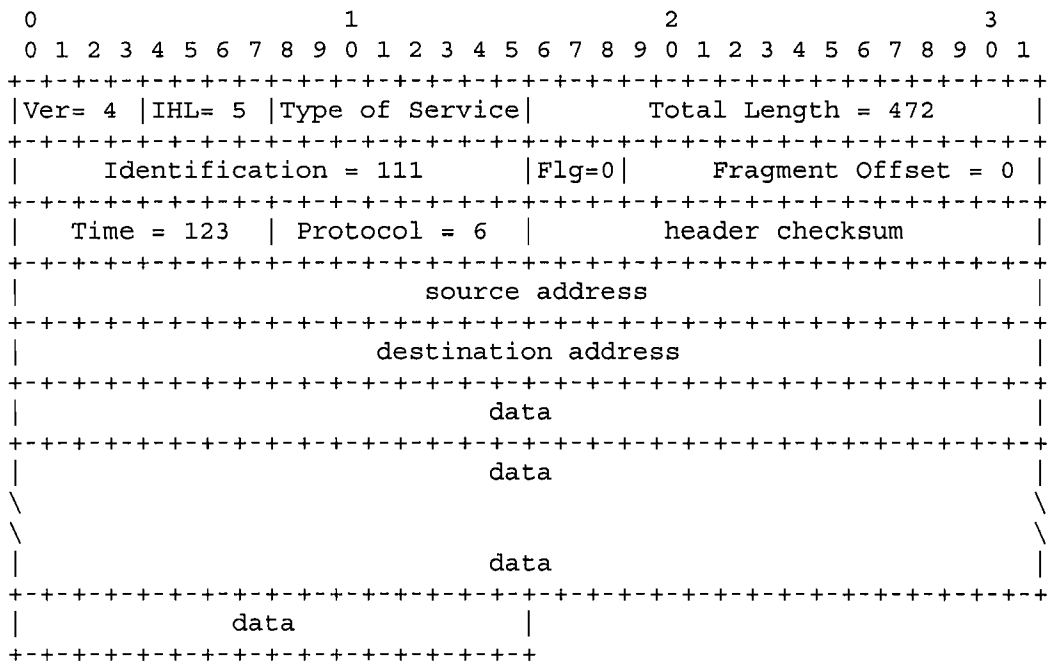
Figure 5.

Note that each tick mark represents one bit position.

This is a internet datagram in version 4 of internet protocol; the internet header consists of five 32 bit words, and the total length of the datagram is 21 octets. This datagram is a complete datagram (not a fragment).

Example 2:

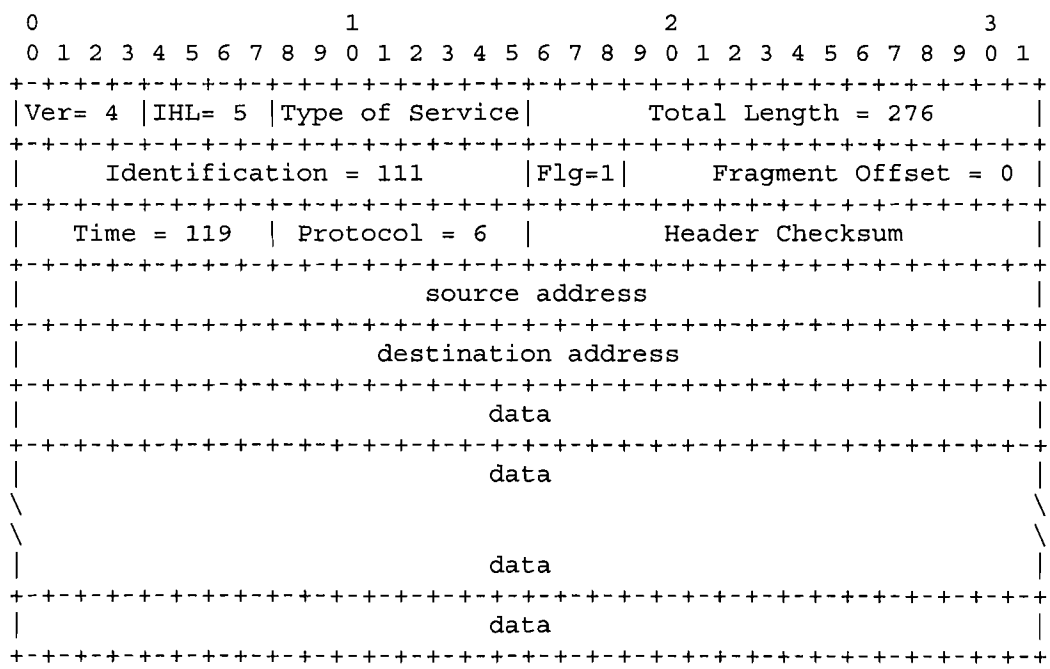
In this example, we show first a moderate size internet datagram (452 data octets), then two internet fragments that might result from the fragmentation of this datagram if the maximum sized transmission allowed were 280 octets.



Example Internet Datagram

Figure 6.

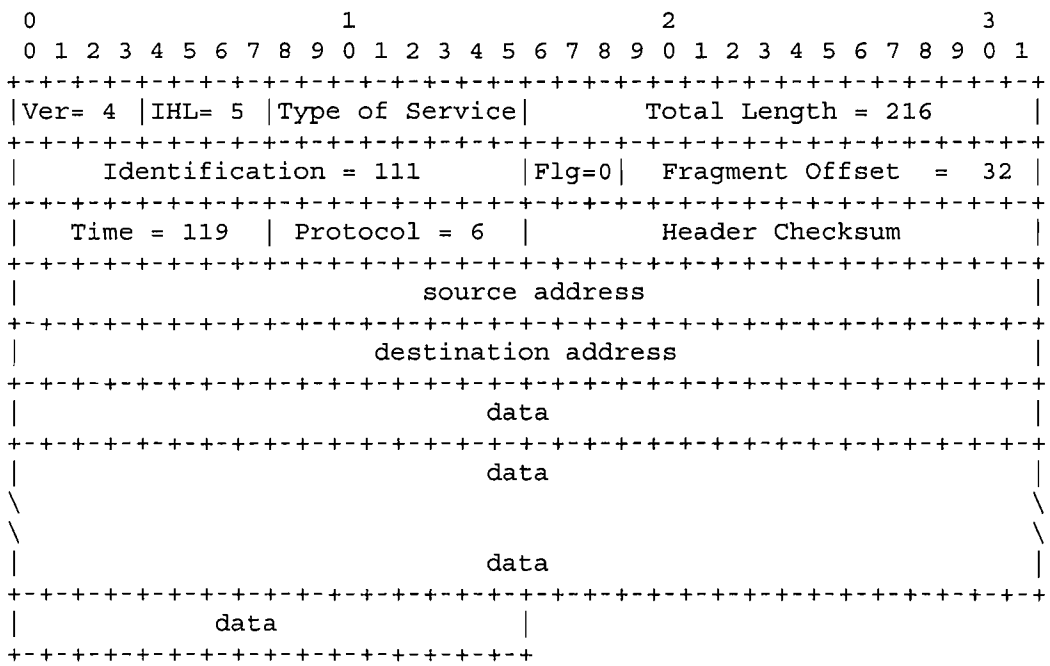
Now the first fragment that results from splitting the datagram after 256 data octets.



Example Internet Fragment

Figure 7.

And the second fragment.

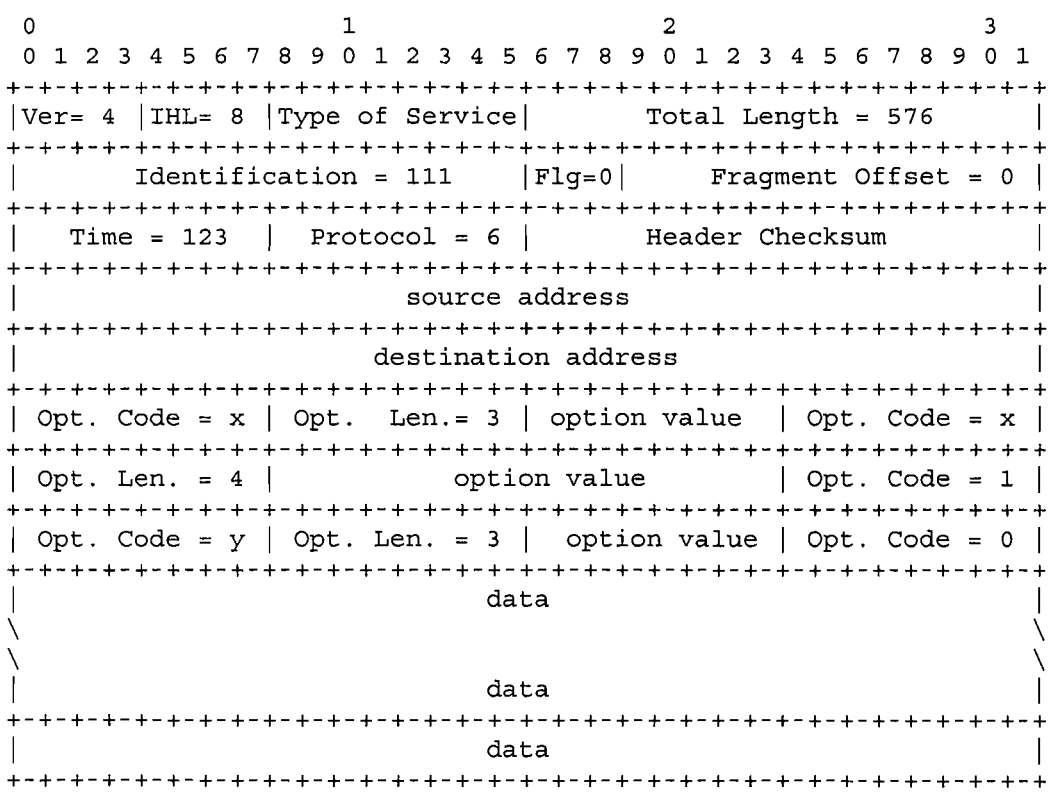


Example Internet Fragment

Figure 8.

Example 3:

Here, we show an example of a datagram containing options:

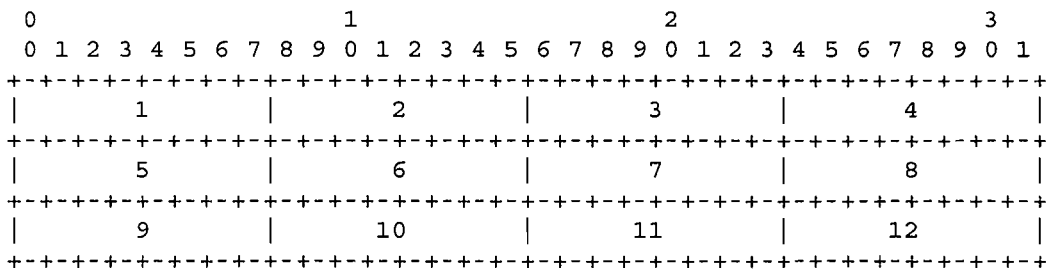


Example Internet Datagram

Figure 9.

APPENDIX B: Data Transmission Order

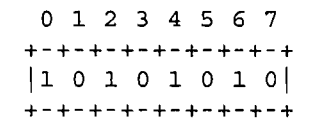
The order of transmission of the header and data described in this document is resolved to the octet level. Whenever a diagram shows a group of octets, the order of transmission of those octets is the normal order in which they are read in English. For example, in the following diagram the octets are transmitted in the order they are numbered.



Transmission Order of Bytes

Figure 10.

Whenever an octet represents a numeric quantity the left most bit in the diagram is the high order or most significant bit. That is, the bit labeled 0 is the most significant bit. For example, the following diagram represents the value 170 (decimal).



Significance of Bits

Figure 11.

Similarly, whenever a multi-octet field represents a numeric quantity the left most bit of the whole field is the most significant bit. When a multi-octet quantity is transmitted the most significant octet is transmitted first.

GLOSSARY

1822
: BBN Report 1822, "The Specification of the Interconnection of a Host and an IMP". The specification of interface between a host and the ARPANET.

ARPANET leader
: The control information on an ARPANET message at the host-IMP interface.

ARPANET message
: The unit of transmission between a host and an IMP in the ARPANET. The maximum size is about 1012 octets (8096 bits).

ARPANET packet
: A unit of transmission used internally in the ARPANET between IMPs. The maximum size is about 126 octets (1008 bits).

Destination
: The destination address, an internet header field.

DF
: The Don't Fragment bit carried in the flags field.

Flags
: An internet header field carrying various control flags.

Fragment Offset
: This internet header field indicates where in the internet datagram a fragment belongs.

GGP
: Gateway to Gateway Protocol, the protocol used primarily between gateways to control routing and other gateway functions.

header
: Control information at the beginning of a message, segment, datagram, packet or block of data.

ICMP

Internet Control Message Protocol, implemented in the internet module, the ICMP is used from gateways to hosts and between hosts to report errors and make routing suggestions.

Identification
An internet header field carrying the identifying value assigned by the sender to aid in assembling the fragments of a datagram.

IHL
The internet header field Internet Header Length is the length of the internet header measured in 32 bit words.

IMP
The Interface Message Processor, the packet switch of the ARPANET.

Internet Address
A four octet (32 bit) source or destination address consisting of a Network field and a Local Address field.

internet datagram
The unit of data exchanged between a pair of internet modules (includes the internet header).

internet fragment
A portion of the data of an internet datagram with an internet header.

Local Address
The address of a host within a network. The actual mapping of an internet local address on to the host addresses in a network is quite general, allowing for many to one mappings.

MF
The More-Fragments Flag carried in the internet header flags field.

module
An implementation, usually in software, of a protocol or other procedure.

more-fragments flag
A flag indicating whether or not this internet datagram contains the end of an internet datagram, carried in the internet header Flags field.

NFB

The Number of Fragment Blocks in a the data portion of an internet fragment. That is, the length of a portion of data measured in 8 octet units.

octet
An eight bit byte.

Options
The internet header Options field may contain several options, and each option may be several octets in length.

Padding
The internet header Padding field is used to ensure that the data begins on 32 bit word boundary. The padding is zero.

Protocol
In this document, the next higher level protocol identifier, an internet header field.

Rest
The local address portion of an Internet Address.

Source
The source address, an internet header field.

TCP
Transmission Control Protocol: A host-to-host protocol for reliable communication in internet environments.

TCP Segment
The unit of data exchanged between TCP modules (including the TCP header).

TFTP
Trivial File Transfer Protocol: A simple file transfer protocol built on UDP.

Time to Live
An internet header field which indicates the upper bound on how long this internet datagram may exist.

TOS
Type of Service

Total Length
The internet header field Total Length is the length of the datagram in octets including internet header and data.

TTL

Time to Live

Type of Service
An internet header field which indicates the type (or quality) of service for this internet datagram.

UDP
User Datagram Protocol: A user level protocol for transaction oriented applications.

User
The user of the internet protocol. This may be a higher level protocol module, an application program, or a gateway program.

Version
The Version field indicates the format of the internet header.

REFERENCES

[1] Cerf, V., "The Catenet Model for Internetworking," Information Processing Techniques Office, Defense Advanced Research Projects Agency, IEN 48, July 1978.

[2] Bolt Beranek and Newman, "Specification for the Interconnection of a Host and an IMP," BBN Technical Report 1822, Revised May 1978.

[3] Postel, J., "Internet Control Message Protocol - DARPA Internet Program Protocol Specification," RFC 792, USC/Information Sciences Institute, September 1981.

[4] Shoch, J., "Inter-Network Naming, Addressing, and Routing," COMPCON, IEEE Computer Society, Fall 1978.

[5] Postel, J., "Address Mappings," RFC 796, USC/Information Sciences Institute, September 1981.

[6] Shoch, J., "Packet Fragmentation in Inter-Network Protocols," Computer Networks, v. 3, n. 1, February 1979.

[7] Strazisar, V., "How to Build a Gateway", IEN 109, Bolt Beranek and Newman, August 1979.

[8] Postel, J., "Service Mappings," RFC 795, USC/Information Sciences Institute, September 1981.

[9] Postel, J., "Assigned Numbers," RFC 790, USC/Information Sciences Institute, September 1981.

Exhibit 11

NEW COLLEGE EDITION



THE AMERICAN HERITAGE DICTIONARY OF THE ENGLISH LANGUAGE

Words that are believed to be registered trademarks have been checked with authoritative sources. No investigation has been made of common-law trademark rights in any word, because such investigation is impracticable. Words that are known to have current registrations are shown with an initial capital and are also identified as trademarks. The inclusion of any word in this Dictionary is not, however, an expression of the publishers' opinion as to whether or not it is subject to proprietary rights. Indeed, no definition in this Dictionary is to be regarded as affecting the validity of any trademark.

©1969, 1970, 1971, 1973, 1975, 1976 by Houghton Mifflin Company
All correspondence and inquiries should be directed to
Dictionary Division, Houghton Mifflin Company
One Beacon Street, Boston, Massachusetts 02107

All rights reserved under Bern and Pan-American Copyright Conventions

ISBN: 0-395-20360-0 (new college edition; thumb-indexed)
0-395-20359-7 (new college edition; plain edges)
0-395-24575-3 (high-school edition)

Library of Congress Catalog Card Number 76-86995

Manufactured in the United States of America

Computer-composed by Inforonics, Inc.
in Maynard, Massachusetts

Case 1:05-cv-11298-DPW Document 131-12 Filed 08/17/2007 Page 4 of 4

paper, stone, wood, or other surface **2** To mark or engrave (a surface) with words or letters **3** To enter (a name) on a list or in a register **4.** To dedicate (a book or photograph, for example) to another **5** *Geometry* To enclose (a polygon or polyhedron) within a closed configuration of lines, curves, or surfaces so that every vertex of the enclosed figure is incident on the enclosing configuration [Latin *inscribere* : *in-*, in + *scribere*, to write (see **skeri-** in Appendix*)] —**in·scrib′er** *n*

in·scrip·tion (in-skrĭp′shən) *n* **1** The act or instance of inscribing. **2.** That which is inscribed, such as the wording on a coin or medal, an epitaph, or a dedication of a book or work of art. **3.** An enrollment or registration of names [Middle English *inscripcioun*, from Latin *inscriptio*, a writing in or upon, from *inscribere* (past participle *inscriptus*), INSCRIBE.] —**in·scrip′tion·al, in·scrip′tive** *adj* —**in·scrip′tive·ly** *adv.*

in·scru·ta·ble (in-skrōō′tə-bəl) *adj* Not able to be fathomed or understood; impenetrable, enigmatic: *"people who prefer to stand at the bar have, universally, an inscrutable look"* (Thomas Pynchon). See Synonyms at **mysterious.** —**in·scru′ta·bil′i·ty, in·scru′ta·ble·ness** *n* —**in·scru′ta·bly** *adv*

in·sect (in′sĕkt′) *n* **1.** Any of numerous usually small invertebrate animals of the class Insecta (or Hexapoda), having an adult stage characterized by three pairs of legs, a segmented body with three major divisions, and usually two pairs of wings. **2.** Loosely, any of various similar invertebrate animals such as a spider, centipede, or tick. **3.** One who is small or contemptible. [Latin *insectum* (*animale*), "segmented (animal)" (translation of Greek *entomon;* see **entomo-**), from *insectus*, past participle of *insecāre*, to cut into : *in-*, in + *secāre*, to cut (see **sek-** in Appendix*).]

in·sec·tar·i·um (in′sĕk-târ′ē-əm) *n*, *pl* **-ums** or **-ia** (-ē-ə). Also **in·sec·tar·y** (in′sĕk-tĕr′ē) *pl* **-ies.** A place in which living insects are kept or bred.

in·sec·ti·cide (in-sĕk′tə-sīd′) *n* An agent used to kill insects. —**in·sec′ti·ci′dal** (-sīd′l) *adj.*

in·sec·ti·vore (in-sĕk′tə-vôr′, -vōr′) *n* **1.** Any of various mammals of the order Insectivora, characteristically feeding on insects, and including the shrews, moles, and hedgehogs. **2.** An organism that feeds on insects. [New Latin *Insectivora* (order) : Latin *insectum*, INSECT + *-vorus*, -VOROUS.]

in·sec·tiv·o·rous (in′sĕk-tĭv′ər-əs) *adj.* **1.** Feeding on insects. **2.** *Botany.* Capable of trapping and absorbing insects, as the pitcher plant or Venus's-flytrap. [INSECT + -VOROUS.]

in·se·cure (in′sĭ-kyōōr′) *adj* **1.** Not secure or safe; inadequately guarded or protected. **2.** Unsure; unstable; shaky. **3.** Apprehensive or lacking self-confidence. —**in′se·cure′ly** *adv.* —**in′se·cu′ri·ty, in′se·cure′ness** *n.*

in·sem·i·nate (in-sĕm′ə-nāt′) *tr.v.* **-nated, -nating, -nates. 1. a.** To sow seed in. **b.** To introduce semen into the uterus of. **2.** To sow ideas in. [Latin *insēmināre* : *in-*, in + *sēmināre*, to plant, from *sēmen* (stem *sēmin-*), seed, SEMEN.] —**in·sem′i·na′tion** *n.* —**in·sem′i·na′tor** (-nā′tər) *n*

in·sen·sate (in-sĕn′sāt′, -sĭt) *adj.* **1. a.** Lacking sensation; inanimate. **b.** Unconscious. **2.** Lacking sensibility; inhuman; unfeeling. **3.** Lacking sense; foolish. —**in·sen′sate·ly** *adv*

in·sen·si·ble (in-sĕn′sə-bəl) *adj* **1.** Imperceptible; inappreciable: *an insensible change.* **2.** Deprived of the power of feeling; unconscious: *"Sir Henry lay insensible where he had fallen"* (Arthur Conan Doyle). **3. a.** Unsusceptible to or unaffected by: *insensible to the cold.* **b.** Unheeding; incognizant; unmindful: *I am not insensible of your concern.* **c.** Unfeeling; indifferent; callous. **4.** *Rare.* Lacking intelligence; irrational. —**in·sen′si·bil′i·ty** *n.* —**in·sen′si·bly** *adv.*

in·sen·si·tive (in-sĕn′sə-tĭv) *adj.* **1.** Not sensitive; lacking sensitivity. **2.** Unfeeling. —**in·sen′si·tiv′i·ty, in·sen′si·tive·ness** *n.* —**in·sen′si·tive·ly** *adv.*

in·sen·ti·ent (in-sĕn′shənt) *adj.* Without sensation or consciousness; inanimate. —**in·sen′ti·ence** *n.*

in·sep·a·ra·ble (in-sĕp′ər-ə-bəl) *adj. Abbr.* **insep.** Incapable of being separated. —**in·sep′a·ra·bil′i·ty, in·sep′a·ra·ble·ness** *n* —**in·sep′a·ra·bly** *adv.*

in·sert (in-sûrt′) *tr.v.* **-serted, -serting, -serts. 1.** To put or set into, between, or among another or other things. **2.** To introduce into the body or text of something; interpolate. **3.** To place into an orbit, trajectory, or stream. —*n.* (in′sûrt′) Something inserted or intended for insertion, such as a paragraph, a map, an extra leaf, or a circular inserted in an article or a book. [Latin *inserere* (past participle *insertus*) : *in-*, in + *serere*, to sow, plant (see **sē-**[1] in Appendix*).] —**in·sert′er** *n*

in·sert·ed (in-sûr′tĭd) *adj.* Joined to another part, as stamens to a corolla.

in·ser·tion (in-sûr′shən) *n.* **1.** The act of inserting **2.** Something inserted. **3.** A point or mode of attachment. **4.** A strip of lace, embroidery, or other trim to be inserted in a garment, tablecloth, or the like. —**in·ser′tion·al** *adj.*

in·ses·so·ri·al (in′sə-sôr′ē-əl, -sōr′ē-əl) *adj. Rare.* Perching or adapted for perching. [From Late Latin *insessor*, "one that perches," from Latin *insidēre* (past participle *insessus*), to sit upon : *in-*, on + *sedēre*, to sit (see **sed-**[1] in Appendix*).]

in·set (in-sĕt′) *tr.v.* **-set, -setting, -sets.** To insert; set in. —*n.* (in′sĕt′). **1.** Something set in, as: **a.** A small map or illustration set within a larger one **b.** A leaf or group of pages inserted in a publication. **c.** A piece of material set into a dress as trim. **2. a.** An inflow, as of water. **b.** A channel.

in·shore (in′shôr′, -shor′) *adj* **1.** Close to the shore. **2.** Coming toward the shore —*adv* Toward the shore

in·shrine. Variant of **enshrine.**

in·side (in-sīd′, in′sīd′) *n* **1.** The inner or interior part. **2.** An inner side or surface **3.** The middle part; the part away from the edge **4.** *Plural Informal* **a.** The inner organs, entrails **b.** The inner parts or workings **5** *Slang* Confidential information; tip. —**inside out.** With the inner surface turned out or reversed —**on the inside** In a position of confidence or influence —*adj.* **1.** Inner, interior **2** For the interior **3** Pertaining to or coming from those in authority: *the inside office* **4.** *Baseball.* Passing too near the body of the batter: Said of a pitch. —*adv* (in′sīd′). Into or in the interior, within —*prep* (in′sīd′) **1.** Within. *inside an hour* **2** Into: *to go inside the house* —**inside of.** Within the boundaries of, inside

Usage: *Inside* (preposition) is capable of functioning without a succeeding *of.* Especially in writing, *inside* is preferably used without *of* to indicate position or location: *Stay inside the house. Inside of* is used informally for *within* in references to time or distance: *inside of an hour, inside of a mile*

Inside Passage. A protected navigation route extending 950 miles off the coasts of Alaska and British Columbia, Canada, through the islands of the Alexander Archipelago. Also called "Inland Passage."

in·sid·er (in-sī′dər) *n* **1.** An accepted member of a clique **2.** One who has the ear of those in charge

inside track. 1. In a curved race track, the path next to the inner rail **2.** An advantageous spot in a competition

in·sid·i·ous (in-sĭd′ē-əs) *adj.* **1.** Working or spreading harmfully in a subtle or stealthy manner **2.** Intended to entrap, wily; treacherous: *"a tedious argument of insidious intent"* (T.S Eliot) **3.** Sly; beguiling: *"I was now a woman/Insidious, subtle, versed in the world and rich"* (Edgar Lee Masters). **4** Lying in wait. [Latin *insidiōsus*, "lying in wait for," from *insidiae*, ambush, from *insidēre*, to sit in or on, lie in wait for: See **in-sessorial.**] —**in·sid′i·ous·ly** *adv* —**in·sid′i·ous·ness** *n*

in·sight (in′sīt′) *n* **1.** The capacity to discern the true nature of a situation, penetration. **2.** An elucidating glimpse.

in·sig·ni·a (in-sĭg′nē-ə) *n., pl.* **insignia** or **-as.** Also **in·sig·ne** (-nē). See Usage note below. **1.** A badge of office, rank, membership, or nationality; emblem. **2.** A distinguishing sign: *the insignia of success.* [Latin, plural of *insigne*, sign, mark, from *insignis*, distinguished, marked : *in-*, in + *signum*, SIGN.]

Usage: *Insignia* is acceptably used as a singular, according to 73 per cent of the Usage Panel. The plural *insignias*, a newer coinage, is acceptable to 56 per cent of the Panel.

in·sig·nif·i·cance (in′sĭg-nĭf′ĭ-kəns) *n* The quality or state of being insignificant.

in·sig·nif·i·can·cy (in′sĭg-nĭf′ĭ-kən-sē) *n, pl.* **-cies. 1.** Insignificance. **2.** Something insignificant.

in·sig·nif·i·cant (in′sĭg-nĭf′ĭ-kənt) *adj.* **1. a.** Not significant. **b.** Trivial. **2.** Small. **3.** Meaningless. —**in′sig·nif′i·cant·ly** *adv.*

in·sin·cere (in′sĭn-sĭr′) *adj.* Not sincere; hypocritical. —**in′sin·cere′ly** *adv.* —**in·sin·cer′i·ty** (-sĕr′ə-tē) *n.*

in·sin·u·ate (in-sĭn′yōō-āt′) *v.* **-ated, -ating, -ates.** —*tr.* **1. a.** To introduce gradually and insidiously. **b.** To edge or worm (oneself) by subtle and artful means: *"she insinuated herself into the company of those who were celebrated for their conversation"* (Thornton Wilder). **2.** To convey with oblique hints and allusions; hint covertly. —*intr.* To make insinuations. —See Synonyms at **suggest.** [Latin *insinuāre*, to wind one's way into : *in-*, in + *sinuāre*, to curve, from *sinus*, curve, SINUS.] —**in·sin′u·a′tive** *adj* —**in·sin′u·a′tor** (-ā′tər) *n*

in·sin·u·at·ing (in-sĭn′yōō-ā′tĭng) *adj* **1.** Provoking gradual doubt or suspicion; suggestive: *insinuating remarks* **2.** Ingratiating. *a silky insinuating voice* —**in·sin′u·at′ing·ly** *adv*

in·sin·u·a·tion (in-sĭn′yōō-ā′shən) *n.* **1.** The act or practice of insinuating. **2.** An artfully indirect suggestion.

in·sip·id (in-sĭp′ĭd) *adj.* **1.** Lacking flavor or zest, unpalatable. **2.** Lacking excitement or interest; unstimulating; vapid: *"I have no taste of those insipid dry discourses"* (Congreve). [Late Latin *insipidus* : Latin *in-*, not + *sapidus*, SAPID.] —**in′si·pid′i·ty** (in′sĭ-pĭd′ə-tē), **in·sip′id·ness** *n.* —**in·sip′id·ly** *adv.*

in·sip·i·ence (in-sĭp′ē-əns) *n. Archaic* Lack of wisdom. [Middle English, from Old French, from Latin *insipientia*, from *insipiēns*, unwise : *in-*, not + *sapiēns*, wise, SAPIENT.]

in·sist (in-sĭst′) *v* **-sisted, -sisting, -sists.** —*intr.* To keep resolutely to or emphasize an assertion, demand, or course. Usually used with *on* or *upon* —*tr.* To assert or demand vehemently and persistently *We insist that you accept these gifts* [Latin *insistere*, to stand on, persist : *in-*, on + *sistere*, to cause to stand, stand firm (see **stā-** in Appendix*).] —**in·sis′tence, in·sis′ten·cy** *n*

in·sis·tent (in-sĭs′tənt) *adj* **1.** Persistent; pertinacious. **2** Demanding notice: *insistent hunger.* —**in·sis′tent·ly** *adv*

in si·tu (in sī′tōō, sĭt′ōō). *Latin* In (its original) place.

in·snare. Variant of **ensnare.**

in·so·bri·e·ty (in′sə-brī′ə-tē) *n* Lack of sobriety; intemperance.

in·so·far (in′sō-fär′) *adv* To such an extent. Used with *as:* *"The scientist loves both the truth he discovers and himself insofar as he discovers it"* (Paul Tillich).

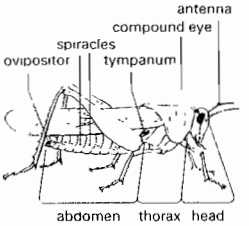
in·so·late (in′sō-lāt′) *tr.v.* **-lated, -lating, -lates.** To expose to sunlight, as for bleaching. [Latin *insolāre* : *in-*, in + *sōl*, sun (see **sawel-** in Appendix*).]

in·so·la·tion (in′sō-lā′shən) *n.* **1 a.** Exposure to sunlight **b.** Therapeutic exposure to sunlight. **2. Sunstroke** (*see*). **3. a.** The solar radiation incident on the earth or another planet. **b.** The rate of delivery of such radiation per unit area surface.

in·sole (in′sōl′) *n* **1.** The inner sole of a shoe or boot **2.** An extra strip of material put inside a shoe for comfort or protection.

in·so·lence (in′sə-ləns) *n.* **1.** The quality of being insolent **2.** An instance of insolent behavior.

in·so·lent (in′sə-lənt) *adj* **1.** Presumptuous and insulting in



insect
Grasshopper

Exhibit 12

UNITED STATES DISTRICT COURT
DISTRICT OF MASSACHUSETTS
EASTERN DIVISION

ENTERASYS NETWORKS, INC., Plaintiff, v. FOUNDRY NETWORKS, INC. AND EXTREME NETWORKS, INC., Defendants.	Civil Action No: 05-CV-11298 (DPW)

DECLARATION OF JAMES F. KUROSE IN SUPPORT OF OPENING CLAIM CONSTRUCTION BRIEF OF PLAINTIFF ENTERASYS NETWORKS, INC.

I, JAMES F. KUROSE, declare as follows:

1. I am an expert witness retained on behalf of the named Plaintiff Enterasys Networks, Inc. ("Enterasys") in the above-referenced action. I submit this Declaration in support of Enterasys' Opening Claim Construction Brief due to be filed with the Court under separate cover on August 17, 2007. I have personal knowledge of the facts and opinions stated herein and, if sworn as a witness, I could and would testify competently thereto.

STATEMENT OF QUALIFICATIONS

2. I presently hold tenure as a Distinguished University Professor in the Department of Computer Science at the University of Massachusetts at Amherst. I received my Ph.D. in Computer Science from Columbia University in 1984 and have been on the faculty at the University of Massachusetts since that time. During my 1990/1991 sabbatical year, I was a visiting scientist in the Communication Networking Department at IBM Research. During my 1997/1998 sabbatical year, I was a Visiting Researcher at the Institut National de Recherche en Informatique et en Automatique (INRIA) and at Institut Eurecom, both in Sophia Antipolis,

France. I was a Visiting Professor, University of Paris VI (Universite Pierre et Marie Curie) in June 2005 and 2006, and was a Visiting Scientist at Thomson Paris Research Lab for two months in 2006.

3. My research is in the area of computer networks and their protocols, network measurement, sensor networks, multimedia communication, and modeling and performance evaluation. I have worked in these technical areas for more that 25 years. I have co-authored more than 200 refereed journal articles and conference papers. With Keith Ross, I am co-author of the textbook, Computer Networking: A Top-down Approach, 4th edition, (Addison-Wesley Longman). I also hold two U.S. patents.

4. I have served on various networking-related panels for the National Academy of Sciences / National Research Council, and for the National Science Foundation. I have served as the Editor-in-Chief of the IEEE Transactions on Communications, and was the founding Editor-in-Chief of the IEEE/ACM Transactions on Networking. I have been active in the program committees for the IEEE Infocom, ACM SIGCOMM, and ACM SIGMETRICS conferences for a number of years, and have served as Technical Program Co-Chair for these conferences. I am a Fellow of the Institute of Electrical and Electronics Engineers (IEEE) and of the Association for Computing Machinery (ACM).

5. In the 1995 timeframe, I was performing research in the area of Internet protocols. The computer networks course that I was teaching at the time covered the topic of encapsulation.

BRIEF SUMMARY OF CONCLUSIONS

6. This Declaration presents my expert opinion on the meaning of the term "encapsulation." I will show that several forms of encapsulation would have been known to one of ordinary skill in the art of data communications networking in the mid-1990s. While some of these forms of encapsulation consisted of pre-pending additional information to the data being

"encapsulated" (*i.e.*, placing this additional information "in front of" the data being encapsulated), other definitions and use of encapsulation placed this additional information either 'in the middle of" the data being encapsulated (*i.e.*, within the data being encapsulated) or "following" the data being encapsulated. Also, in some cases, the act of encapsulation modified the data being encapsulated.

7. The remainder of this Declaration is structured as follows. I will begin with a short informal discussion of the process of encapsulation. I will then cover what is likely the most common form of encapsulation -- placing the additional information in a so-called header field that is pre-pended to (*i.e.*, added "at the front of") the data being encapsulated. I will then describe three additional forms of encapsulation that would have been well-understood and known to one of ordinary skill in the art in the mid-1990s, in which this additional information is placed either within (*i.e.*, "in the middle of") or following the data being encapsulated. I will then conclude with a short discussion of encapsulation in the IEEE's 802.1Q standard.

ENCAPSULATION

8. In a computer network, two entities communicate by exchanging messages, often referred to as "packets." A protocol defines the format and the order of messages exchanged between the two or more communicating entities, as well as the actions taken on the transmission and/or receipt of a message or other event. Typically, the sending side of a protocol takes the unit of data to be sent to the other side and adds additional information to this original data to form the message to be sent. Often, the original unit of data may itself be structured to consist of a number of fields of information, and these fields may include header information and a data field, and so this original unit of data may itself be referred to as a packet. This process of taking the original unit of data and then including additional information to form a packet is referred to as encapsulation. The original unit of data is referred to as the data being encapsulated.

9. The term "encapsulation" has many interpretations in the networking literature of the 1990s. Perhaps the most common interpretation is similar to that offered in Perlman's text:

> **encapsulation** – Handling protocol A's packets, complete with A's header information, as data carried by protocol B. Encapsulated protocol A packets have a B header, followed by an A header, followed by the information that A is carrying as its own data.

(Perlman 1992, p. 370.) [1]

10. In Perlman's definition, the encapsulated packet (header plus payload) from A properly follows the protocol B header. That is, there is a strict ordering in which the B header comes first and the A packet follows.

11. Perlman's definition above is probably the most widely-used definition. But strictly interpreted, it is also quite narrow, and numerous other definitions and uses of "encapsulation" were in use in the mid-1990s.

12. Indeed, in the second edition of Perlman's text [Perlman, 1999][2], Perlman herself clarifies her original definition to note that protocol A and protocol B could be the same protocol:

> **encapsulation** – Handling protocol A's packets, complete with A's header information, as data carried by protocol B. Encapsulated protocol A packets have a B header, followed by an A header, followed by the information that A is carrying as its own data. Note that A could =B, as IP inside IP.

(Perlman 2d Ed. 1999, p. 525.)

13. A broader, alternative definition of encapsulation is provided in RFC 1208 [RFC 1208][3], "A Glossary of Networking Terms," published in 1991. Internet RFCs are publicly available documents provided by the Internet Engineering Task Force. In some cases they define

[1] R. Perlman, *Interconnections: Bridges And Routers*, Addison Wesley Professional Computing Series, 1992.

[2] R. Perlman, *Interconnections: Bridges, Routers, Switches, And Internetworking Protocols (2nd Ed.)*, Addison Wesley Professional Computing Series, 1999.

[3] O.J. Jacobsen, D.C. Lynch, "A Glossary Of Networking Terms," RFC 1208, March 1991.

specific Internet protocols, and in other cases are informational, providing information that could be of interest to the technical community. A person of ordinary skill in the art would know about the existence of RFCs since they are widely cited in textbooks, and each Internet protocol is standardized in the form of an RFC. RFC 1208 defines encapsulation as follows:

> **encapsulation**: The technique used by layered protocols in which a layer adds header information to the protocol data unit (PDU) from the layer above. As an example, in Internet terminology, a packet would contain a header from the physical layer, followed by a header from the network layer (IP), followed by a header from the transport layer (TCP), followed by the application protocol data.

(RFC 1208.)

14. Note here that the first sentence in the definition above discusses the fact that header information is added to the PDU, without specifying the details (*e.g.*, where) this information is added. The RFC expressly notes that placing header information preceding the data packet being encapsulated is but one "example" of how to encapsulate.

15. The Perlman definition specifies, and RFC 1208 suggests an example of, the order of packet headers, with the encapsulating header (header B in Perlman's definition) preceding the header of the packet being encapsulated. However, just as Perlman's original definition, later definition, and RFC 1208 increasingly broaden the definition of encapsulation, there were also definitions of "encapsulation" in the literature at the time that, as discussed below, do not require that the encapsulating packet header precede the data being encapsulated.

16. In the following three sections, I will discuss three forms of encapsulation that would have been known to one of ordinary skill in the art in which encapsulation header information is interspersed with (*i.e.*, occurs "in the middle of") or follows the data being encapsulated, or where the data being encapsulated is itself modified:

- **[RFC 2004][4] and [Johnson 1994][5].** Here, the header of the packet being encapsulated is modified and a new header is inserted "in the middle"[6] of the packet begin encapsulated. Specifically, it follows the modified header and precedes the data payload portion of the packet being encapsulated.

- **Trailer Encapsulation.** Here, the encapsulation header information is inserted "at the end" of the packet being encapsulated. Additionally, the header fields and the data payload field of the original packet being encapsulated are reversed.

- **IP switching.** Here, the header fields of the original IP packet being encapsulated are changed - some fields that were previously in the IP header are deleted, and encapsulation information appears "at the end of" the packet being encapsulated.

ENCAPSULATION IN [RFC 2004] AND [JOHNSON 1994]

17. [RFC 2004], "Minimal Encapsulation within IP," is an Internet standards track document that presents an encapsulation method in which the header of the packet being encapsulated is modified and new header is inserted following the modified header, and preceding the IP payload (data) portion of the packet being encapsulated. The figure and text below is taken verbatim from [RFC 2004], and shows that the encapsulation information is added "in the middle" of the packet being encapsulated:

3. Minimal Encapsulation

A minimal forwarding header is defined for datagrams which are not fragmented prior to encapsulation. Use of this encapsulating method is optional. Minimal encapsulation MUST NOT be used when an original datagram is already fragmented, since there is no room in the minimal forwarding header to store fragmentation information. To encapsulate an IP datagram using minimal encapsulation, the minimal forwarding header is inserted into the datagram, as follows:

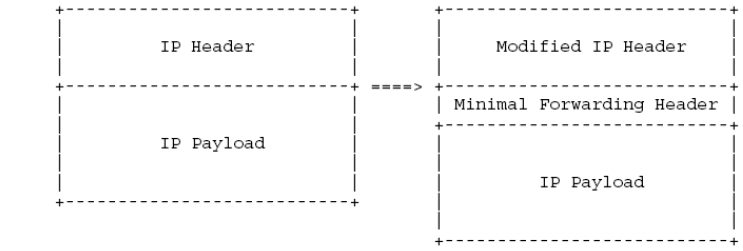


Figure 1. Excerpted text and figure from [RFC 2004].

[4] C. Perkins, "Minimal Encapsulation Within IP," RFC 2004, Standards Track, October 1996.

[5] D.B. Johnson, "Scalable And Robust Internetwork Routing For Mobile Hosts," *Proceedings Of The 14th International Conference On Distributed Computing Systems*, pp. 2-11, June 1994.

[6] Here, above and below, the quotations surrounding text regarding the location of header information indicate colloquial use of a phrase, rather than quotations from a referenced text. For example, header information appearing "in the middle" of the data being encapsulation need not appear exactly equidistant from the first and last byte of the data being encapsulated, but rather strictly after the beginning of, and strictly before the end of, the data being encapsulated.

18. While [RFC 2004] was released in October 1996, the diagram above can also be found in an earlier, July 6, 1995 Internet Draft [Perkins 1995][7] document. By way of background, Internet drafts are non-reviewed documents, some of which later become Requests for Comments [*i.e.*, "RFCs"]. The diagram above "specifies a method by which an IP datagram may be encapsulated (carried as payload) within an IP datagram, …" [RFC 2004, p. 1].

19. The encapsulation described in [RFC 2004] is similar to the encapsulation defined in [Johnson 1994], which also inserts a new encapsulation header "in the middle of" the packet being encapsulated, after a modified original header. Figure 2 is taken directly from [Johnson 1994]:

4. Forwarding Packets to a Mobile Host

4.1. The MHRP Encapsulation Protocol

In order to forward IP packets destined for a mobile host to the foreign agent currently serving that host, MHRP introduces a new *encapsulation* protocol. The home agent or initial cache agent handling the packet transforms it into a new IP packet, addressed to the foreign agent, by adding a new header (the *MHRP header*) to the packet between the IP header and any existing transport-level header such as TCP [11] or UDP [8], as illustrated in Figure 2. Once the MHRP header is added, the packet uses only normal IP routing for delivery to the foreign agent. This use of encapsulation is known as *tunneling*.

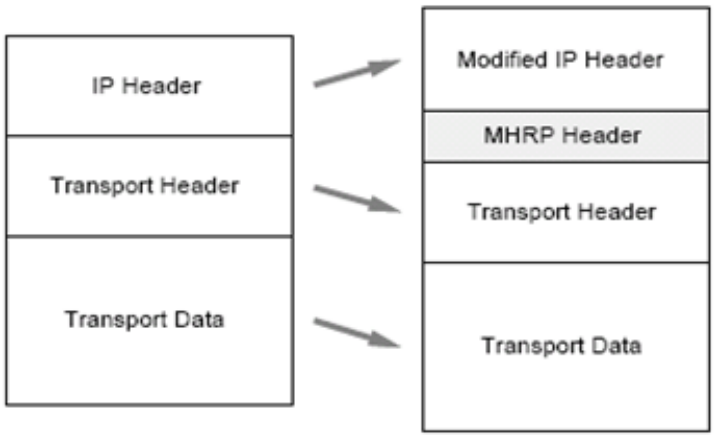


Figure 2. Excerpted text and figure from [Johnson 1994].

[7] C. Perkins, "Minimal Encapsulation Within IP," Internet Draft, *http://tools.ietf.org/id/draft-ietf-mobileip-minenc-oo.txt*, July 6, 1995.

20. [RFC 2004] and [Johnson 1994] teach the following:

- Encapsulation methods have been defined in which the header of the packet being encapsulated is modified and a new encapsulation header is inserted "in the middle of" the packet being encapsulated, specifically it follows the modified header and precedes the IP payload (data) portion of the packet being encapsulated.

(RFC 2004; Johnson 1994.)

TRAILER ENCAPSULATION

21. [RFC 893][8], "Trailer encapsulations," and [Leffler 1989][9] describe trailer encapsulation in which portions of the encapsulating header information for the packet being encapsulated follow the data portion of the packet being encapsulated, and precede the header portion of the packet being encapsulated, and thus occur "in the middle of" the packet (header and data) being encapsulated. Furthermore, the header fields and the payload (data) field of the original packet being encapsulated are reversed.

22. [RFC 893] describes "trailer encapsulations" as the "rearrangement of data at the link level to force variable length header information to "trail" the data." The text and diagram in

[8] S. Leffler, M.J. Karels, "Trailer Encapsulations," RFC 893, April 1984.

[9] S. Leffler et al., *The Design And Implementation Of The 4.3 BSD Unix Operating System*, Addison Wesley, 1989.

Figure 3 is taken directly from [RFC 893]:

The format of a trailer packet is shown in the following diagram.

```
+----+-------------------------------------------------+----+
| LH |                      data                       | TH |
+----+-------------------------------------------------+----+
     ^                     (  ^  )                     ^
```

LH:

The fixed-size local network header. For 10 a Mb/s Ethernet, the 16-byte Ethernet header. The type field in the header indicates that both the packet type (trailer) and the length of the data segment.

For the 10 Mb/s Ethernet, the types are between 1001 and 1010 hexadecimal (4096 and 4112 decimal). The type is calculated as 1000 (hex) plus the number of 512-byte pages of data. A maximum of 16 pages of data may be transmitted in a single trailer packet (8192 bytes).

data:

The "data" portion of the packet. This is normally only data to be delivered to the receiving processes (i.e. it contains no TCP or IP header information). Data size is always a multiple of 512 bytes.

TH:

The "trailer". This is actually a composition of the original protocol headers and a fixed size trailer prefix which defines the type and size
of the trailing data. The format of a trailer is shown below.

The carats (^) indicate the page boundaries on which the receiving host would place its input buffer for optimal alignment when receiving a trailer packet. The link level receiving routine is able to locate the trailer using the size indicated in the link level header's type field. The receiving routine is expected to discard the link level header and trailer prefix, and remap the trailing data segment to the front of the packet to regenerate the original network level packet format.

Trailer Format

```
+----------------+----------------+------~...~----------+
|      TYPE      |  HEADER LENGTH |  ORIGINAL HEADER(S) |
+----------------+----------------+------~...~----------+
```

Type: 16 bits

The type field encodes the original link level type of the transmitted packet. This is the value which would normally be placed in the link level header if a trailer were not generated.

Header length: 16 bits

The header length field of the trailer data segment. This specifies the length in bytes of the following header data.

Original headers: <variable length>

The header information which logically belongs before the data segment. This is normally the network and transport level protocol headers.

Summary

A link level encapsulation which promotes alignment properties necessary for the efficient use of virtual memory hardware facilities has been described. This encapsulation format is in use on many systems and is a standard facility in 4.2BSD UNIX. The encapsulation provides an efficient mechanism by which cooperating hosts on a local network may obtain significant performance improvements. The use of this encapsulation technique currently requires uniform cooperation from all hosts on a network; hopefully a per host negotiation mechanism may be added to allow consenting hosts to utilize the encapsulation in a non-uniform environment.

Figure 3. Text and figure excerpted from [RFC 893].

23. Note that the original header(s) of the packet being encapsulated have been reordered to occur after the data portion of the packet being encapsulated. This form of encapsulation not only places encapsulation information both before and after portions of the original packet that is being encapsulated, but actually rearranges the order of fields of the packet being encapsulated. Note that in the summary paragraph of Figure 3, the authors note "This encapsulation format is in use on many systems and is a standard facility in 4.2BSD UNIX."

24. [RFC 1122][10], "Requirements for Internet Hosts -- Communication Layers," characterizes the encapsulation in [RFC 893] as:

> The trailer protocol is a link-layer encapsulation technique that rearranges the data contents of packets sent on the physical network. In some cases, trailers improve the throughput of higher layer protocols by reducing the amount of data copying within the operating system. Higher layer protocols are unaware of trailer use, but both the sending and receiving host MUST understand the protocol if it is used.

[RFC 1122] characterizes itself and a companion document by noting,

> These [two]documents are intended to provide guidance for vendors, implementers, and users of Internet communication software. They represent the consensus of a large body of technical experience and wisdom, contributed by the members of the Internet research and vendor communities. This RFC enumerates standard protocols that a host connected to the Internet must use, and it incorporates by reference the RFCs and other documents describing the current specifications for these protocols.

25. In their text [Leffler 1989, p. 339], the authors describe a trailer encapsulation of a TCP/IP packet within an Ethernet frame in which encapsulating trailer-protocol header information occurs "in the middle of" the TCP/IP packet being encapsulated -- *i.e.*, following the TCP/IP data and preceding the TCP IP header. Figure 4 below is Figure 11.11 from [Leffler 1989]:

[10] R. Braden, Ed., "Requirements For Internet Hosts – Communication Layers," RFC 1122, October 1989.

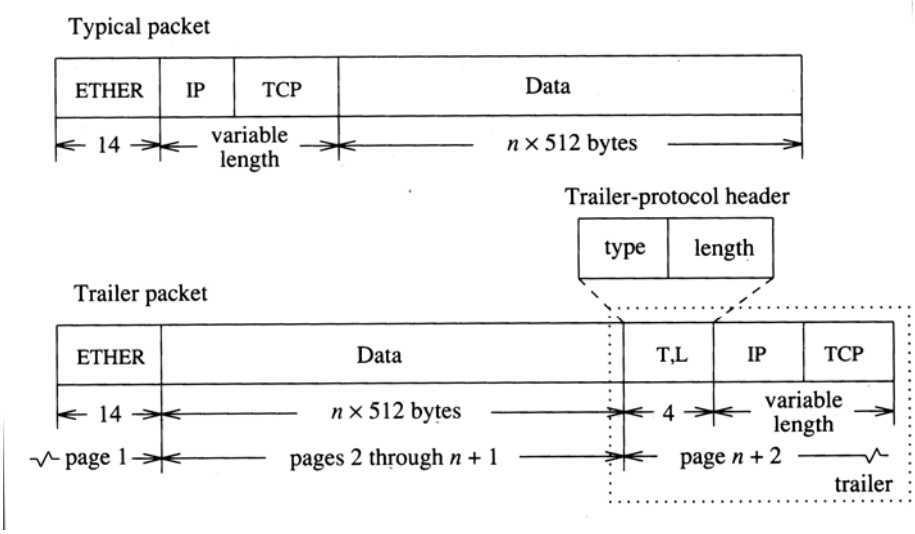


Figure 4. Trailer-protocol packet format (Figure 11.11from [Leffler 1989]).

26. Trailer encapsulation teaches:

- There are forms of encapsulation that do not require header information to be placed before the packet being encapsulated. In particular, portions of the encapsulating header may appear "in the middle of" the header and data portions of the packet being encapsulated.
- There are forms of encapsulation, implemented in fielded systems, that exchange the order of the header and data fields of the packet being encapsulated. Thus encapsulation may actively re-order the fields of the original packet being encapsulated.

IP SWITCHING

27. Ipsilon was a company that developed so-called IP-switching technology in the 1990s. Under IP switching, IP packets were encapsulated in ATM AAL5 link-layer frames in one of several different formats. Importantly, the encapsulation of an IP packet in the AAL5 link-layer frame could change as the encapsulated packet traveled from one ATM switch to another.

28. The top half of the Figure 5 below (adapted from Figures 3.9 and 3.10 in [Davie 2000][11]) shows the form of encapsulation when the IP packet is encapsulated in an ATM AAL5

[11] B. Davie, Y. Rekhter, *MPLS Technology And Applications*, Moran Kaufman 2000.

frame that is carried on the default VC. If the encapsulated packet is redirected to a non-default VC at a switch in the network on its path from source to destination, the packet is encapsulated in a different format, using the format shown on the bottom half of Figure 5. When an encapsulated packet is redirected to a non-default VC, the header fields of the original IP packet that is encapsulated within AAL5 are changed – some fields that were previous in the IP header have been deleted, and other fields in the IP header have been reordered:

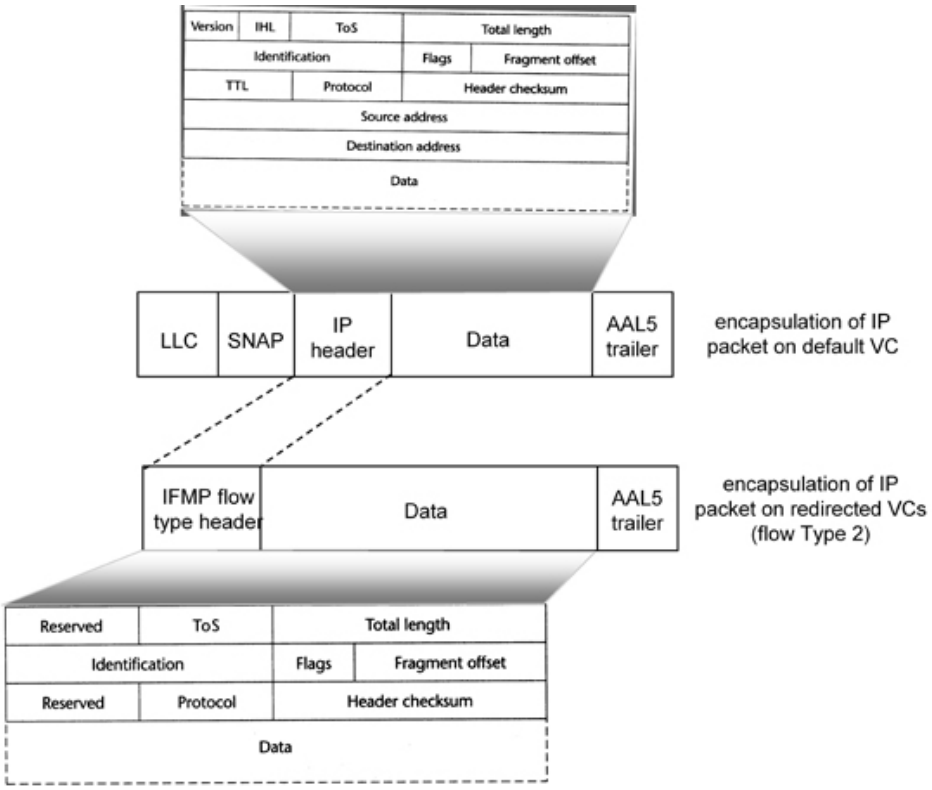


Figure 5. Adapted from Figures 3.9 and 3.10 in [Davie 2000].

29. The IP switching example teaches:

- There are forms of encapsulation that do not require headers to be placed before the encapsulated packet. For example, encapsulation header information follows the packet being encapsulated in Figure 5. [RFC 1483][12] from 1993 additionally gives AAL5 encapsulations where encapsulating header fields follows the packet being encapsulated.

[12] J. Heinanen, "Multiprotocol Encapsulation Over ATM Adaptation Layer 5," RFC 1483, July 1993.

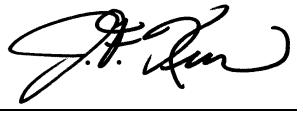
- There are forms of encapsulation that change the fields within the header of the IP packet being encapsulated, deleting some fields and adding new reserved fields in the case of IP Switching. Thus encapsulation need not preserve the content or the identity of the header fields in the packet being encapsulated.

30. In sum, in the above three sections, we have seen definitions and use of encapsulation from the mid-1990s and before that placed encapsulating header information either "in the middle" (*i.e.*, within), or following, the data being encapsulated.

ENCAPSULATION IN THE 802.1Q STANDARD

31. Looking at the 1998 802.1Q standard, we see that in examining "the set of services, frame formats, and MAC methods involved in the provision of VLAN services across 802 LANs," the authors of the 1998 802.1Q standard note that there are "a total of 32 potential frame/encapsulation formats to consider." For the cases where tagging is used, all of these referenced encapsulation formats involve 802.1Q VLAN tags that occur "in the middle of" the 802 frame.

I declare under penalty of perjury that the foregoing is true and correct and that this Declaration is executed on this 16th day of August, 2007 at Paris, France.



JAMES F. KUROSE

BN1: 35040609

Exhibit 13

Merriam-Webster
Merriam-Webster OnLine

Home
Visit Our Sites
Premium Services
Downloads
Word of the Day
Word Games
Open Dictionary
Spelling Bee Hive
Word for the Wise
Online Store
Help
About Us

Also Visit Unabridged Dictionary Learner's Dictionary Word Central for Kids Collegiat

Dictionary Thesaurus Spanish/English Medical

e.g.

2 entries found for **eg**.
To select an entry, click on it.

eg
Eg

Go

Main Entry: **e.g.**
Function: *abbreviation*
Etymology: Latin *exempli gratia*
for example

Learn more about "e.g." and related topics at Britannica.com

Find more about "e.g." instantly with Live Search

See a map of "e.g." in the Visual Thesaurus

Pronunciation Symbols

Products Premium Services Company Info Contact Us Advertising I

© 2006-2007 Merriam-Webster, Incorporated

Exhibit 14

Merriam-Webster
Merriam-Webster OnLine

Home
Visit Our Sites
Premium Services
Downloads
Word of the Day
Word Games
Open Dictionary
Spelling Bee Hive
Word for the Wise
Online Store
Help
About Us

Also Visit Unabridged Dictionary Learner's Dictionary Word Central for Kids Collegiat

Dictionary Thesaurus Spanish/English Medical

i.e.

4 entries found for **i.e.**.
To select an entry, click on it.

ie
IE
ie
-ie

Go

Main Entry: **i.e.**
Function: *abbreviation*
Etymology: Latin *id est*
that is

Learn more about "i.e." and related topics at Britannica.com

Find more about "i.e." instantly with Live Search

See a map of "i.e." in the Visual Thesaurus

Pronunciation Symbols

Products Premium Services Company Info Contact Us Advertising I

© 2006-2007 Merriam-Webster, Incorporated

Exhibit 15

RANDOM HOUSE UNABRIDGED DICTIONARY

Second Edition

Dedicated to the memory of
Jess Stein

COPYRIGHT © 1993, 1987, BY RANDOM HOUSE, INC.

First Edition: Copyright © 1983, 1981, 1979, 1973, 1971, 1970, 1969, 1967, 1966, by Random House, Inc.

All rights reserved under International and Pan-American Copyright Conventions. No part of this book may be reproduced in any form or by any means, electronic or mechanical, including photocopying, without permission in writing from the publisher. All inquiries should be addressed to Reference Department, Random House, Inc., 201 E. 50th Street, New York, N.Y. 10022.
Published in the United States by Random House, Inc., and simultaneously in Canada by Random House of Canada Limited, Toronto

The *Random House Unabridged Dictionary, Second Edition,* is a revised and updated edition of *The Random House Dictionary of the English Language, Second Edition, Unabridged.*

Random House Unabridged Dictionary, The Random House Dictionary of the English Language, and the abbreviations RHD, RHDEL, RHD-I, and RHD-II are trademarks of Random House, Inc.

Library of Congress Cataloging-in-Publication Data
Random House unabridged dictionary.
(Random House dictionaries)
1. English language–Dictionaries. I. Flexner,
Stuart Berg. II. Series.
PE1625.R3 1987 423 93-84591
ISBN 0-679-42917-4; 0-679-42441-5 (with CD-ROM)

A number of entered words which we have reason to believe constitute trademarks have been designated as such. However, no attempt has been made to designate as trademarks or service marks all words or terms in which proprietary rights may exist. The inclusion, exclusion, or definition of a word or term is not intended to affect, or to express a judgment on, the validity or legal status of the word or term as a trademark, service mark, or other proprietary term.

The Concise French Dictionary, edited by Francesca L. V. Langbaum, Copyright © 1983, 1954, by Random House, Inc.

The Concise German Dictionary, edited by Jenni Karding Moulton, Copyright © 1983, 1959, by Random House, Inc.

The Concise Italian Dictionary, edited by Robert A. Hall, Jr., Copyright © 1983, 1957, by Random House, Inc.

The Concise Spanish Dictionary, edited by Donald F. Solá, Copyright © 1983, 1954, by Random House, Inc.

International Phonetic Alphabet, courtesy International Phonetic Association.

Manufactured in the United States of America

d.s./sn

New York Toronto London Sydney Auckland

cal percussion instrument having steel bars that are struck with a wooden or metal beater. **6.** *Anat.* incus (def. 1). [bef. 900; ME *anvelt, anfelt,* OE *anfilt(e), anfealt;* c. MD *anvilte,* OHG *anafalz.* See ON, FELT²]

anx•i•e•ty (ang zī′ĭ tē), *n., pl.* **-ties.** **1.** distress or uneasiness of mind caused by fear of danger or misfortune: *He felt anxiety about the possible loss of his job.* **2.** earnest but tense desire; eagerness: *He had a keen anxiety to succeed in his work.* **3.** *Psychiatry.* a state of apprehension and psychic tension occurring in some forms of mental disorder. [1515–25; < L *anxietās,* equiv. to *anxi(us)* ANXIOUS + *-etās,* var. of *-itās* before a vowel]
—**Syn. 1.** fear, foreboding; worry, disquiet. See **apprehension.** —**Ant. 1.** certainty, serenity, tranquillity.

anxi′ety neuro′sis, *Psychiatry.* a neurotic disorder characterized by pervasive anxiety. [1900–05]

anx•i•o•lyt•ic (ang′zē ə lit′ik), *adj.* **1.** anxiety relieving. —*n.* **2.** See **antianxiety drug.** [1960–65; ANXI(ETY) + -O- + -LYTIC]

anx•ious (angk′shəs, ang′-), *adj.* **1.** full of mental distress or uneasiness because of fear of danger or misfortune; greatly worried; solicitous: *Her parents were anxious about her poor health.* **2.** earnestly desirous; eager (usually fol. by an infinitive or *for*): *anxious to please; anxious for our happiness.* **3.** attended with or showing solicitude or uneasiness: *anxious forebodings.* [1615–25; < L *anxius* worried, distressed, deriv. of *angere* to strangle, pain, distress; cf. ANGUISH, -OUS] —**anx′ious•ly,** *adv.* —**anx′ious•ness,** *n.*
—**Usage.** The earliest sense of ANXIOUS (in the 17th century) was "troubled" or "worried": *We are still anxious for the safety of our dear sons in battle.* Its meaning "earnestly desirous, eager" arose in the mid-18th century: *We are anxious to see our new grandson.* Some insist that ANXIOUS must always convey a sense of distress or worry and object to its use in the sense of "eager," but such use is fully standard.

anx′ious seat′, **1.** Also called **anx′ious bench′.** *Chiefly North Atlantic States and Southern and South Midland U.S.* a seat reserved at a revival meeting for those troubled by conscience and eager for spiritual assistance. **2.** a state of anxiety, esp. about the outcome of a vote, negotiation, etc.: *Strikers have been in the anxious seat for the last three days.* [1810–20, *Amer.*]

an•y (en′ē), *adj.* **1.** one, a, an, or some; one or more without specification or identification: *If you have any witnesses, produce them. Pick out any six you like.* **2.** whatever or whichever it may be: *cheap at any price.* **3.** in whatever quantity or number, great or small; some: *Do you have any butter?* **4.** every; all: *Any schoolboy would know that. Read any books you find on the subject.* **5.** (following a negative) at all: *She can't endure any criticism.* —*pron.* **6.** an unspecified person or persons; anybody; anyone: *He does better than any before him.* **7.** a single one or ones; an unspecified thing or things; a quantity or number: *We don't have any left.* —*adv.* **8.** in whatever degree; to some extent; at all: *Do you feel any better?* **9. any which way,** in any manner whatever; indifferently or carelessly: *Doing your work any which way is just not good enough.* [bef. 950; ME *eni, ani,* OE *ænig* (OE *ān* ONE + *-ig* -Y¹)]
—**Syn. 3.** See **some.**
—**Usage.** See **anybody, anyone, anyplace, anyway, either, they.**

An•yang (än′yäng′), *n. Pinyin, Wade-Giles.* a city in N Henan province, in E China: site of the ancient city of Yin, the center of the Shang dynasty, dated c1350–1027 B.C. 225,000. Also, **An′-yang′.**

an•y•bod•y (en′ē bod′ē, -bud′ē), *pron., n., pl.* **-bod•ies.** —*pron.* **1.** any person. **2. anybody's guess,** a matter of conjecture: *It's anybody's guess why she quit.* —*n.* **3.** a person of some importance: *If you're anybody, you'll receive an invitation.* [1250–1300; ME *ani bodi.* See ANY, BODY]
—**Usage.** The pronoun ANYBODY is always written as one word: *Is anybody home? There isn't anybody in the office.* The two-word noun phrase ANY BODY means "any group" (*Any body of students will include a few dissidents*) or "any physical body": *The search continued for a week despite the failure to find any body.* If the word *a* can be substituted for *any* without seriously affecting the meaning, the two-word noun phrase is called for: *a body of students; failure to find a body.* If the substitution cannot be made, the spelling is ANYBODY. ANYBODY is less formal than ANYONE. See also **anyone, each, they.**

an•y•how (en′ē hou′), *adv.* **1.** in any way whatever. **2.** in any case; at all events. **3.** in a careless manner; haphazardly. [1730–40; ANY + HOW]

an•y•more (en′ē môr′, -mōr′), *adv.* **1.** any longer. **2.** nowadays; presently. [1350–1400; ME *ani more* any longer]
—**Usage.** The adverb ANYMORE meaning "any longer" or "nowadays" is most commonly spelled as one word. It is used in negative constructions and in some types of questions: *Sally doesn't work here anymore. Do you play tennis anymore?* In some dialects, chiefly South Midland in origin, it is found in positive statements meaning "nowadays": *Baker's bread is all we eat anymore. Anymore we always take the bus.* Its use at the beginning of a sentence is almost exclusive to speech or to representations of speech.

an•yon (an′yon), *n.* an elementary particle or particle-like excitation having properties intermediate between those of bosons and fermions. [1983; ANY + -ON¹]

an•y•one (en′ē wun′, -wən), *pron.* any person at all; anybody: *Did anyone see the accident?* [1350–1400; ME *ani on.* See ANY, ONE]
—**Usage.** ANYONE as a pronoun meaning "anybody" or "any person at all" is written as one word: *Does anyone have the correct time?* The two-word phrase ANY ONE means "any single member of a group of persons or things" and is often followed by *of*: *Can any one of the members type? Any one of these books is exciting reading.* ANYONE is somewhat more formal than ANYBODY. See also **each, they.**

an•y•place (en′ē plās′), *adv.* anywhere. [1915–20; ANY + PLACE]
—**Usage.** The adverb ANYPLACE is most often written as one word: *Anyplace you look there are ruins.* It occurs mainly in informal speech and only occasionally in writing. ANYWHERE is by far the more common form in formal speech and edited writing. The same holds true, respectively, of the adverbial pairs EVERYPLACE and EVERYWHERE; NOPLACE and NOWHERE; and SOMEPLACE and SOMEWHERE. The two-word noun phrases ANY PLACE, EVERY PLACE, NO PLACE, and SOME PLACE occur, however, in all contexts: *We can build the house in any place we choose. There's no place like home.*

an•y•road (en′ē rōd′), *adv. Brit. Slang.* anyway; anyhow. [1885–90]

an•y•thing (en′ē thing′), *pron.* **1.** any thing whatever; something, no matter what: *Do you have anything for a toothache?* —*n.* **2.** a thing of any kind. **3. anything goes,** any type of conduct, dress, speech, etc., is considered acceptable or valid or is likely to be encountered and tolerated: *That resort is a place where anything goes!* —*adv.* **4.** in any degree; to any extent; in any way; at all: *Does it taste anything like chocolate?* **5. anything but,** in no degree or respect; not in the least: *The plans were anything but definite.* [bef. 900; ME *ani thing, eni thing,* OE *ænig thing.* See ANY, THING¹]

an•y•time (en′ē tīm′), *adv.* **1.** at any time; regardless of hour, date, etc.; whenever. **2.** invariably; without doubt or exception; always: *I can do better than that anytime.* [1780–90; ANY + TIME]

an•y•way (en′ē wā′), *adv.* **1.** in any case; anyhow; nonetheless; regardless: *Whether you like it or not, I'm going anyway.* **2.** (used to continue or resume the thread of a story or account): *Anyway, we finally found a plumber who could come right over.* [1150–1200; ME *ani wei.* See ANY, WAY]
—**Usage.** The adverb ANYWAY is spelled as one word: *It was snowing hard, but we drove to the play anyway.* The two-word phrase ANY WAY means "in any manner": *Finish the job any way you choose.* If the words "in the" can be substituted for "any," the two-word phrase is called for: *Finish the job in the way you choose.* If the substitution cannot be made, the spelling is ANYWAY.

an•y•ways (en′ē wāz′), *adv. Nonstandard.* anyway.

an•y•where (en′ē hwâr′, -wâr′), *adv.* **1.** in, at, or to any place. **2.** to any extent; to some degree: *Does my answer come anywhere near the right one?* **3. get anywhere,** to achieve success: *You'll never get anywhere with that attitude!* —*n.* **4.** any place or direction: *They knew the attack could come from anywhere.* [1350–1400; ME *anywher(e), aniquar.* See ANY, WHERE]
—**Usage.** See **anyplace.**

an•y•wheres (en′ē hwârz′, -wârz′), *adv. Nonstandard.* anywhere. [1765–75; ANYWHERE + -S¹]

an•y•wise (en′ē wīz′), *adv.* in any way or respect. [bef. 1000; ME *ani wise,* OE *on ænige wīsan* in any wise. See ANY, WISE²]

An•za (än′sä), *n.* **Juan Bau•tis•ta de** (hwän bou tēs′tä thē), 1735–88, Spanish frontiersman and army officer, born in Mexico: explored western coast of U.S.

An•zac (an′zak), *n.* **1.** a member of the Australian and New Zealand Army Corps during World War I. **2.** a soldier from Australia or New Zealand. **3.** any Australian or New Zealander. [1910–15]

An′zac Day′, April 25, a national day of remembrance in Australia and New Zealand commemorating the Anzac landing on Gallipoli, Turkey, in 1915, the first major engagement of Australian and New Zealand forces in World War I.

An•zen•gru•ber (än′tsən grōō′bər), *n.* **Lud•wig** (lōōt′vikh, lōōd′-), 1839–89, Austrian playwright and novelist.

An•zhe•ro-Su•dzhensk (un zhe′rə sōō′jinsk), *n.* a city in the S Russian Federation in central Asia. 105,000.

An•zi•o (an′zē ō′; *It.* än′tsyô), *n.* a town in Italy, S of Rome on the Tyrrhenian coast: site of Allied beachhead in World War II. 23,421.

ANZUS (an′zəs), *n.* Australia, New Zealand, and the United States, esp. as associated in the mutual defense treaty (**ANZUS Pact** or **ANZUS Treaty**) of 1952.

A/O, **1.** account of. **2.** and others. Also, **a/o**

AOA, Administration on Aging.

ao dai (ou′ dī′, ô′ dī′), *n., pl.* **ao dais.** a costume composed of a long tunic with side slits over wide trousers, worn by Vietnamese women as traditional dress. [< Vietnamese *áo dài* lit., long garment]

A•oe•de (ā ē′dē), *n. Class. Myth.* one of the original three Muses; the Muse of song. Also, **A•oi•de** (ā oi′dē). Cf. **Melete, mneme** (def. 2).

AOH, Ancient Order of Hibernians.

A-OK (ā′ō kā′), *adj., adv. Informal.* OK; perfect: *an A-OK rocket launching.* Also, **A-O.K., A′-O-kay′.** [1955–60]

A•o•ki (ä ō′kē), *n.* **Shu•zo** (shōō′zô), 1844–1914, first Japanese ambassador to U.S., 1905–09.

a•o•le (ä ō′lā), *adv. Hawaiian.* no; not at all. [< Hawaiian *'a'ole*]

A•o•mo•ri (ä′ô mô′rē), *n.* a seaport on N Honshu, in N Japan. 287,609.

A one (ā′ wun′), **1.** noting a vessel regarded by a shipping-classification society as being equipped to the highest standard and with equipment maintained in first-class condition. **2.** Also, **A number one.** *Informal.* first-class; excellent; superior: *The meals there are A one.* Also, **A-one, A 1, A-1** [1830–40]

AOR, **1.** *Law.* advice of rights. **2.** Album-Oriented Radio. **3.** Album-Oriented Rock.

A•o•ran•gi (ä′ō räng′gē), *n.* See **Cook, Mount.**

a•o•rist (ā′ə rist), *Gram.* —*n.* **1.** a verb tense, as in Classical Greek, expressing action or, in the indicative mood, past action, without further limitation or implication. —*adj.* **2.** of or in this tense. [1575–85; < Gk *aóristos* unlimited, equiv. to *a-* A-⁶ + *(h)oristós* limited (*horid-* (base of *horizein* to bound, limit; see HORIZON) + *-tos* adj. suffix)]

a•o•ris•tic (ā′ə ris′tik), *adj.* **1.** *Gram.* pertaining to the aorist. **2.** indefinite; indeterminate. [1840–50; AORIST + -IC] —**a′o•ris′ti•cal•ly,** *adv.*

a•or•ta (ā ôr′tə), *n., pl.* **-tas, -tae** (-tē). *Anat.* the main trunk of the arterial system, conveying blood from the left ventricle of the heart to all of the body except the lungs. See illus. under **heart.** [1570–80; < ML < Gk *aortḗ* the great artery, lit., something hung, carried; akin to *aeírein* to lift, carry] —**a•or′tic, a•or′tal,** *adj.*

aor′tic arch′, *Embryol.* one member of a series of paired curved blood vessels that arise in the embryo from the ventral aorta, pass around the pharynx through the branchial arches, and join with the dorsal aorta to form the great vessels of the head and neck. [1900–05]

aor′tic insuffi′ciency, *Pathol.* abnormal closure of the aortic valve resulting in regurgitation of blood to the left ventricle. Also called **aor′tic incom′petence.**

aor′tic steno′sis, *Pathol.* abnormal narrowing of the aorta, esp. of its orifice, usually as a result of rheumatic fever or embryologic anomalies.

aor′tic valve′, *Anat.* a semilunar valve between the aorta and the left ventricle of the heart that prevents the blood from flowing back into the left ventricle.

a•or•ti•tis (ā′ôr tī′tis), *n. Pathol.* inflammation of the aorta. [1835–45; < NL; see AORTA, -ITIS]

a•or•to•cla•sia (ā ôr′tə klā′zhə, -zhē ə), *n. Pathol.* rupture of the aorta. [AORT(A) + -O- + *-clasia* < Gk *klâs(is)* a breaking (see -CLASIS) + -IA]

a•or•tog•ra•phy (ā′ôr tog′rə fē), *n., pl.* **-phies.** x-ray examination of the aorta following injection of radiopaque dye. [1930–35; AORT(A) + -O- + -GRAPHY] —**a•or•to•graph•ic** (ā ôr′tə graf′ik), *adj.*

A.O.U., American Ornithologists' Union.

a•ou•dad (ä′ōō dad′), *n.* a wild sheep, *Ammotragus lervia,* of northern Africa, having a long fringe of hair on the throat, chest, and forelegs. Also called **Barbary sheep.** [1860–65; < F < Berber, equiv. to *a-* masc. sing. prefix + *udad* ram]

AP, **1.** See **adjective phrase.** **2.** *Education.* Advanced Placement. **3.** See **Air Police.** **4.** See **American plan.** **5.** antipersonnel. Also, **A.P.**

ap-¹, var. of **ad-** before *p: appear.*

ap-², var. of **apo-** before a vowel or *h: aphelion.*

Ap., **1.** Apostle. **2.** Apothecaries'. **3.** April.

A/P, **1.** account paid. **2.** accounts payable. **3.** authority to pay or purchase. Also, **a/p**

a.p., **1.** additional premium. **2.** author's proof.

a•pa (ə pä′), *n.* a tree, *Eperua falcata,* of tropical America, having reddish-brown wood used in the construction of houses. [< Tupi]

A•pa (ä′pä), *n.* a river in central South America that flows W along the E Paraguay-SW Brazil border to the Paraguay River. ab. 125 mi. (200 km) long.

APA, **1.** American Psychiatric Association. **2.** American Psychological Association.

A.P.A., **1.** American Philological Association. **2.** American Protective Association. **3.** American Protestant Association. **4.** American Psychiatric Association. **5.** American Psychological Association. **6.** Associate in Public Administration.

a•pace (ə pās′), *adv.* with speed; quickly; swiftly. [1275–1325; ME *a pas(e)* at a (good) pace. See A-¹, PACE]

a•pache (ə päsh′, ə pash′; *Fr.* A PASH′), *n., pl.* **a•paches** (ə pä′shiz, ə pash′iz; *Fr.* A PASH′). a Parisian gangster, rowdy, or ruffian. [1735–45, *Amer.*; < F: APACHE]

A•pach•e (ə pach′ē), *n., pl.* **A•pach•es,** (*esp. collectively*) **A•pach•e.** **1.** a member of an Athabaskan people of the southwestern U.S. **2.** any of the several Athabaskan languages of Arizona and the Rio Grande basin. **3.** *Mil.* a two-man U.S. Army helicopter designed to attack enemy armor with rockets or a 30mm gun and equipped for use in bad weather and in darkness. [1915–20; < MexSp, perh. < Zuni *ʔa·paču* Navajos, presumably applied formerly to the Apacheans (Navajos and Apaches) generally]

A•pach•e•an (ə pach′ē ən), *n.* **1.** a subgroup of the Athabaskan language family comprising the languages of the Apache tribes and the Navajo. **2.** a speaker of an Apachean language. [APACHE + -AN]

a•pache′ dance′ (ə päsh′, ə pash′ *or, often,* ə pach′ē), a vigorous dance representing a Parisian apache dancing in a rough, domineering way with his woman.

Apach′e plume′, a low shrub, *Fallugia paradoxa,* of the rose family, of southwestern North America, having white flowers and fruit heads of feathery tufts. [1885–90, *Amer.*]

Ap′a•lach′ee Bay′ (ap′ə lach′ē, ap′-), a bay of the Gulf of Mexico, on the coast of NW Florida. ab. 30 mi. (48 km) wide.

Ap•a•lach•i•co•la (ap′ə lach′ə kō′lə), *n.* a river flowing S from NW Florida into the Gulf of Mexico. 90 mi. (145 km) long.

ap•a•nage (ap′ə nij), *n.* appanage.

CONCISE ETYMOLOGY KEY: <, descended or borrowed from; >, whence; b., blend of, blended; c., cognate with; cf., compare; deriv., derivative; equiv., equivalent; imit., imitative; obl., oblique; r., replacing; s., stem; sp., spelling, spelled; resp., respelling, respelled; trans., translation; ?, origin unknown; *, unattested; ‡, probably earlier than. See the full key inside the front cover.