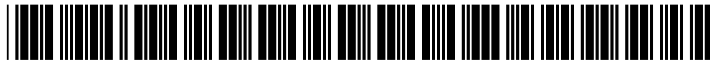



(19) **United States**
(12) **Patent Application Publication**
**Ogier et al.**
(10) **Pub. No.: US 2002/0062388 A1**
(43) **Pub. Date: May 23, 2002**

(54) **SYSTEM AND METHOD FOR DISSEMINATING TOPOLOGY AND LINK-STATE INFORMATION TO ROUTING NODES IN A MOBILE AD HOC NETWORK**

(76) Inventors: **Richard G. Ogier**, Half Moon Bay, CA (US); **Bhargav R. Bellur**, Fremont, CA (US); **Fred Lambert Templin**, Portola Valley, CA (US)

Correspondence Address:
**TESTA, HURWITZ & THIBEAULT, LLP**
**HIGH STREET TOWER**
**125 HIGH STREET**
**BOSTON, MA 02110 (US)**

(21) Appl. No.: **09/728,020**

(22) Filed: **Dec. 1, 2000**

**Related U.S. Application Data**

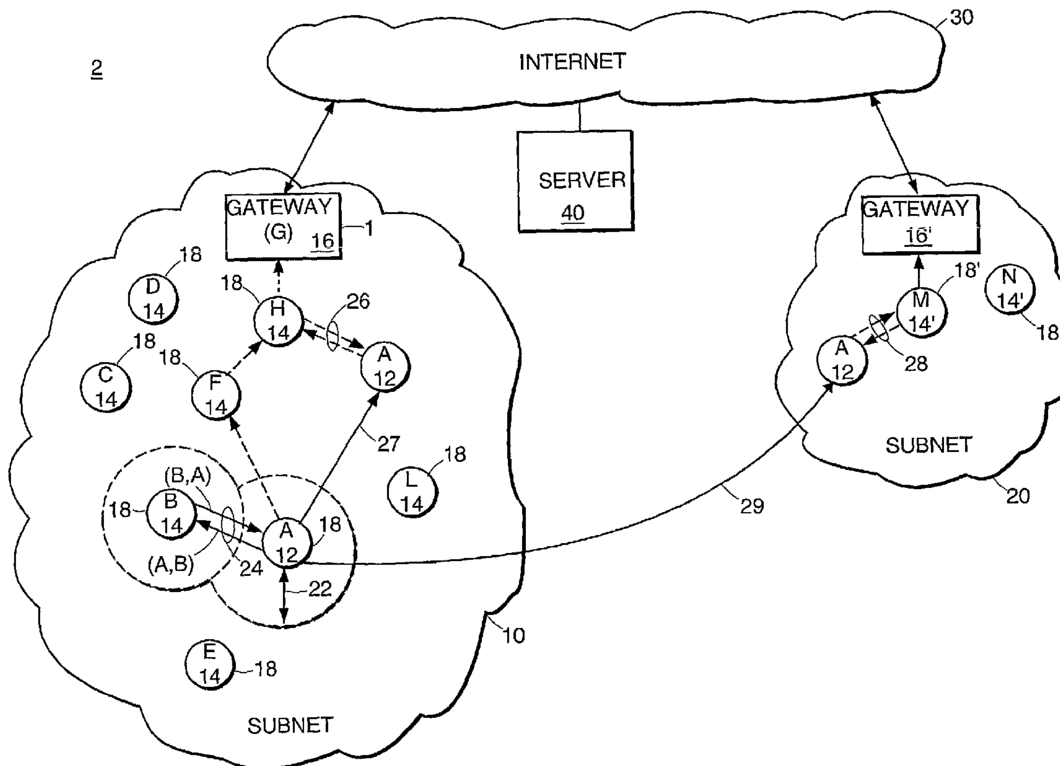
(63) Non-provisional of provisional application No. 60/232,047, filed on Sep. 12, 2000.

**Publication Classification**

(51) **Int. Cl.**$^{7}$ .......................... **G06F 15/16**; G06F 15/173
(52) **U.S. Cl.** .......................................... **709/238**; 709/203

(57) **ABSTRACT**

Described is a link-state routing protocol used in a mobile ad hoc network or in an Internet for disseminating topology and link-state information throughout the network. Reverse-path forwarding is used to broadcast each update along the minimum-hop-path tree rooted at the source of the update. Each path tree has the source node as a root node, a parent node, and zero or more children nodes. Updates are received from the parent node in the path tree for the source node that originates the update. Each update includes information related to a link in the network. A determination is made whether to forward the update message to children nodes, if any, in the path tree maintained for the source node originating the update in response to information in the received update. This information itself can indicate whether the update is to be forwarded to other nodes.




**IPR2025-01304**
**Cisco Systems, Inc. v. Dynamic Mesh Networks**
**Cisco's Exhibit 1003**
**EX1003, Page 1**

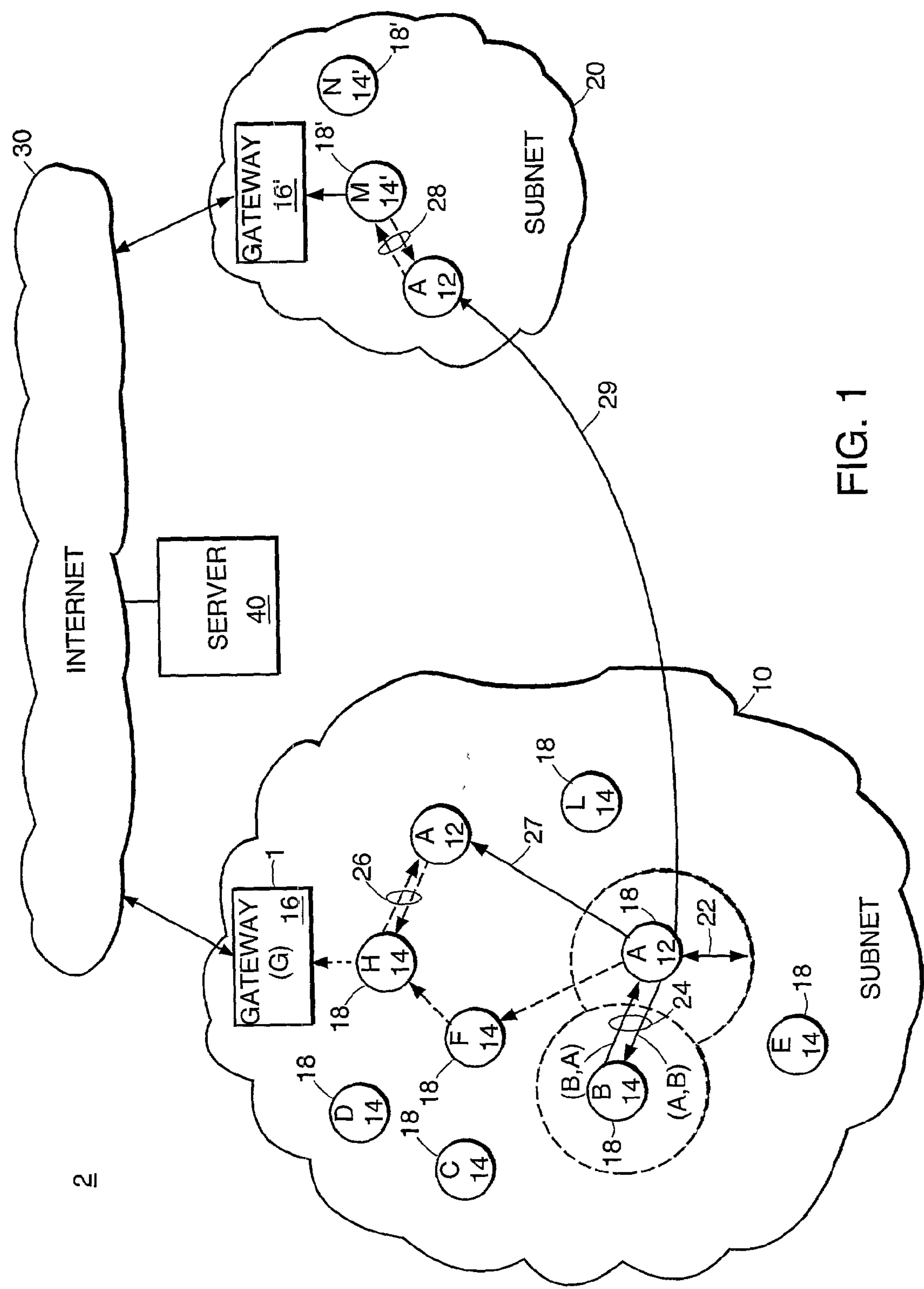



FIG. 1

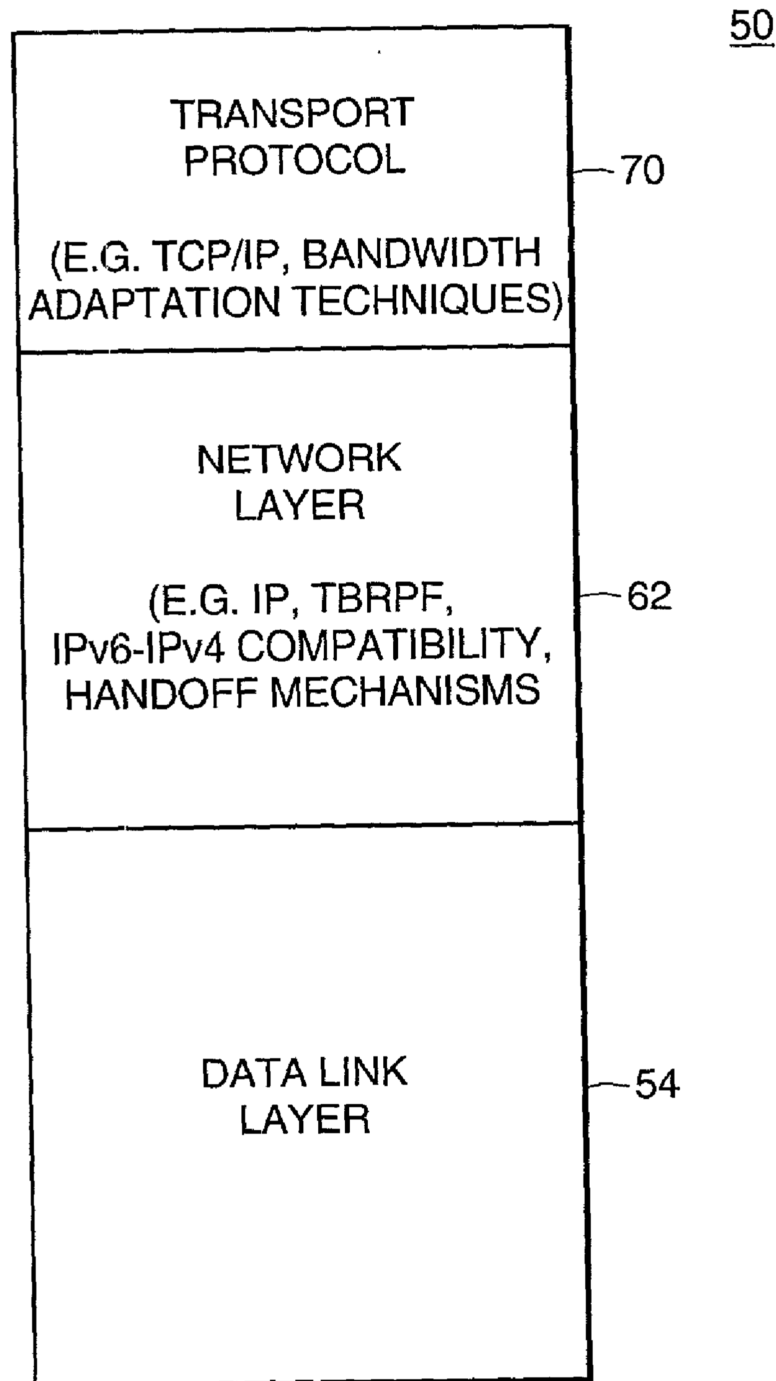



FIG. 2

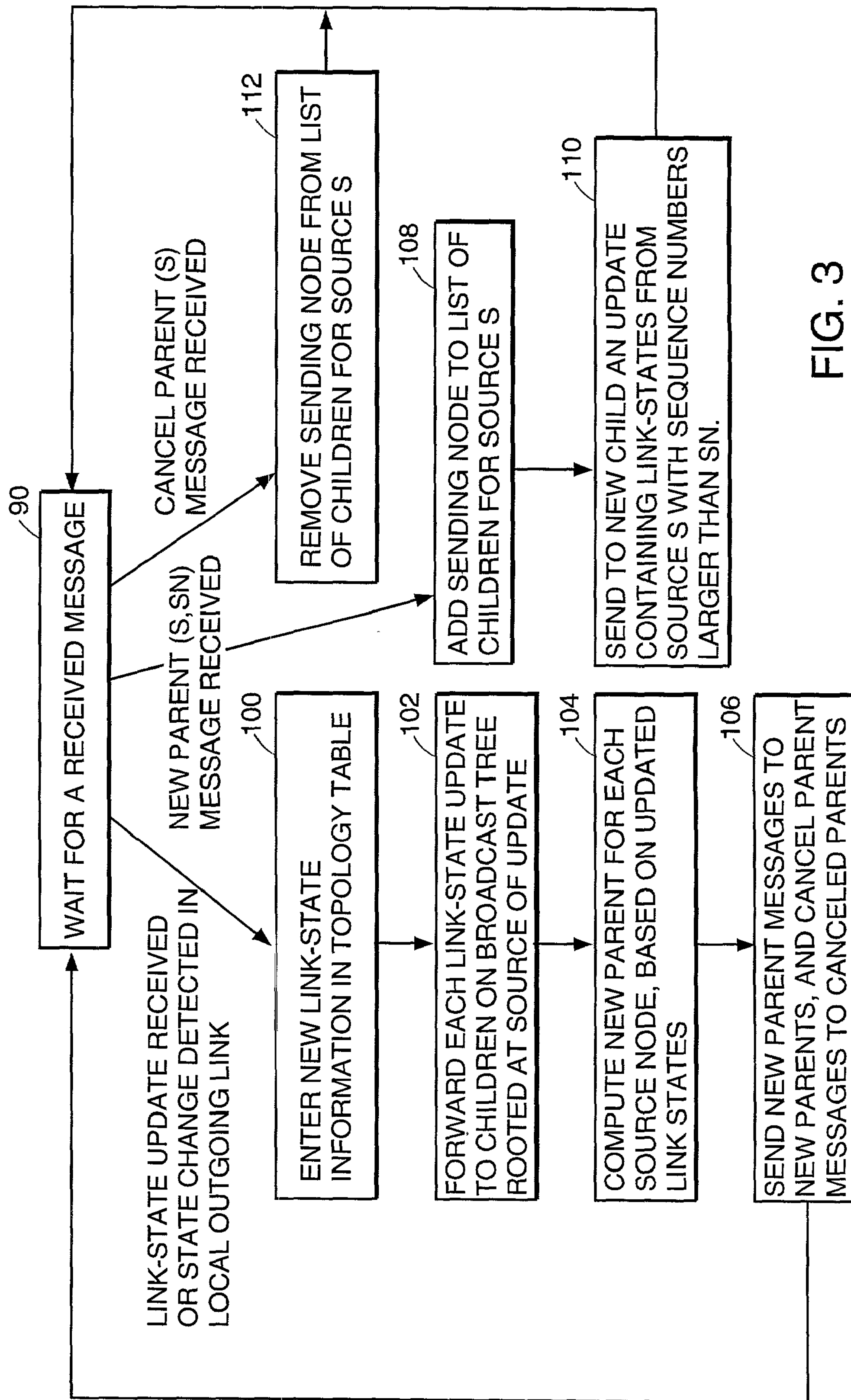



FIG. 3
TBRPF STEPS FOR LINK-STATE DISSEMINATION (NOT INCLUDING NEIGHBOR DISCOVERY)

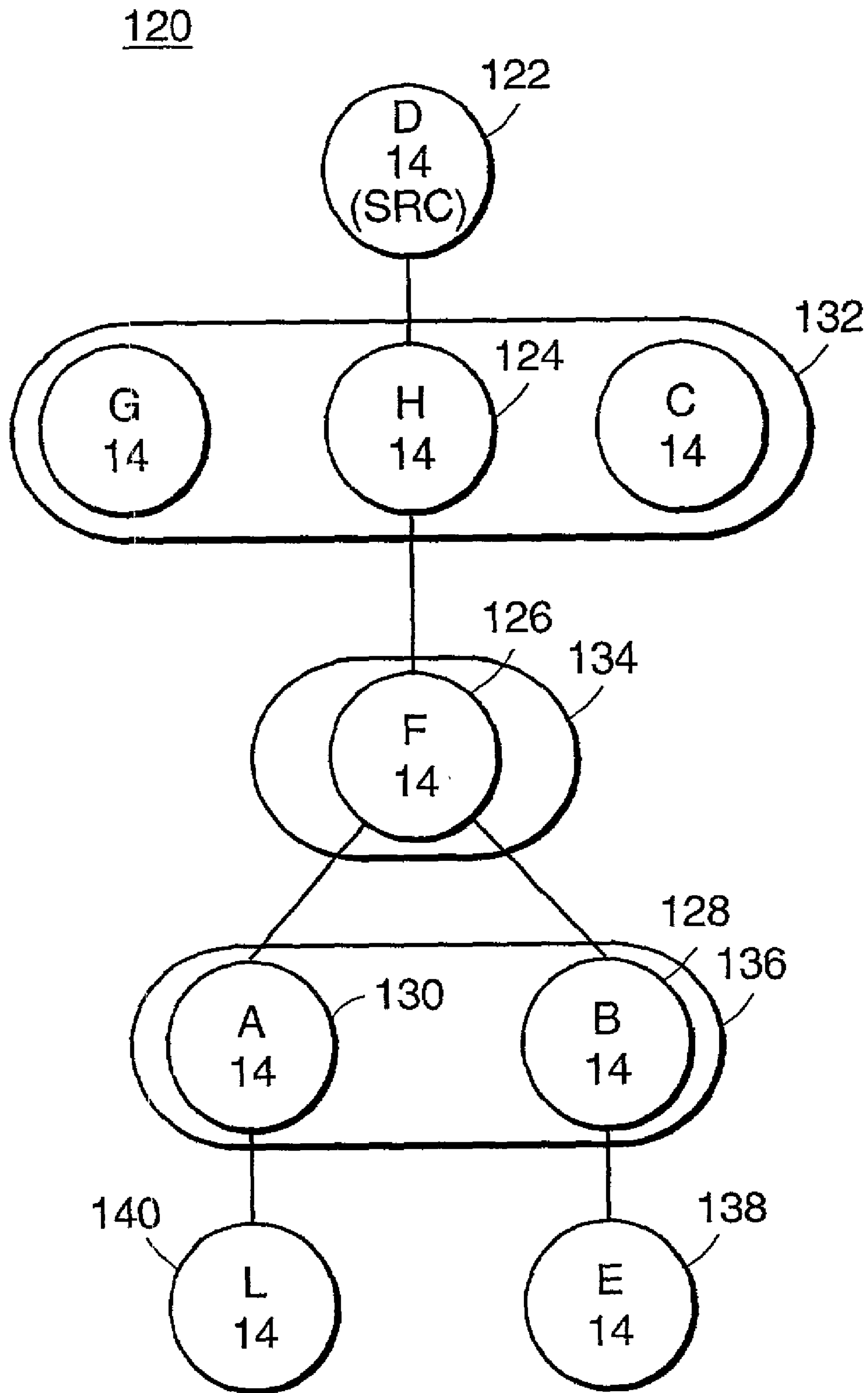



FIG. 4

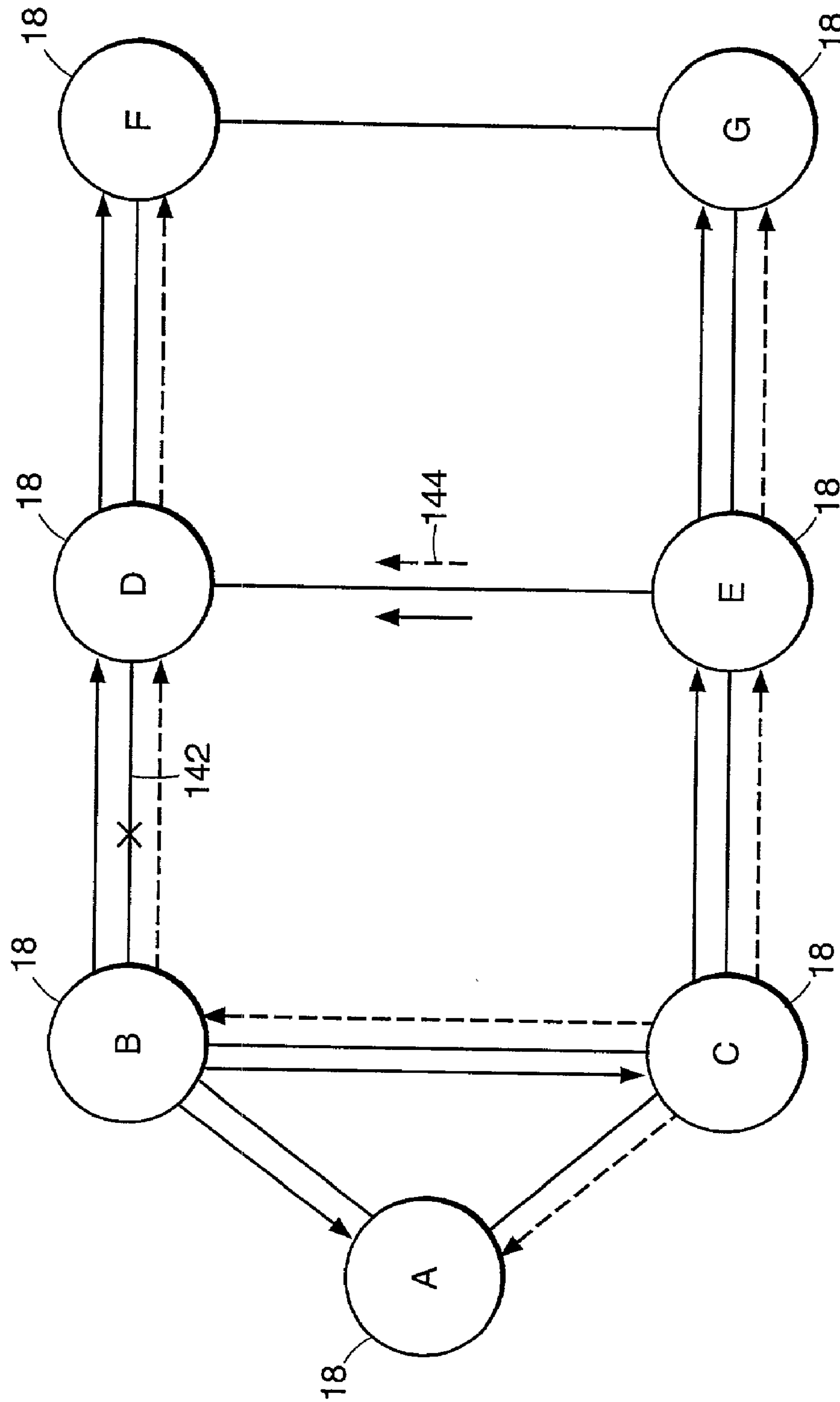
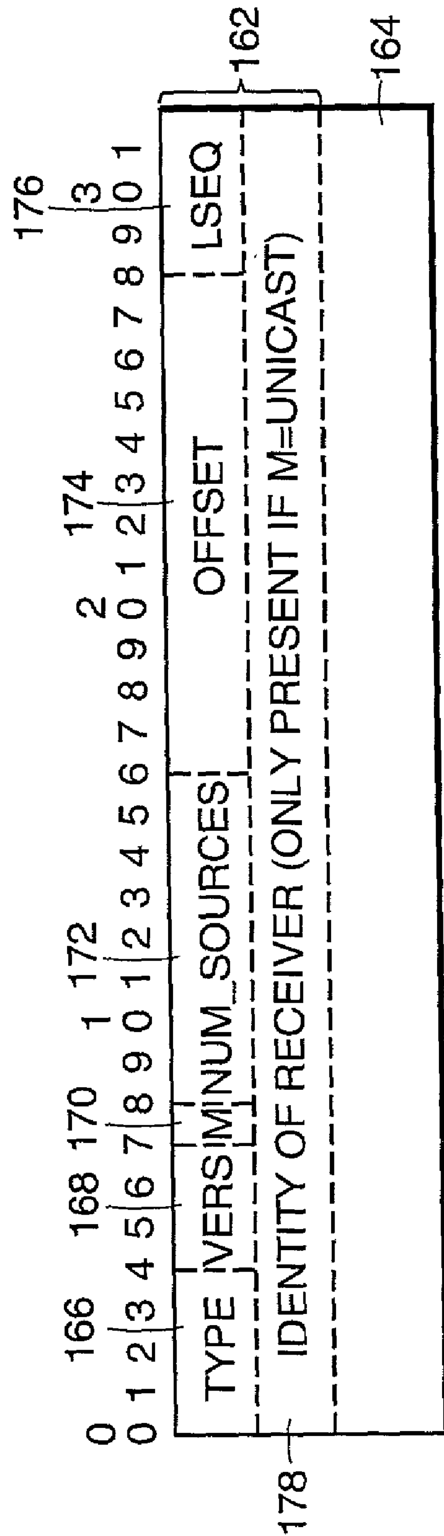



FIG. 5




TBRPF MESSAGE HEADER

FIG. 6

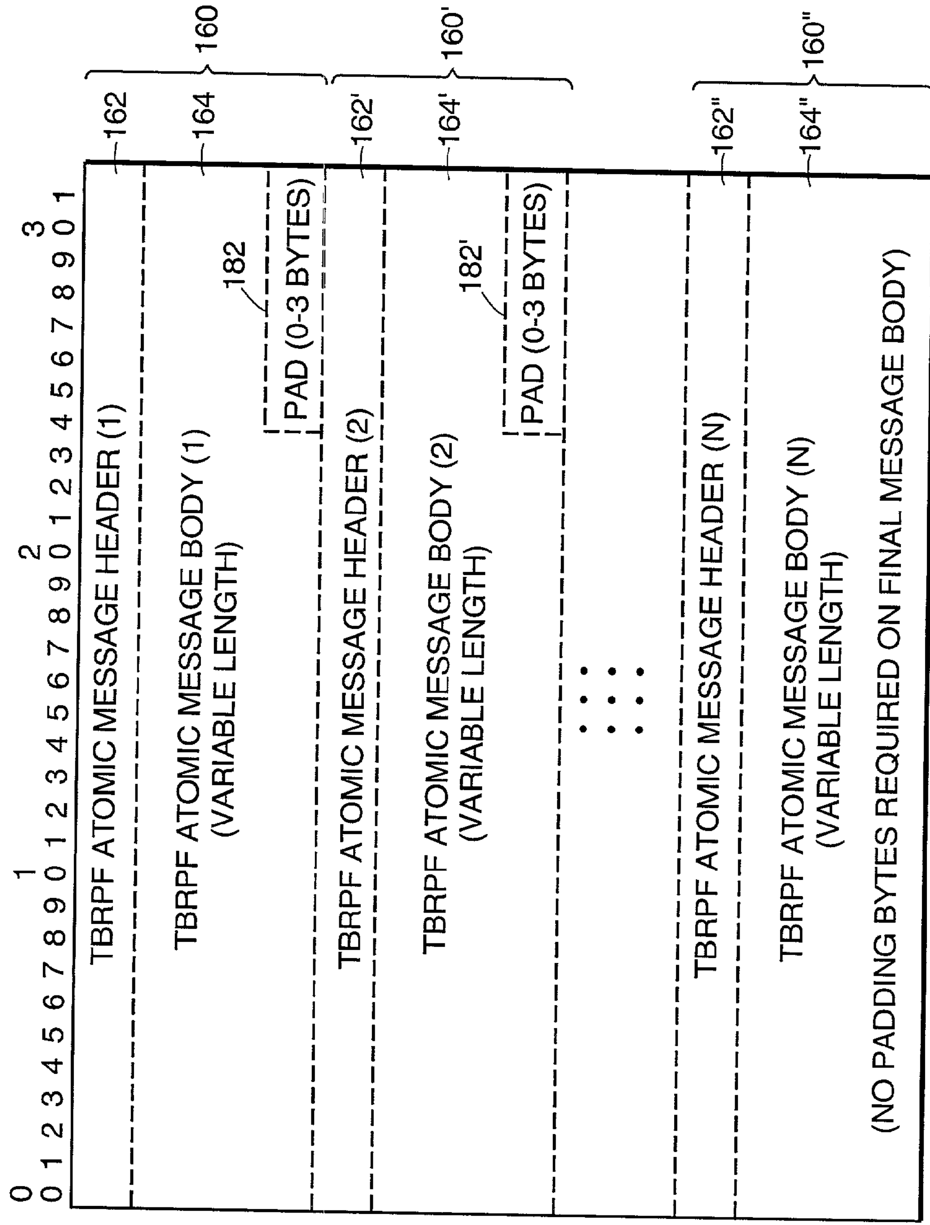
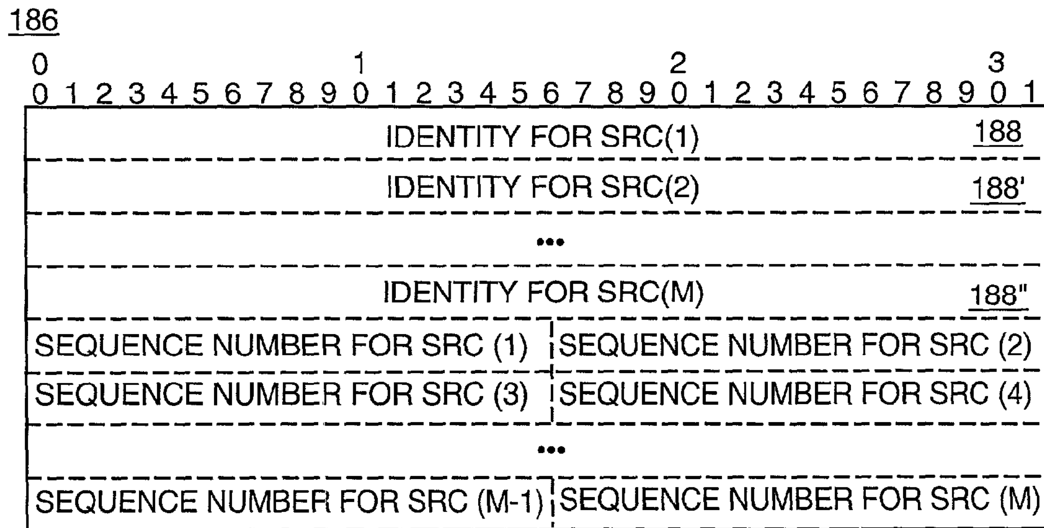
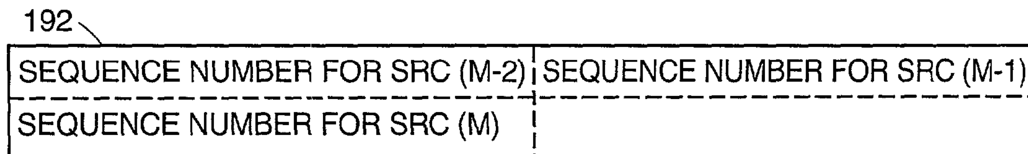



FIG. 7



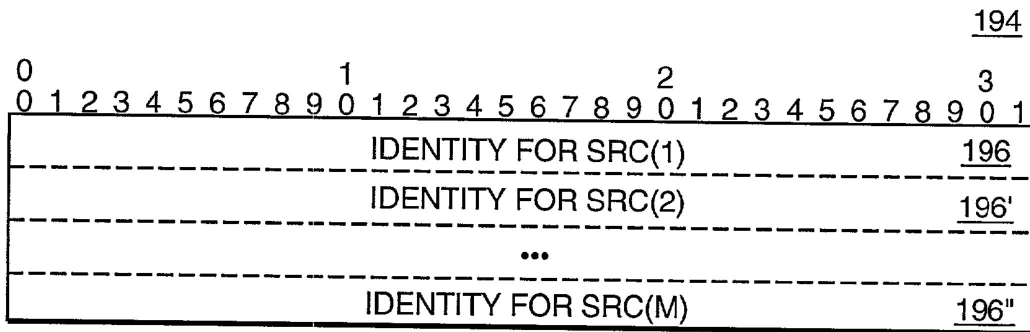

NEW PARENT MESSAGE

FIG. 8A




SHOWS AN ALTERNATIVE ENDING THE NEW PARENT MESSAGE

FIG. 8B




CANCEL_PARENT MESSAGE

FIG. 9

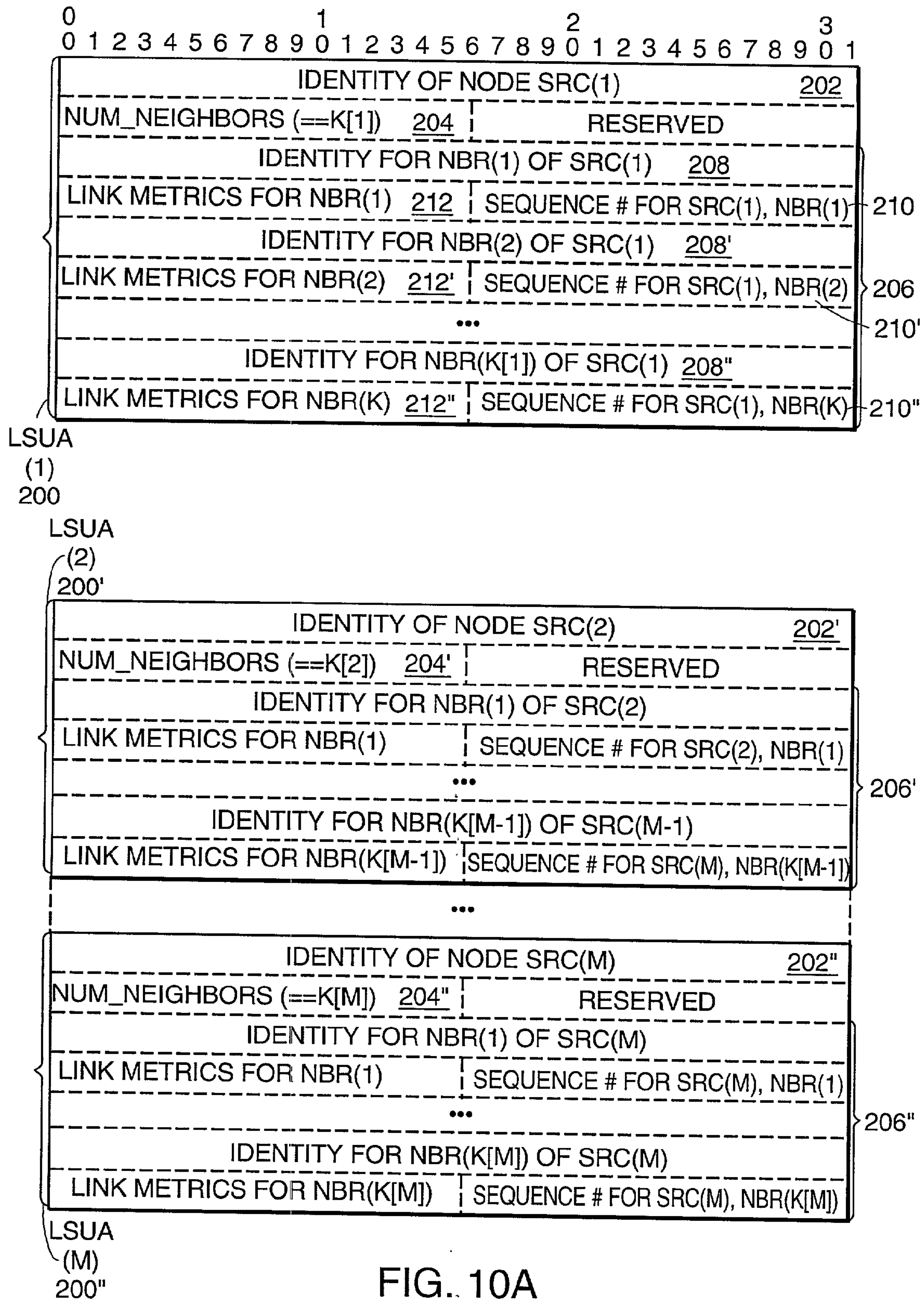



FIG. 10A

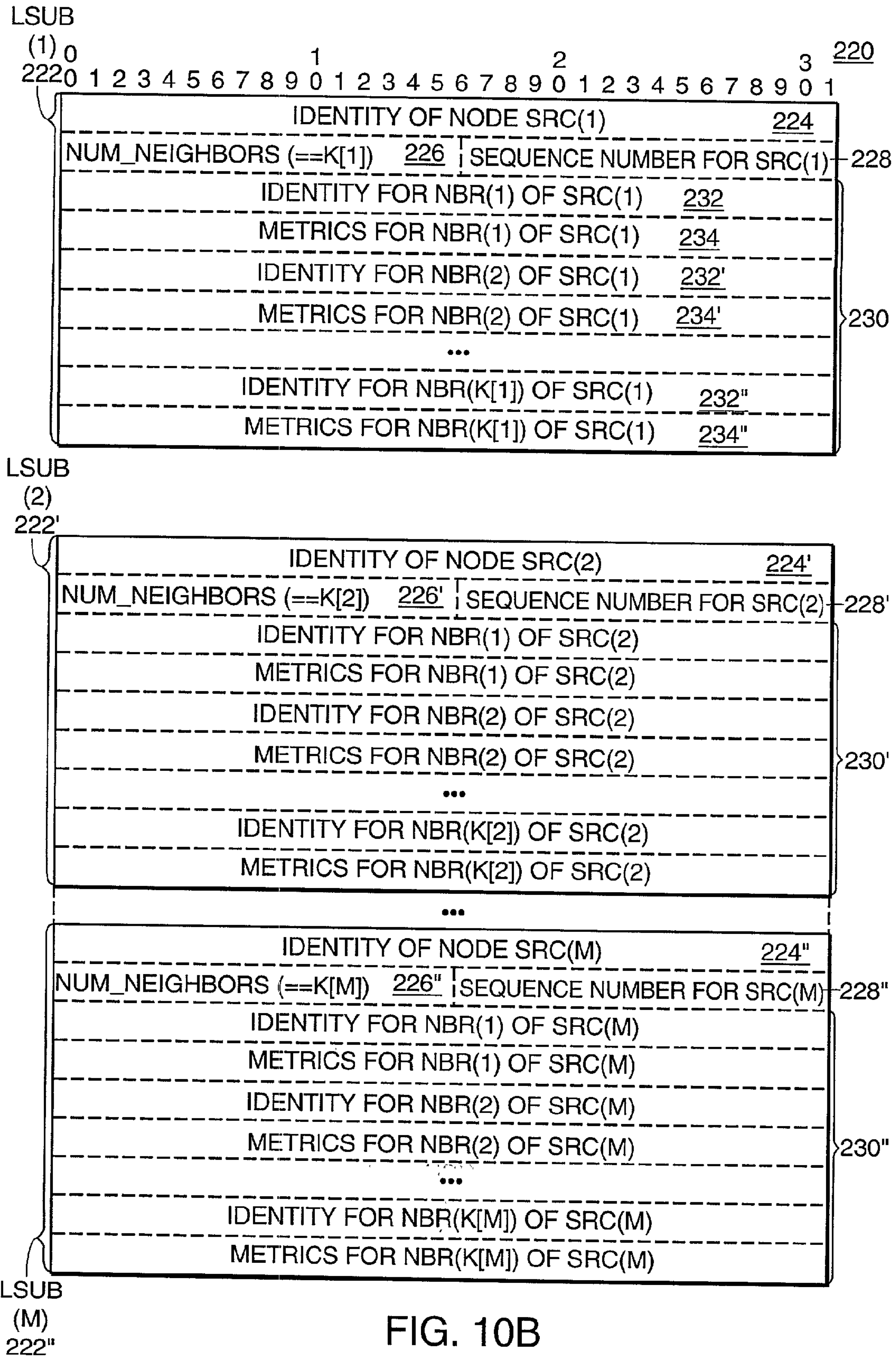
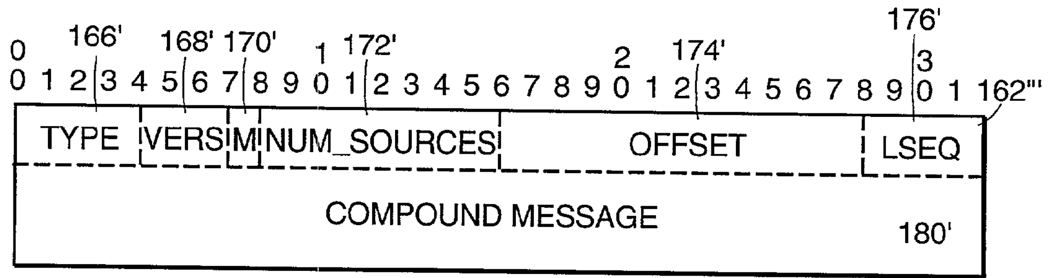



FIG. 10B




RETRANSMISSION_OF_BROADCAST MESSAGE

FIG. 11

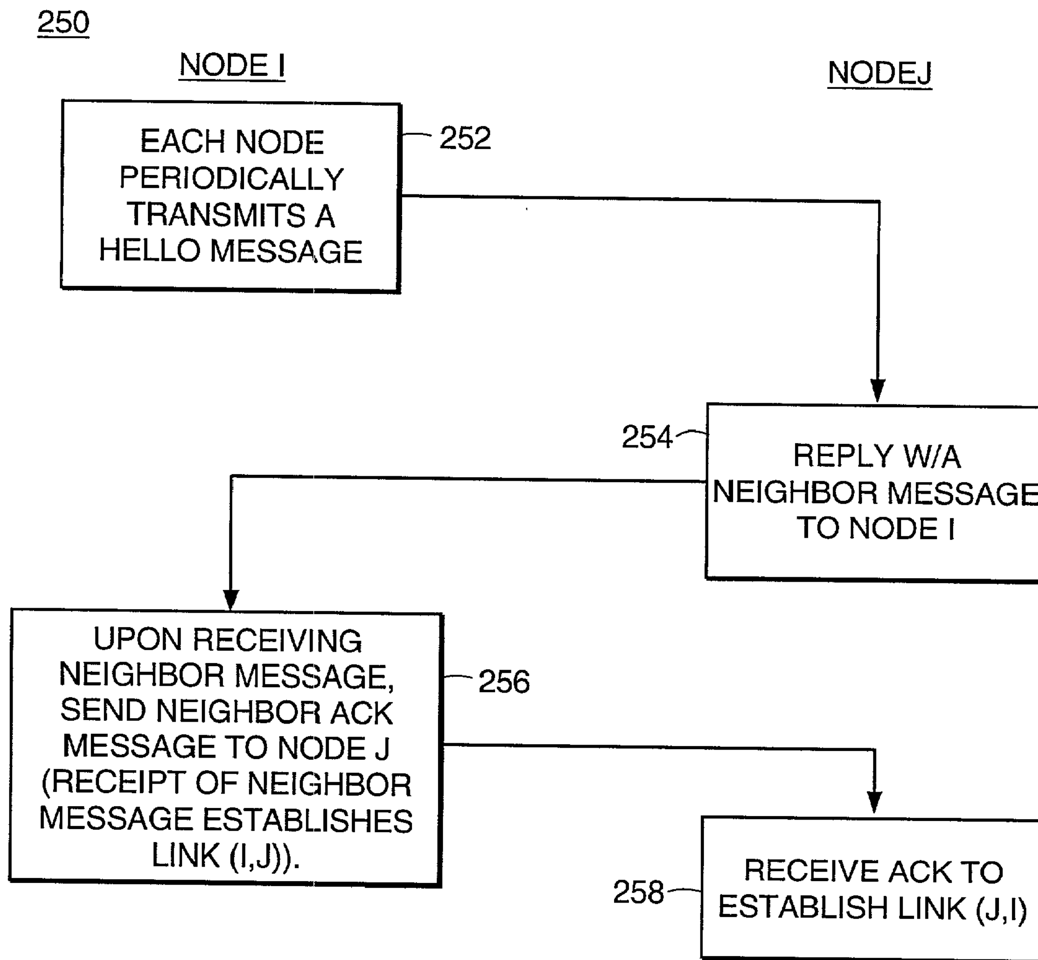



FIG. 12

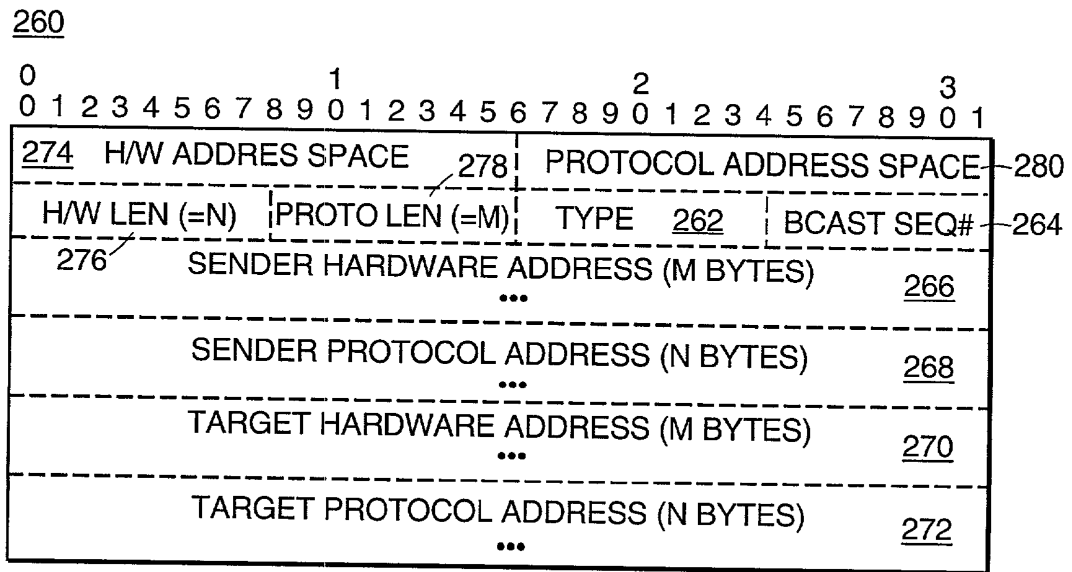



FIG. 13

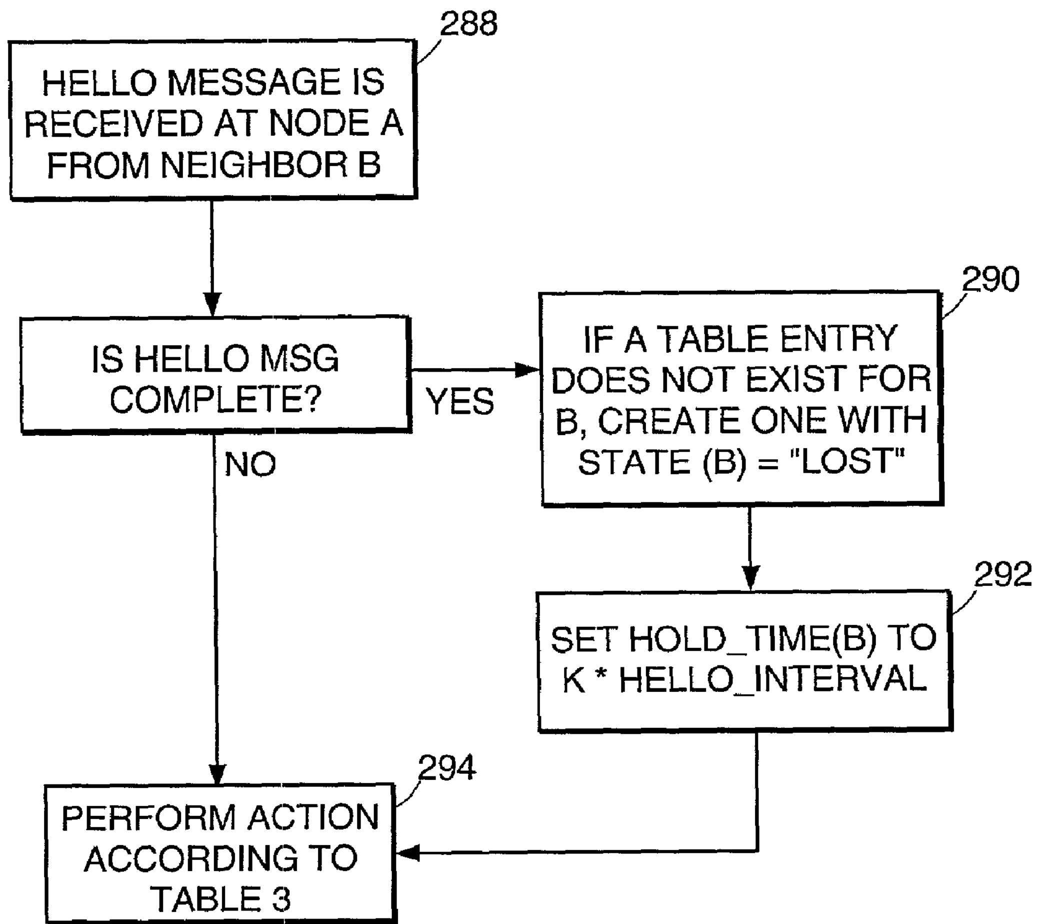



FIG. 14

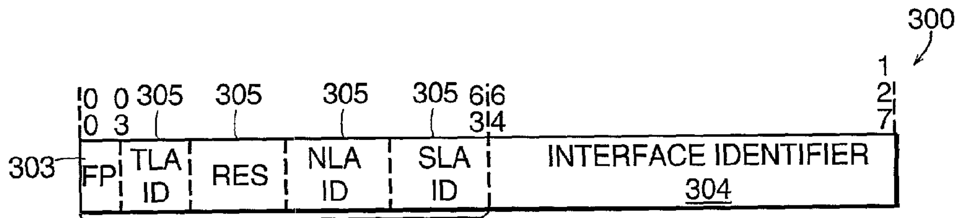



FIG. 15A
IPv6-IPv4 COMPATIBILITY ADDRESS FORMAT

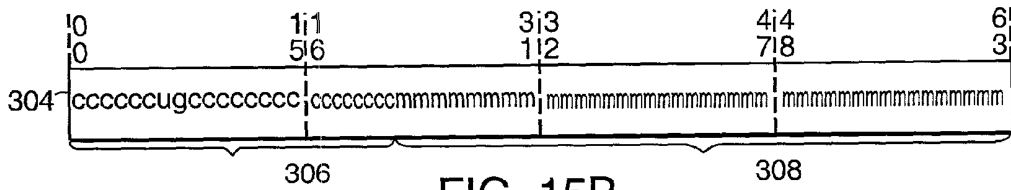



FIG. 15B
INTERFACE IDENTIFIER FORMAT

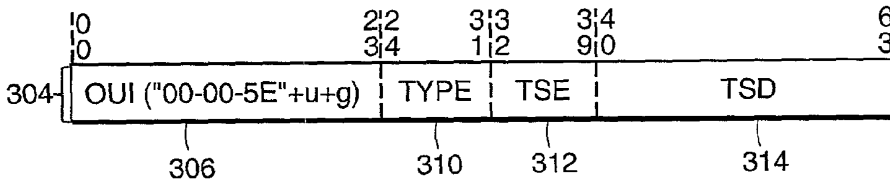



FIG. 15C
INTERFACE IDENTIFIER FORMAT FOR IANA

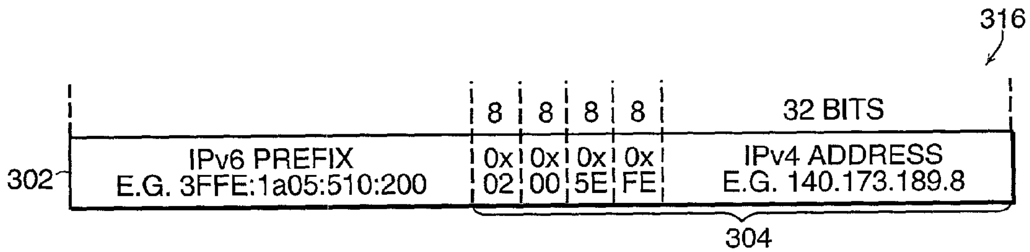



FIG. 15D
IPv6-IPv4 COMPATIBILITY ADDRESS

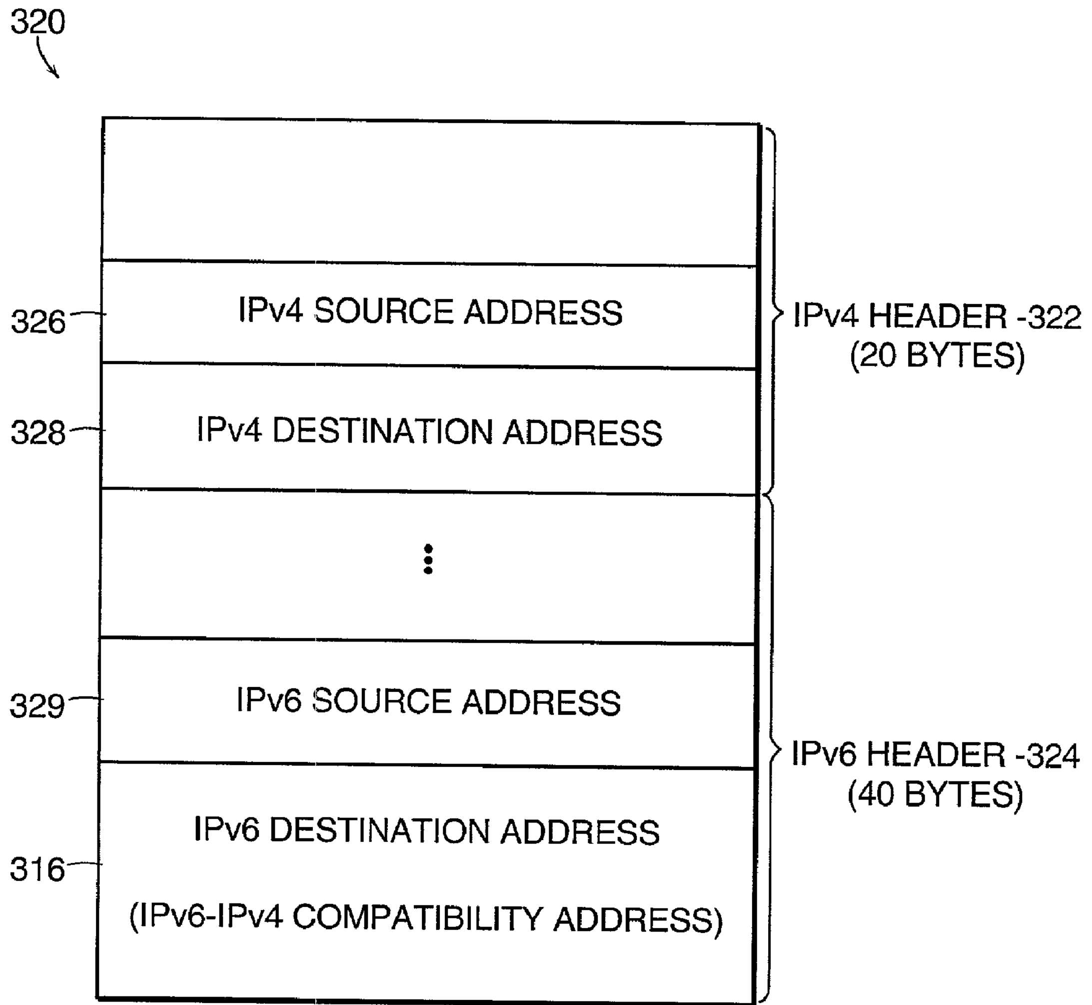



FIG. 15E

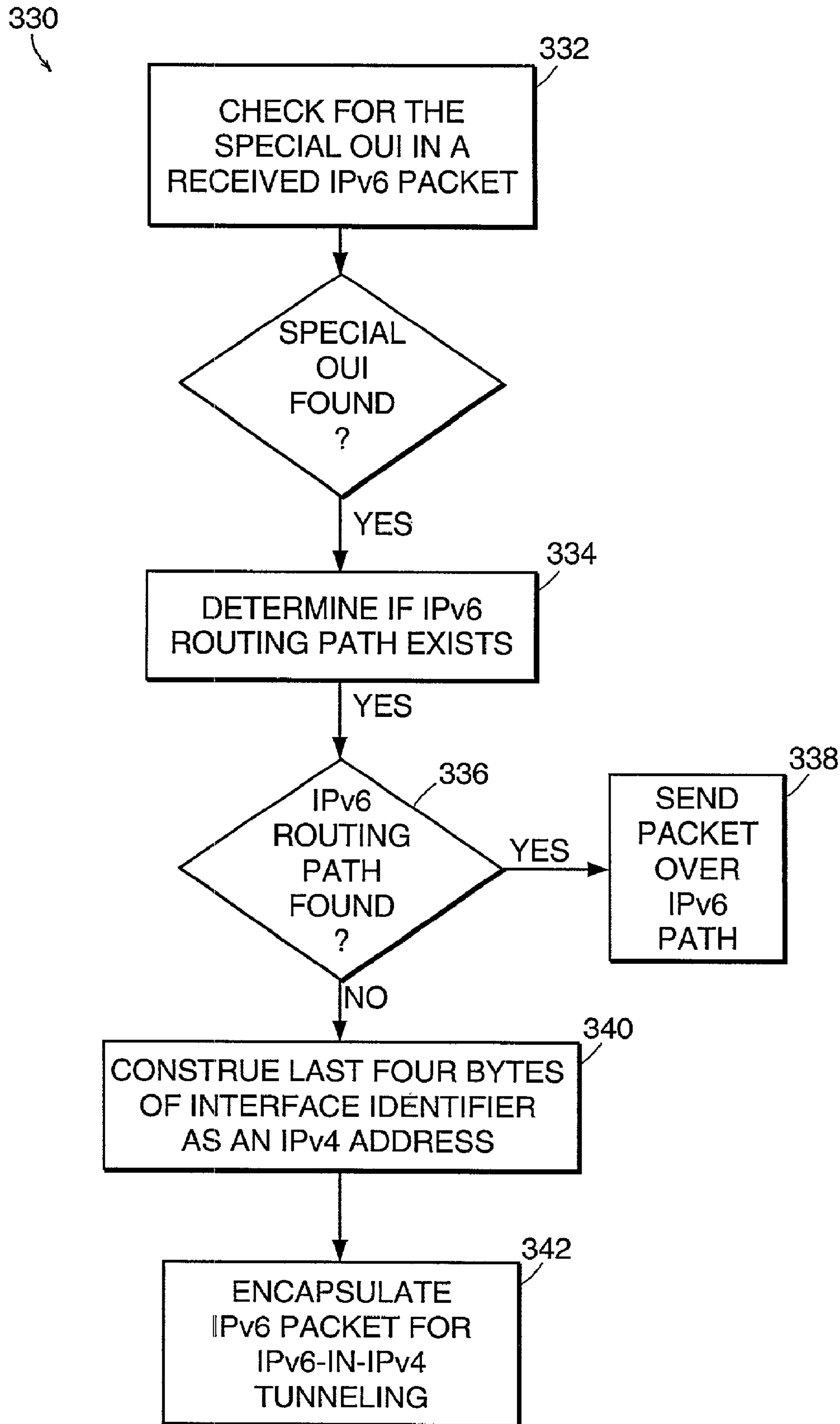



FIG. 16

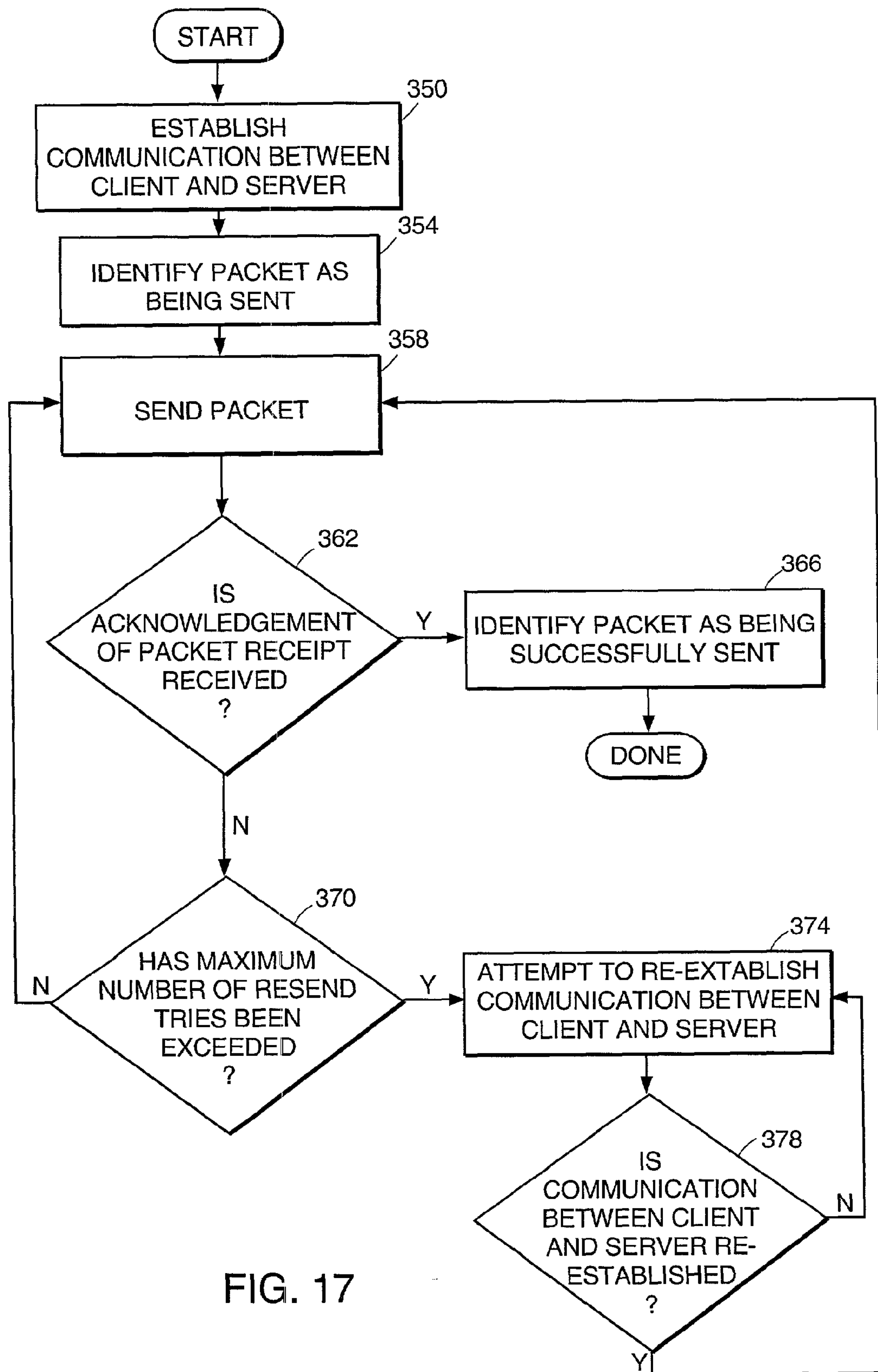



FIG. 17

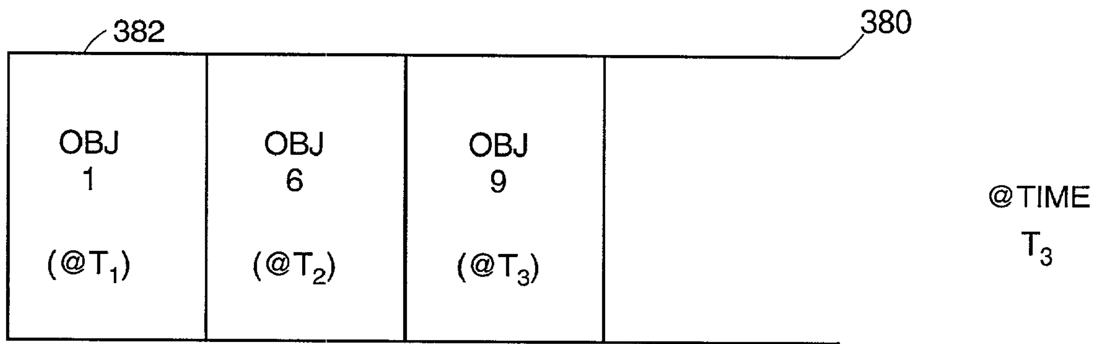



FIG. 18A

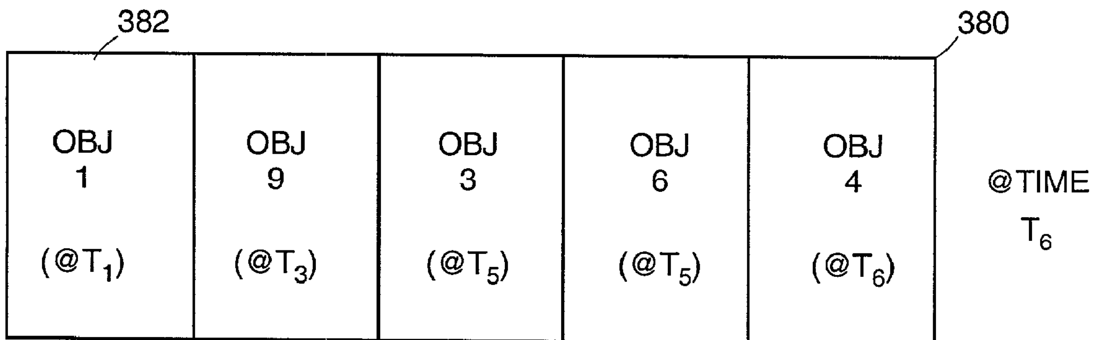



FIG. 18B

# SYSTEM AND METHOD FOR DISSEMINATING TOPOLOGY AND LINK-STATE INFORMATION TO ROUTING NODES IN A MOBILE AD HOC NETWORK

## RELATED APPLICATIONS

[0001] This application claims the benefit of the filing date of co-pending U.S. Provisional Application, Serial No. 60/232,047, filed Sep. 12, 2000, entitled "Techniques for Improved Topology Broadcast Based on Reverse-Path Forwarding" and co-pending U.S. Provisional Application, Seroal No. ______, filed Nov. 14, 2000, entitled "Efficient Routing Protocols for Packet-Radio Networks Based on Tree Sharing", the entirety of which provisional applications is incorporated by reference herein.

## BACKGROUND

[0002] A network is a collection of communications entities (e.g., hosts, routers, and gateways) that are in communication with each other over communication links. Organizing communications entities into networks increases the capabilities of the communication entities beyond that which each communications entity alone is capable by enabling such entities to share resources. A network that interconnects communications entities within a common geographical area (for example, the personal computers in an office) is called a local area network (LAN). Some LANs employ one or more network servers that direct the flow of data within the network and control access to certain network functions such as storing data in a central file repository, printing, accessing other networks. In other LANs, computers communicate with each other without the use of servers.

[0003] A wide area network (WAN), of which the Internet is an example, is a collection of geographically distributed LANs joined by long-range communication links. The Internet is a publicly accessible, worldwide network of networks based upon a transmission protocol known as TCP/IP (Transmission Control Protocol/Internet Protocol). Communications on the Internet is packet-switched; that is, the information that is to pass from one communications entity to another is broken into packets that are individually passed from router to router until the packets arrive at their destination. The TCP divides the data into segments and provides reliable delivery of bytes in the segments to the destination, which reconstructs the data. The IP further subdivides the TCP segments into packets and routes the packets to their final destination. The route taken by packets may pass through one or more networks, depending upon the Internet Protocol (IP) address of the destination.

[0004] A rapidly growing part of the Internet is the World Wide Web ("Web"), which operates according to a client-server model. Client software, commonly referred to as a Web browser, runs on a computer system. After establishing an Internet connection, the client user launches the Web browser to communicate with a Web server on the Internet. Using TCP/IP, the Web browser sends HTTP (Hypertext Transport Protocol) requests to the Web server. The request traverses the Internet's TCP/IP infrastructure to Web host server as HTTP packets.

[0005] A private network based on Internet technology and consisting of a collection of LAN and WAN components is called an Intranet. Accordingly, communications entities that are part of an intranet can use a Web browser to access Web servers that are within the intranet or on the Internet.

[0006] Today, most of the communication links between the various communications entities in a networks are wire-line; that is, client systems are typically connected to a server and to other client systems by wires, such as twisted-pair wires, coaxial cables, fiber optic cables, and the like. Wireless communication links, such as microwave links, radio frequency (RF) links, infrared (IR) links, and satellite links, are becoming more prevalent in networks.

[0007] A characteristic of wireless networks is that the communication entities in the network are mobile. Such mobility creates frequent, dynamic changes to the network topology and state of the communication links between the communication entities. Mobility is less of a concern for those communication entities connected to the Internet by wire-line, however, the topology of the Internet is perpetually changing, with communication entities joining and leaving the Internet often. Also, the state of communication links between communication entities on the Internet may change for various reasons, such as increased packet traffic.

[0008] To effectively route messages through such dynamically changing networks, routers need to remain informed of topology and link-state changes. Existing methods based on flooding are inefficient and consume too much network bandwidth. The inefficiency of flooding is the result, in part, of the following redundancies: (1) link-state and topology updates are sent over multiple paths to each router; and (2) every router forwards every update to all neighboring routers, even if only a small subset of the neighboring routers need to receive it.

[0009] Thus, there remains a need for a mobile wireless network that can perform reliably and efficiently despite the aforementioned difficulties associated with the mobility of the communication entities in the network.

## SUMMARY OF THE INVENTION

[0010] In one aspect, the invention features a method for disseminating topology and link-state information over a multi-hop network comprised of nodes. A path tree is maintained for each source node in the network that can produce an update message. The path tree associated with each node can be a minimum-hop-path tree. Link-state information obtained from one or more nodes in the path tree maintained for a given source node is used in developing the path tree to that source node. Each path tree has that source node as a root node, a parent node, and zero or more children nodes. An update message is received from the parent node in the path tree maintained for the source node that originated the update message. The update message includes information related to a link in the network. A determination is made whether to forward the update message to children nodes, if any, in the path tree maintained for the source node originating the update message in response to information in the received update message. In one embodiment, the information related to the link indicates whether the update message is to be forwarded to other nodes. The link can be a wireless communication link.

[0011] In one embodiment, a new parent message is sent to a node, which selects that node as a new parent node for the source node originating the update message. In response

to the new parent message link-state information associated with the source node that originated the update message is received from the new parent node. The new parent message can include a serial number. The link-state information then received in response to the new parent message is associated with update messages having serial numbers that are greater than the serial number included in the new parent message.

[0012] In another embodiment, a path through a new parent node for the source node originating the update message may have the same number of node hops as the path through the current parent node. In this case the current parent node is maintained as the parent node for the given source node. In still another embodiment, a path to the source node originating the update message may cease to exist. In this case, the current parent node is maintained as the parent node for the source node in the event that the path to the source node is recovered.

[0013] The update message may be broadcast to the children nodes if the number of children nodes exceeds a predefined threshold when forwarding the update message to children nodes. Alternatively, the update message may be transmitted to each child node using a unicast mode if the number of children nodes is less than a predefined threshold when forwarding the update message to children nodes.

[0014] In another aspect, the invention features a network comprising a plurality of nodes in communication with each other over communication links. Each node maintains a path tree for each source node in the network that can produce an update message. Each path tree has that source node as a root node, a parent node, and zero or more children nodes. One of the nodes (i) receives an update message from the parent node in the path tree maintained for the source node that originated the received update message, the update message including information related to a link in the network, (ii) and determines whether to forward the update message to children nodes, if any, in the path tree maintained for the source node that originated the update message in response to the information in the received update message.

## BRIEF DESCRIPTION OF THE DRAWINGS

[0015] The invention is pointed out with particularity in the appended claims. The objectives advantages of the invention described above, as well as further objectives and advantages of the invention, may be better understood by reference to the following description taken in conjunction with the accompanying drawings, in which:

[0016] **FIG. 1** is a block diagram of an embodiment of a mobile internetworking system including a plurality of subnets in communication with the Internet;

[0017] **FIG. 2** is a block diagram of a portion of an embodiment of protocol stack that can be implemented by each of the routing nodes in each subnet to communicate in accordance with the principles of the invention;

[0018] **FIG. 3** is a flow diagram illustrating an embodiment of a process by which each routing node selects a parent neighbor node and children neighbor node(s) for each potential source node in the subnet to define a minimum-hop-path tree for each potential source node along which routing nodes receive and forward link-state updates originating from that source node;

[0019] **FIG. 4** is a diagram illustrating an embodiment of an exemplary minimum-hop-path tree for the nodes in the subnet of **FIG. 1**;

[0020] **FIG. 5** is a block diagram illustrating the operation of a partial topology version of the TBRPF protocol;

[0021] **FIG. 6** is a diagram of an embodiment of a format of a message header for an atomic TBRPF protocol message;

[0022] **FIG. 7** is a diagram of an embodiment of a format of a compound TBRPF message;

[0023] **FIGS. 8A and 8B** are diagrams of embodiments of a format of a NEW PARENT protocol message;

[0024] **FIG. 9** is a diagram of an embodiment of a format for a CANCEL PARENT message;

[0025] **FIGS. 10A and 10B** are diagrams of embodiments of exemplary formats for link-state update messages;

[0026] **FIG. 11** is a diagram of an embodiment of an exemplary format of a RETRANSMISSION_OF_BROADCAST message;

[0027] **FIG. 12** is a flow diagram of an embodiment of a process performed by the nodes of the subnet to achieve neighbor discovery;

[0028] **FIG. 13** is a diagram of a packet format for the protocol messages used for neighbor discovery;

[0029] FIGS. **14** are a flow diagram of another embodiment of a process for performing neighbor discovery;

[0030] **FIG. 15A** is a diagram of a format for an IPv6 address including a prefix and an interface identifier;

[0031] **FIG. 15B** is a diagram of an embodiment of the interface identifier including a 24-bit company identifier concatenated with a 40-bit extension identifier;

[0032] **FIG. 15C** is a diagram of an embodiment of the interface identifier including a 24-bit company identifier concatenated with the 40-bit extension identifier;

[0033] **FIG. 15D** is a diagram of an IP v6-IPv4 compatibility address;

[0034] **FIG. 15E** is a diagram of an embodiment of a message format for tunneling an IPv6-IPv4 compatibility address through IPv4 infrastructure;

[0035] **FIG. 16** is a flow diagram of an embodiment of a process by which a router tests an IPv6-IPv4 compatibility address;

[0036] **FIG. 17** is a flow diagram of an embodiment of a process by which a mobile node and a server exchange messages;

[0037] **FIGS. 18A and 18B** are diagrams illustrating an example of the operation of a message queue.

## DESCRIPTION OF THE INVENTION

[0038] **FIG. 1** shows an embodiment of an internetworking system **2** including communication sub-networks ("subnets") **10**, **20** that are components of a worldwide network of networks **30** (i.e., the "Internet"). The Internet **30** includes communications entities, (e.g., hosts and routers), that exchange messages according to an Internet Protocol (IP) such as IPv4 (version 4) and IPv6 (version 6). On the

Internet **30**, entities implementing IPv6 may coexist with IPv4 entities. In general, the IPv4 and IPv6 versions are incompatible. The incompatibility is due, in part, to the difference in addressing format: the IPv4 specifies a 32-bit address format, whereas the IPv6 specifies a 128-bit address format.

[**0039**] A server **40** is connected to the Internet **30** by a wire-line or wireless connection. The server **40** can be internal or external to the subnet **10**. For purposes such as hosting application programs, delivering information or Web pages, hosting databases, handling electronic mail ("e-mail"), or controlling access to other portions of the Internet **30**, the server **40** is a computer system that typically handles multiple connections to other entities (e.g., client systems) simultaneously. Although represented as a single server **40**, other embodiments can have a group of interconnected servers. The data on the server **40** are replicated on one or more of these interconnected servers to provide redundancy in the event that a connection to the server **40** cannot be established.

[**0040**] Each subnet **10**, **20** includes one or more networks that can include both local area network (LAN) and wide area network (WAN) components. Each subnet **10**, **20** may be a freely accessible component of the public Internet **30**, or a private Intranet. The subnet **10** includes IP hosts **12**, routers **14**, and a gateway **16** (collectively referred to as nodes **18**). As used hereafter, a router **14** is any node **18** that forwards IP packets not explicitly addressed to itself, and an IP host **12** is any node **18** that is not a router **14**. Examples of devices that can participate as a node **18** in the subnet **10** include laptop computers, desktop computers, wireless telephones, and personal digital assistants (PDAs), network computers, television sets with a service such as Web TV, client computer systems, server computer systems. The gateway **16** is a particular type of routing node **14** that connects the subnet **10** to the Internet **30**. The subnet **20** is similarly configured with nodes **18**' (i.e., hosts **12**', routers **14**', and gateways **16**').

[**0041**] The subnet **10** can be associated with one organization or administrative domain, such as an Internet service provider (ISP), which associates each node **18** with an assigned IPv6 or IPv4 network address. Each IPv6 address is globally unique, whereas each IPv4 address is locally unique at least within the subnet **10**, and may be globally unique. Presumably, the assigned IP address has some topological relevance to the "home" subnet **10** of the node **18** so that the nodes **18** of the subnet **10** can be collectively identified by a common address prefix for routing purposes (called address aggregation). In one embodiment, the gateway **16** is a dual-stack node; that is, the gateway **16** has two IP addresses, an IPv6 address and an IPv4 address, and can route packets to IPv4 and IPv6 nodes.

[**0042**] Although it is conceivable that all nodes **18** in subnet **10** are initially assigned network addresses that follow a common address convention and have a common network prefix, dynamic topology changes may result in nodes **18** leaving their home subnet **10** to join a "foreign" subnet (e.g., subnet **20**) and new nodes joining the home subnet **10**. Because the nodes **18** maintain the same IP address irrespective of whether the node **18** changes its location within the subnet **10** or moves to the foreign subnet **20**, mobility may result in a heterogeneous conglomerate of IPv6 and IPv4 addresses, having various network prefixes, within the single subnet **10** unless some form of dynamic address assignment or other address-renumbering scheme is used. Further, the gradual transition from the use of IPv4 network addresses to IPv6 network addresses within the subnet **10** increases the likelihood of such a heterogeneous conglomeration. Thus, like the Internet **30**, the infrastructure of the subnet **10** can become heterogeneous; some nodes **18** can be IPv4 nodes, while others are IPv6 nodes.

[**0043**] In the subnet **10**, each node **18** can establish connectivity with one or more other nodes **18** through broadcast or point-to-point links. In general, each link is a communication facility or medium over which nodes **18** can communicate at the link layer (i.e., the protocol layer immediately below the Internet Protocol layer.) Such communication links can be wire-line (e.g., telephone lines) or wireless; thus, nodes **18** are referred to as wireless or wire-line depending upon the type of communication link that the node **18** has to the subnet **10**. Examples of wireless communication links are microwave links, radio frequency (RF) links, infrared (IR) links, and satellite links. Protocols for establishing link layer links include Ethernet, PPP (Point-to-Point Protocol) links, X.25, Frame Relay, or ATM (asynchronous transfer mode). Each wireless node **18**, e.g., IP host A **12**, has a range **22** of communication within which that node **18** can establish a connection to the subnet **10**. When beyond the range **22** of communication, the IP host A **12** cannot communicate with the server **40** on the Internet **30** or with other nodes **18** in the subnet **10**.

[**0044**] Each broadcast link connecting multiple nodes **18** is mapped into multiple point-to-point bi-directional links. For example, a pair of nodes **18** is considered to have established a bi-directional link **18**, if each node **18** can reliably receive messages from the other. For example, IP host A **12** and node B **14** have established a bi-directional link **24** if and only if IP host A **12** can receive messages sent from node B **14** and node B **14** can receive messages sent from IP host A **12** at a given instant in time. Nodes **18** that have established a bi-directional link are considered to be adjacent (i.e., neighboring nodes). Such a bi-directional link **24** between the two nodes A and B is represented by a pair of unidirectional links (A, B) and (B, A). Each link has at least one positive cost (or metric) that can vary in time, and for any given cost, such cost for the link (A, B) may be different from that for the link (B, A). Any technique for assigning costs to links can be used to practice the invention. For example, the cost of a link can be one, for minimum-hop routing, or the link delay plus a constant bias.

[**0045**] In one embodiment, the subnet **10** is a mobile "ad hoc" network ("MANET") in that the topology of the subnet **10** and the state of the links (i.e., link state) between the nodes **18** in the subnet **10** can change frequently because several of the nodes **18** are mobile. That is, each mobile node **18** may move from one location to another location within the same subnet **10** or to another subnet **20**, dynamically breaking existing links and establishing new links with other nodes **18**, **18**' as a result. Such movement by one node **18** does not necessarily result in breaking a link, but may diminish the quality of the communications with another node **18** over that link. In this case, a cost of that link has increased. Movement that breaks a link may interrupt any on-going communications with other nodes **18** in the subnet **10** or in the foreign subnet **20**, or with servers (e.g., server

**40**) connected to the Internet **30**. In another embodiment, the position of every node **18** in the subnet **10** is fixed (i.e., a static network configuration in which no link state changes occur due to node mobility). As the principles of the invention apply to both static and dynamic network configurations, a reference to the subnet **10** contemplates both types of network environments.

[**0046**] The following example illustrates the operation of the subnet **10**. Consider, for example, that node A is communicating with the server **40** over a route through subnet **10** that includes the link (A, B) to node B **14**, when node A **12** moves from its present location. This movement breaks the communication link with node B **14** and, as a result, interrupts communications with the server **40**. The relocation of node A **12** may break a link with one or more other nodes **18** as well. As one example, the movement by node A **12** may temporarily take node A **12** out of communication range with node B **14**, and upon returning within range, node A **12** can reestablish the broken link **24** with node B **14**. In this example, the link **24** is intermittent. As another example, node A **12** may move to a different location within the subnet **10** altogether and reestablish a bi-directional link **26** with a different node, (e.g., here node H). In yet another example, node A **12** may move to the foreign subnet **20** and establish a bi-directional link **28** with a node **14**' in the subnet **20** (e.g., node M **14**').

[**0047**] Each router **14** in the subnet **10** is responsible for detecting, updating, and reporting changes in cost and up-or-down status of each outgoing communication link to neighbor nodes. Thus, each router **14** in the subnet **10** runs a link-state-routing protocol for disseminating subnet topology and link-state information to the other routers **14** in the subnet **10**. Each router **14** also executes a neighbor discovery protocol for detecting the arrival of new neighbor nodes and the loss of existing neighbor nodes. To achieve discovery, IP hosts **12** connected to the subnet **10** also run the neighbor discovery protocol. IP hosts **12** can also operate as routers by running the link-state-routing protocol (in the description, such routing IP hosts are categorically referred to as routers **14**). The link-state-routing protocol, referred to as a topology broadcast based on reverse-path forwarding (TBRPF) protocol, seeks to substantially minimize the amount of update and control traffic required to maintain shortest (or nearly shortest) paths to all destinations in the subnet **10**.

[**0048**] In brief, the TBRPF protocol performed by each of the routers **14** in the subnet **10** operates to inform a subset of the neighboring routers **14** in the subnet **10** of the current network topology and corresponding link-state information. Thus, for the examples above, each router **14** in the subnet **10** that detects a change in a link to node A **12**, (e.g., node B **14** in the cost of the link (B, A)), operates as the source (i.e., source node) of an update. Each source node sends a message to a neighbor of that source node, informing the neighbor of the update to that link. Each router **14** receiving the update may subsequently forward the update to zero or more neighbor nodes, until the change in the topology of the subnet **10** disseminates to the appropriate routers **14** in the subnet **10**.

[**0049**] To transmit update messages, the TBRPF protocol supports unicast transmissions (e.g., point-to-point or receiver directed), in which a packet reaches only a single neighbor, and broadcast transmissions, in which a single packet is transmitted simultaneously to all neighbor nodes. In particular, the TBRPF protocol allows an update to be sent either on a common broadcast channel or on one or more unicast channels, depending on the number of neighbors that need to receive the update.

[**0050**] Upon recovering the same link to node B **14**, or upon reestablishing a new link to another node **18** in the same subnet **10** or in the foreign subnet **20**, the node A **12** can resume the interrupted communications with server **40**. In effect, one of the nodes **18**, **18**' in the subnet **10**, **20**, respectively, using the neighbor discovery protocol, discovers node A **12** and, using the TBRPF protocol, initiates dissemination of topology and link-state information associated with the link to node A **12**. The routers **14** also use the TBRPF protocol to disseminate this information to the other routers in the respective subnet **10** so that one or more routes to the node A **12** become available for communication with the server **40**.

[**0051**] In one embodiment, such communications resume at their point of interruption. In brief, node A **12** maintains, in local cache, copies of objects that are located on the server **40**. When node A **12** and the server **40** are in communication, node A **12** updates the objects as necessary, thereby maintaining substantially up-to-date copies of the objects. Thus, when node A **12** moves out of the communication range **22** with the subnet **10**, the node A **12** initially has up-to-date information. Then when node A **12** reconnects to the subnet **10**, the server **40** forwards previously undelivered updates to the objects locally stored at node A **12**, along a route determined by information stored at each of the routing nodes **14**. In the event node A **12** reconnects to the foreign subnet **20**, a hand-off protocol, such as MobileIP, is used to achieve the redirection of the messages between the server **40** and the node A **12**.

[**0052**] The route taken by the object updates may traverse a heterogeneous IPv6/IPv4 infrastructure. Normally, IPv6 nodes are unable to route packets to other IPv6 nodes **18** over routes that pass through IPv4 infrastructure. In one embodiment, described in more detail below, the nodes **18** use an IPv6-IPv4 compatible aggregatable global unicast address format to achieve such routing. This IPv6-IPv4 compatibility address format also enables incremental deployment of IPv6 nodes **18** that do not share a common multiple access data-link with another IPv6 node **18**.

[**0053**] Accordingly, the internetworking system **2** provides various mobile ad hoc extensions to the Internet **30** that are particularly suited to the dynamic environment of mobile ad hoc networks. Such extensions, which are described further below, include techniques for (1) disseminating update information to nodes **18** in the subnet **10** using the TBRPF protocol; (2) detecting the appearance and disappearance of new neighbor nodes using a neighbor discovery protocol; (3) establishing an address format that facilitates deployment of IPv6 nodes in a predominantly IPv4 network infrastructure; (4) updating information upon resuming communications between nodes; and (5) adaptively using network bandwidth to establish and maintain connections between nodes **18** and the server **40**.

[**0054**] **FIG. 2** shows a portion of an embodiment of protocol stack **50** that can be used by each of the routing nodes **14**, **14**' to communicate with other routing nodes **14** in the subnet **10**, **20** and on the Internet **30**, and thereby

implement the various extensions to the Internet **30** described herein. The protocol stack **50** includes a data-link layer **54**, a network layer **62**, and a transport layer **70**.

[**0055**] The data link layer **54** can implemented by any conventional data link protocol (e.g., IEEE 802.11) with an addressing scheme that supports broadcast, multicast and unicast addressing with best-effort (not guaranteed) message delivery services between nodes **18** having instantaneous bi-directional links. For such implementations, each node **18** in the subnet **10** has a unique data link layer unicast address assignment.

[**0056**] The network layer **62** is the protocol layer responsible for assuring that packets arrive at their proper destination. Some of the mobile ad hoc extensions for the Internet **30** described herein operate at the network layer **62**, such as the TBRPF protocol **58** and the IPv6-IPv4 compatibility address format, described in more detail below. Embodiments that redirect communications from foreign subnets to home subnets also use hand-off mechanisms such as Mobile IP, which operate at the network layer **62**. At the transport layer **70**, other mobile ad hoc extensions to the Internet **30** are implemented, such as techniques for updating communications upon restoring connections between nodes and for adaptively using the network bandwidth.

[**0057**] 1. Topology Broadcast Based on Reverse-Path Forwarding (TBRPF) Protocol

[**0058**] In brief, the TBRPF protocol uses the concept of reverse-path forwarding to broadcast each link-state update in the reverse direction along a tree formed by the minimum-hop paths from all routing nodes **14** to the source of the update. That is, each link-state update is broadcast along the minimum-hop-path tree rooted at the source (i.e., source node "src") of the update. The minimum-hop-path trees (one tree per source) are updated dynamically using the topology and link-state information that are received along the minimum-hop-path trees themselves. In one embodiment, minimum-hop-path trees are used because they change less frequently than shortest-path trees that are determined based on a metric, such as delay. Other embodiments of the TBRPF protocol can use other types of trees, such as shortest path trees, to practice the principles of the invention.

[**0059**] Based on the information received along the minimum-hop-path trees, each node **18** in the subnet **10** computes a parent node and children nodes, if any, for the minimum-hop-path tree rooted at each source node src. Each routing node **14** may receive and forward updates originating from a source node src along the minimum-hop-path tree rooted at that source node src. Each routing node **14** in the subnet **10** also engages in neighbor discovery to detect new neighbor nodes and the loss of existing neighbor nodes. Consequently, the routing node **14** may become the source of an update and thus may generate an update message. When forwarding data packets to a destination node, each routing node **14** selects the next node on a route to the destination.

[**0060**] To communicate according to the TBRPF protocol, each routing node **14** (or node i, when referred to generally) in the subnet **10** stores the following information:

[**0061**] 1. A topology table, denoted TT_i, consisting of all link-states stored at node i. The entry for link (u, v) in this table is denoted TT_i(u, v) and includes the most recent update (u, v, c, sn) received for link (u, v). The component c represents the cost associated with the link, and the component sn is a serial number for identifying the most recent update affecting link (u, v) received by the node i. The components c and sn of the entry for the link (u, v) is denoted TT_i(u, v).c and TT_i(u, v).sn. Optionally, the dissemination of multiple link metrics is attainable by replacing the single cost c with a vector of multiple metrics.

[**0062**] 2. A list of neighbor nodes, denoted N_i.

[**0063**] 3. For each node u other than node i, the following is maintained:

[**0064**] a. The parent, denoted p_i(u), which is the neighbor node ("nbr") of node i that is the next node on a minimum-hop path from node i to node u, as obtained from the topology table TT_i.

[**0065**] b. A list of children nodes of node i, denoted children_i(u).

[**0066**] c. The sequence number of the most recent link-state update originating from node u received by node i, denoted sn_i(u). The sequence number is included in the link-state update message. The use of sequence numbers helps achieve reliability despite topology changes, because node i avoids sending another node information that the other node already has. Each node i maintains a counter (i.e., the sequence number) for each link that the node i monitors. That counter is incremented each time the status of the link changes.

[**0067**] d. The routing table entry for node u, consisting of the next node on a preferred path to node u. The routing table entry for node u can be equal to the parent p_i(u) if minimum-hop routing is used for data packets. However, in general, the routing table entry for node u is not p_i(u), because the selection of routes for data traffic can be based on any objective.

[**0068**] One embodiment of the TBRPF protocol uses the following message types:

[**0069**] LINK-STATE UPDATE: A message containing one or more link-state updates (u, v, c, sn).

[**0070**] NEW PARENT: A message informing a neighbor node that the node has selected that neighbor node to be a parent with respect to one or more sources of updates.

[**0071**] CANCEL PARENT: A message informing a neighbor that it is no longer a parent with respect to one or more sources of updates.

[**0072**] HELLO: A message sent periodically by each node i for neighbor discovery.

[**0073**] NEIGHBOR: A message sent in response to a HELLO message.

[**0074**] NEIGHBOR ACK: A message sent in response to a NEIGHBOR message.

[**0075**] ACK: A link-level acknowledgment to a unicast transmission.

[0076] NACK: A link-level negative acknowledgment reporting that one or more update messages sent on the broadcast channel were not received.

[0077] RETRANSMISSION OF BROADCAST: A retransmission, on a unicast channel, of link-state updates belonging to an update message for which a NACK message was received.

[0078] HEARTBEAT: A message sent periodically on the broadcast channel when there are no updates to be sent on this channel, used to achieve reliable link-level broadcast of update messages based on NACKs.

[0079] END OF BROADCAST: A message sent to a neighbor over a unicast channel, to report that updates originating from one or more sources are now being sent on the unicast channel instead of the broadcast channel.

[0080] The formats for the various types of TBRPF protocol messages are described below.

[0081] Building the Minimum-Hop-Path Tree for a Source

[0082] **FIG. 3** shows an embodiment of a process by which each routing node **14** selects a parent neighbor node and children neighbor node(s) for each potential source node src in the subnet **10**. The selection of the parent and children neighbor nodes for each potential source node src define a minimum-hop-path tree for that potential source node along which the routing nodes **14** receive and forward link-state updates originating from that source node src. Pseudo-code describing the network-level procedures performed by each routing node **14** is in Appendix A.

[0083] Node i receives (step **90**) a message over a communication link. The received message can represent a link-state update, the discovery of a new neighbor node, the loss of a neighbor node, a change in the cost of a link to a neighbor node, a selection of a new parent neighbor node, or a cancellation of a parent neighbor node. Pseudo-code for processing these types of received messages is provided in Appendix A; the corresponding procedures are called Process_Update, Link_Up, Link_Down, Link_Change, Process_New_Parent, and Process_Cancel_Parent, respectively. The general procedure followed in response to all of these events, and the specific procedure followed by a node that has just started and has no topology information are described below.

[0084] If node i receives a message representing a link-state update, the discovery of a new neighbor node, the loss of a neighbor node, or a change in the cost of a link to a neighbor node, node i enters (step **100**) the new link-state information, if any into the topology table, TT_i, and forwards (step **102**) the link-state information in a link-state update to the neighbor nodes in children_i(src), where src is the source node at which the update originated. Node i then computes (step **104**) the parent nodes p_i(u) for all potential source nodes src by running a shortest-path algorithm such as Dijkstra's algorithm. If this computation results in a change to the parent node p_i(u) for any source u, node i then sends a NEW PARENT(u, sn) message, where sn=sn_i(u), to the new parent node p_i(u) and a CANCEL PARENT message to the old parent node (step **106**).

[0085] If node i receives (step **90**) a NEW PARENT(u, sn) message from a sending node with source u and sequence number sn, node i adds(step **108**) the sending node to node i's list of children nodes children_i(u) for that source u, and then sends (step **110**) the sending node a LINK-STATE UPDATE message containing all updates in node i's topology table, TT_i, originating from source u and having a sequence number greater than sn. If node i receives (step **90**) a CANCEL PARENT(u) message from a sending node with source u, node i removes (step **112**) the sending node from node i's list of children nodes children_i(u) for that source u.

[0086] Consider, for example, the case in which node i initially has no topology information. Accordingly, node i has no links to neighbor nodes, and its topology table TT_i is empty. Also the parent node is p_i(src)=NULL (i.e., not defined), the children_i(src) is the empty set, and sn_i(src)=0 for each source node src. Upon receiving (step 90) messages representing the discovery of neighbor nodes, node i executes the Link_Up procedure to process each link established with each neighbor node nbr. Because each neighbor node nbr of node i is (trivially) the next node on the minimum-hop path from node i to neighbor node nbr, node i selects (step **104**) each of its neighbor nodes nbr as the new parent node p_i(nbr) for source node nbr. Execution of the Link-Up procedure results in node i sending (step **106**) a NEW PARENT message to each neighbor node nbr. Therefore, the NEW PARENT message sent to a new neighbor node nbr contains the neighbor node nbr (and possibly other sources) in its source list.

[0087] In response to the NEW PARENT message, then each neighbor node nbr informs (step **110**) node i of the outgoing links of neighbor node nbr. Information about the outgoing links of neighbor node nbr allows node i to compute minimum-hop paths to the nodes at the other end of the outgoing links, and thus to compute (step **104**) new parents p_i(src), for all source nodes src that are two hops away. Node i sends (step **106**) a NEW PARENT message to each of these computed new parents. Then each parent p_i(src) for each such source node src informs (step **110**) node i of the outgoing links for source node src, which allows node i to compute (step **104**) new parents for all source nodes that are three hops away. This process continues until node i has computed parent nodes for all sources nodes src in the subnet **10**. As a result, for a given source node src, the parents p_i(src) for all nodes i other than source node src define a minimum hop-path tree rooted at source node src (after the protocol has converged).

[0088] Node i cancels an existing parent p_i(src) by sending a CANCEL PARENT(src) message containing the identity of the source node src. Consequently, the set of children, children_i(src), at node i with respect to source node src is the set of neighbor nodes from which node i has received a NEW PARENT message containing the identity of source node src without receiving a subsequent CANCEL PARENT message for that source node src. Node i can also simultaneously select a neighbor node as the parent for multiple sources, so that the node i sends a NEW PARENT(src_list, sn_list) message to the new parent, where src_list is the list of source nodes and sn_list is the corresponding list of sequence numbers. Similarly, a CANCEL PARENT message can contain a list of sources.

[0089] In one embodiment, the TBRPF does not use NEW PARENT and CANCEL PARENT messages in the generation the minimum-hop-path tree. Instead, each node i computes the minimum-hop paths from each neighbor node nbr to all destinations (e.g., by using breadth-first search or Dijkstra's shortest-path algorithm). Consequently, each node i computes the parents p_nbr(src) for each neighbor node nbr and source node src, from which node i determines which neighbor nodes nbr are its children for the given source node src. Although this process eliminates NEW PARENT and CANCEL PARENT messages, the process also requires that each node i (1) sends all updates originating from the source node src to any child node in children_i(src), or (2) periodically sends updates along the minimum-hop-path tree, because node i does not know the sequence number sn_nbr(src) from the neighbor node nbr and thus does not know what updates the neighbor node nbr already has. Either of these actions ensures that each neighbor node nbr receives the most recent information for each link.

[0090] **FIG. 4** shows an embodiment of an exemplary minimum-hop-path tree **120** for the nodes **18** in the subnet **10** of **FIG. 1**. For the sake of illustration, assume that node D is the source of an update. The parent **122** for nodes C, G, and H with respect to the source node D is node D; the parent **124** for node F with respect to source node D is node H; the parent **126** for nodes A and B with respect to source node D is node F; the parent **128** for node E is node B; and the parent **130** for node L is node A. (In this example, node A is a routing node **14**, and thus runs the TBRPF protocol.)

[0091] Conversely, the children **132** of node D are nodes C, G, and H; the child **134** of node H is node F; the children **136** of node F are nodes A and B; the child **138** of node B is node E, and the child **140** of node A is node L. As shown, nodes C, E, G, and L are leaf nodes, which, in accordance with the TBRPF protocol, do not have to forward updates originating from the source node D.

[0092] Updating the Minimum-Hop-Path Tree

[0093] In brief, the TBRPF protocol disseminates link-state updates generated by a source node src along the minimum-hop-path tree rooted at node src and dynamically updates the minimum-hop-path tree based on the topology and link-state information received along the minimum-hop-path tree. More specifically, whenever the topology table TT_i of node i changes, the node i computes its parent p_i(src) with respect to every source node src (see the procedure Update_Parents in Appendix A). The node i computes parents by (1) computing minimum-hop paths to all other nodes using, for example, Dijkstra's algorithm, and (2) selecting the next node on the minimum-hop path to each source node src to be the parent for that source node src (see the procedure Compute_New_Parents in Appendix A). The computation of parents occurs when the node i receives a topology update, establishes a link to a new neighbor node, or detects a failure or change in cost of a link to an existing neighbor node.

[0094] In one embodiment, node i computes a new parent p_i(src) for a given source node src even though the path to the source node src through the new parent has the same number of hops as the path to the source node src through the old parent. In another embodiment, the node keeps the old parent node in this event, thus reducing the overhead of the TBRPF protocol. This embodiment can be implemented, for example, by using the procedure Compute_New_Parents**2** (given in Appendix A) instead of the procedure Compute_New_Parents.

[0095] If the parent p_i(src) changes, node i sends the message CANCEL PARENT(src) to the current (i.e., old) parent, if the old parent exists. Upon receiving the CANCEL PARENT(src) message, the old parent ("k") removes node i from the list children_k(src).

[0096] Node i also sends the message NEW PARENT(src, sn) to the newly computed parent if the new parent exists, where sn=sn_i(src) is the sequence number of the most recent link-state update originating from source node src received by node i. This sequence number indicates the "position" up to which node i has received updates from the old parent, and indicates to the new parent that it should send only those updates that occurred subsequently (i.e., after that sequence number).

[0097] Upon receiving the NEW PARENT(src, sn) message, the new parent "j" for p_i(src) adds node i to the list children_j(src) and sends to node i a link-state update message consisting of all the link states originating from source node src in its topology table that have a sequence number greater than sn (see the procedure Process_New_Parent in Appendix A). Thus, only updates not yet known to node i are sent to node i.

[0098] Generally, the range of sequence numbers is large enough so that wraparound does not occur. However, if a small sequence number range is used, wraparound can be handled by employing infrequent periodic updates with a period that is less than half the minimum wraparound period, and by using a cyclic comparison of sequence numbers. That is, sn is considered less than sn' if either sn is less than sn' and the difference between sn and sn' (sn'−sn') is less than half the largest possible sequence number, or sn' is less than sn and the difference, sn−sn', is greater than half the largest possible sequence number.

[0099] When a node i detects the existence of a new neighbor nbr, it executes Link_Up(i, nbr) to process this newly established link. The link cost and sequence number fields for this link in the topology table at node i are updated. Then, the corresponding link-state message is sent to all neighbors in children_i(i). As noted above, node i also recomputes its parent node p_i(src) for every node src, in response to this topological change. In a similar manner, when node i detects the loss of connectivity to an existing neighbor node nbr, node i executes Link_Down(i, nbr). Link_Change(i, nbr) is likewise executed at node i in response to a change in the cost to an existing neighbor node nbr. However, this procedure does not recompute parents.

[0100] In one embodiment, if a path between the node i and a given source node src ceases to exist, the node i computes a new parent p_i(src) that is set to NULL (i.e., parent does not exist). In another embodiment, although the path between the node i and the given source node src ceases to exist, the node i keeps the current parent, if the current parent is still a neighbor node of the node i. Thus, the overhead of the TBRPF protocol is reduced because it is unnecessary to send a CANCEL PARENT and a subsequent NEW PARENT messages if the old path to the source becomes operational later because of a link recovery. This

embodiment can be implemented by replacing the fifth line of the procedure Update_Parents in Appendix A, "If (new_p_i(src)!=p_i(src)){", with the line "If (new_p_i(src)!=p_i(src) and new_p_i(src)!=NULL){".

[0101] The TBRPF protocol does not use an age field in link-state update messages. However, failed links (represented by an infinite cost) and links that are unreachable (i.e., links (u, v) such that p_i(u)=NULL) are deleted from the topology table TT_i after MAX_AGE seconds (e.g., 1 hour) in order to conserve memory. Failed links (u, v) are maintained for some time in the topology table TT_i, rather than deleted immediately, to ensure that the node i that changes its parent p_i(u) near the time of failure (or had no parent p_i(u) during the failure) is informed of the failure by the new parent.

[0102] Unreachable links, (i.e., links (u, v) such that node i and node u are on different sides of a network partition), are maintained for a period of time to avoid having to rebroadcast the old link state for (u, v) throughout node i's side of the partition, if the network partition soon recovers, which can often happen if the network partition is caused by a marginal link that oscillates between the up and down states. If a link recovers resulting in the reconnection of two network components that were disconnected (i.e., partitioned) prior to the link recovery, the routing nodes **14** in one partition my temporarily have invalid routes to nodes **18** in the other partition. This occurs because the routing nodes **14** may receive an update message for the link recovery before receiving update messages for links in the other partition. Consequently, the link-information for those links in the other partition may be outdated temporarily.

[0103] To correct this situation, in one embodiment, a header field is added to each link-state update message, which indicates whether the update message is sent in response to a NEW PARENT message. The header field also identifies the corresponding NEW PARENT message using a sequence number. For example, if a given node i sends a NEW PARENT message (for multiple sources) to node j following the recovery of the link (i, j), the node i waits for a response from node j to the NEW PARENT message before sending to node i's neighbor nodes an update message corresponding to the link recovery. The response from node j includes the link-state information of the other nodes **18** in the previously disconnected partition. Then node i forwards this link-state information to node i's neighbor nodes. Consequently, the nodes **18** in the same partition as node i receives updates for the links in the other partition at the same time that the nodes **18** receive the update for the link recovery. Thus, the link-state information for those links in the other partition is not outdated temporarily.

[0104] A node i that is turned off (or goes to sleep) operates as if the links to all neighbors have gone down. Thus, the node i remembers the link-state information that it had when turned off. Since all such links are either down or unreachable, these link states are deleted from the topology table TT_i if the node i awakens after being in sleep mode for more than MAX_AGE seconds.

[0105] Infrequent periodic updates occur to correct errors that may appear in table entries or update messages. (See Send_Periodic_Updates in Appendix A.) As discussed above, periodic updates are also useful if the sequence number range is not large enough to avoid wraparound.

[0106] Initiating an Update Message

[0107] When a given routing node **14** detects a change in the state of a neighbor node, that routing node **14** becomes the source (i.e., source node src) of a link-state update message with respect to corresponding link to that neighbor node. As described above, the source node src then broadcasts each link-state update along the minimum-hop-path tree rooted at the source of the update.

[0108] A link-state update message reports the state of the link (src, nbr) as a tuple (src, nbr, c, sn), where c and sn are the cost and the sequence number associated with the update. A cost of infinity represents a failed link. The source node src is the head node of link (src, nbr), and is the only node that can report changes to parameters of link (src, nbr). Therefore, any node **18** receiving the link-state update (src, nbr, c, sn) can determine that the update originated from the source node src.

[0109] The source node src maintains a counter sn_src, which is incremented by at least one each time the cost of one or more outgoing links (src, nbr) changes value. For example, the counter sn_src can be a time stamp that represents the number of seconds (or other units of time) elapsed from some fixed time. When the source node src generates a link-state update (src, nbr, c, sn), the sequence number sn is set to the current value of sn_src.

[0110] Receiving an Update Message

[0111] In brief, each routing node **14** that receives a link-state update message receives that update message along a single path. That is, any link-state update originating from source node src is accepted by node i if (1) the link-state update is received from the parent node p_i(src), and (2) the link-state update has a larger sequence number than the corresponding link-state entry in the topology table TT_i at node i. If the link-state update is accepted, node i enters the link-state update into the topology table TT_i. Node i may then forward the link-state update to zero or more children nodes in children_i(src). In one embodiment, the link-state update passes to every child node in children_i(src). (See the procedures Update Topology Table and Process Update in the Appendix A.)

[0112] Forwarding Update Messages

[0113] In most link-state routing protocols, e.g., OSPF (Opens Shortest Path First), each routing node **18** forwards the same link-state information to all neighbor nodes. In contrast, in one embodiment of the TBRPF protocol, each routing node **14** sends each link-state update only to neighbor nodes that are children on the minimum-hop-path tree rooted at the source of the update. Each routing node **14** having no children for the source node src of the link-state update is a leaf in the minimum-hop-path tree and therefore does not forward updates originating from the source node src. In typical networks, most nodes **18** are leaves, thus the TBRPF protocol makes efficient use of the bandwidth of the subnet **10**. In addition, those nodes having only one child node for the source node src can send updates generated by the source node src to that child node only, instead of broadcasting the updates to all neighbor nodes.

[0114] The TBRPF protocol may utilize bandwidth more efficiently by using unicast transmissions if those routing nodes **14** have only one child, or a few children, for the

source of the update, and broadcast transmissions when several children exist for the update. Therefore, in one embodiment, the TBRPF protocol determines whether to use unicast or broadcast transmissions, depending on the number of children nodes and the total number of neighbor nodes.

[0115] In general, each routing node 14 uses unicast transmissions for updates with only one intended receiver (e.g., only one child), and broadcast transmissions for updates with several intended receivers, to avoid transmitting the update message several times. Therefore, each routing node 14 uses unicast transmission if k−1 and use broadcast if k>1, where k is the number of intended receivers. A possible drawback can occur if the number of children nodes exceeds one and there are a many more neighbors. For example, if there are two children nodes and twenty neighbor nodes, (i.e., k=2 and n=20, where k is the number of children nodes and n is the number of neighbors), then 18 neighbor nodes are listening to a message not intended for them. Such neighbor nodes could instead be sending or receiving other messages.

[0116] To avoid this possible drawback, one option is to use broadcast transmission if $k>(n+1)/2$ and unicast transmission in all other cases. In general, a rule of the form $k>g(n)$ can be used. For update messages, the number of children k may be different for different update sources. Therefore, it is possible to use unicast transmissions for some sources and broadcast transmissions for other sources, and the transmission mode for a given source u, denoted mode_i(u), can change dynamically between unicast and broadcast as the number of children changes.

[0117] While LINK-STATE-UPDATE messages can be transmitted in either unicast or broadcast mode, HELLO messages and HEARTBEAT messages (discussed below) are always transmitted on the broadcast channel, and the following messages are always transmitted on the unicast channel (to a single neighbor): NEIGHBOR, NEIGHBOR ACK, ACK, NACK, NEW PARENT, CANCEL PARENT, RETRANSMISSION OF BROADCAST, END OF BROADCAST, and LINK-STATE-UPDATE messages sent in response to a NEW PARENT message.

[0118] Exemplary pseudo-code for a procedure for sending a LINK-STATE UPDATE message (that is not a response to a NEW PARENT message) on the broadcast or unicast channel is as follows:

```
If (mode_i(src) = = BROADCAST)
    Append the message update_msg to the message queue
    associated with the broadcast channel.
If (mode_i(src) = = UNICAST)
    For (each node k in children_i(src))
        Append the message update_msg to the message queue
        associated with the unicast channel to node k.
```

[0119] Reliable unicast transmission of control packets can be achieved by a variety of reliable link-layer unicast transmission protocols that use sequence numbers and ACKs, and that retransmit a packet if an ACK is not received for that packet within a specified amount of time.

[0120] Reliable Transmission in Broadcast Mode

[0121] For reliable transmission of Link-State Update messages in broadcast mode, each broadcast update message includes one or more link-state updates, denoted lsu(src), originating from sources src for which the transmission mode is BROADCAST. Each broadcast control packet is identified by a sequence number that is incremented each time a new broadcast control packet is transmitted. Reliable transmission of broadcast control packets in TBRPF can be accomplished using either ACKs or NACKs. If ACKs are used, then the packet is retransmitted after a specified amount of time if an ACK has not been received from each neighbor node that must receive the message.

[0122] In one embodiment of TBRPF, NACKs are used instead of ACKs for reliable transmission of broadcast control packets, so that the amount of ACK/NACK traffic is minimized if most transmissions are successful. Suppose node i receives a NACK from a neighbor node nbr for a broadcast update message. In one embodiment, all updates lsu(src) in the original message, for each source node src such that neighbor node nbr belongs to children_i(src), are retransmitted (reliably) on the UNICAST channel to the neighbor node nbr, in a RETRANSMISSION OF BROADCAST message. This message includes the original broadcast sequence number to allow neighbor node nbr to process the updates in the correct order. In another embodiment, such update messages are retransmitted on the broadcast channel. This embodiment may improve the efficiency of the TBRPF protocol in subnets that do not support receiver-directed transmission, because in such subnets unicast transmission provides no efficiency advantage over broadcast transmissions.

[0123] The procedure for the reliable transmission of broadcast update packets uses the following message types (in addition to LINK-STATE UPDATE messages): HEARTBEAT(sn), NACK(sn, bit_map), and RETRANSMISSION OF BROADCAST(sn, update_msg). A NACK(sn, bit_map) message contains the sequence number (sn) of the last received broadcast control packet, and a 16-bit vector (bit_map) specifying which of the 16 broadcast control packets from sn−15 to sn have been successfully received.

[0124] A description of the procedure for the reliable transmission of broadcast update packets at node i uses the following exemplary notation:

[0125] Pkt(sn) represents a control packet with sequence number sn transmitted on the broadcast channel by node i.

[0126] MsgQ represents a message queue for new control messages to be sent on the broadcast channel from node i.

[0127] brdcst_sn_i represents the sequence number of the last packet transmitted on the broadcast channel by node i.

[0128] Heartbeat_Timer represents a timer used in the transmission of the HEARTBEAT message.

[0129] Following the transmission of the broadcast control packet Pkt(brdcst_sn_i) on the broadcast channel, node i increments brdcst_sn_i and reinitializes Heartbeat_Timer. When Heartbeat_Timer expires at node i, the node i appends the control message HEARTBEAT(brdcst_sn_i) to the mes-

sage queue associated with the broadcast channel, and reinitializes Heartbeat_Timer. When the node i receives NACK(sn, bit_map) from neighbor node nbr, node i performs the functions as illustrated by following exemplary pseudo-code:

```
    For each (sn' not received as indicated by bit__map){
        Let update__msg = {(src*, v*, sn*, c*) in Pkt(sn') such that the
            neighbor node nbr is in children__i(src*)}.
        Append the message RETRANSMISSION OF
            BROADCAST(sn', update__msg) to the message queue
            associated with the unicast channel to neighbor node nbr.
            (Message must be sent even if update__msg is empty.)}
```

[0130] Upon receipt at neighbor node nbr of control packet Pkt(sn) transmitted on the broadcast channel by node i, the neighbor node nbr performs the following operations as illustrated by the following pseudo-code:

```
If the control packet Pkt(sn) is received in error{
    Append the control message NACK(sn, bit__map) to the message
    queue
        associated with the unicast channel to node i.}
    If the control packet Pkt(sn) is received out of order (i.e., at least one
        previous sequence number is skipped){
            Withhold the processing of the control packet Pkt(sn).
            Append the control message NACK(sn, bit__map') to the
                message queue associated with the unicast channel
                to node i.}
    Else (control packet Pkt(sn) is received correctly and in order){
            For each Link-State Update message update__msg in
                Pkt(sn), call Process__Update(i, nbr, update__msg).}
```

[0131] When a communication link is established from node i to a new neighbor nbr, in one embodiment the node i obtains the current value of brdcst_sn_nbr from the NEIGHBOR message or NEIGHBOR ACK that was received from neighbor node nbr.

[0132] Each node i can dynamically select the transmission mode for link-state updates originating from each source node src. As described above, this decision uses a rule of the form k>g(n), where k is the number of children (for src) and n is the number of neighbors of node i. However, to ensure that updates are received in the correct order, or that the receiver has enough information to reorder the updates, node i sends an END OF BROADCAST(last_seq_no, src) message on the unicast channel to each child when the mode changes to UNICAST, and waits for all update packets sent on unicast channels to be ACKed on before changing to BROADCAST mode.

[0133] To facilitate this process, each node i maintains a binary variable unacked_i(nbr, src) for each neighbor node nbr and source node src, indicating whether there are any unACKed control packets sent to neighbor node nbr containing link-state updates originating at source node src. The following exemplary pseudo-code illustrates an embodiment of a procedure that is executed periodically at each node i.

```
For each (node src){
    If (mode__i(src) = BROADCAST and|children__i(src)| <= g(n)){
        For each (node nbr in children__i(src)){
            Append the message END OF BROADCAST(brdcst__sn__i,
            src) to
                the message queue associated with the unicast
                channel to node nbr.}
            Set mode__i(src) = UNICAST.}
    If (mode__i(src) = UNICAST and |children__i(src)| > g(n)){
        Set switch__flag = YES.
        For each (node nbr in children__i(src)){
            If (unacked__i(nbr, src) = YES) Set switch__flag = NO.}
        If (switch__flag = YES) Set mode__i(src) = BROADCAST.}}
```

[0134] Full and Partial Topology TBRPF

[0135] In one embodiment, a result of the running the TBRPF protocol is that each router **14** in the subnet **10** obtains the state of each link in the subnet **10** (or within a cluster if hierarchical routing is used). Accordingly, this embodiment of the TBRPF protocol is referred to as full-topology link-state protocol. In some embodiments, described below, the TBRPF protocol is a partial-topology link-state protocol in that each router **14** maintains a subset of the communication links in the subnet **10**. In the full-topology protocol embodiment, each routing node **14** is provided with the state of each link in the subnet **10** (or cluster, if hierarchical routing is used). In other embodiments, the TBRPF is a partial topology protocol in that each routing node **14** is provided with only a subset of the links in the subnet **10**.

[0136] For the full-topology link-state protocol embodiment (1) alternate paths and disjoint paths are immediately available, allowing faster recovery from failures and topology changes; and (2) paths can be computed subject to any combination of quality-of-service (QoS) constraints and objectives. Partial-topology link-state protocols provide each node **18** with sufficient topology information to compute at least one path to each destination. Whether implemented as a full-topology or as a partial-topology protocol, the TBRPF protocol is a proactive link-state protocol in that each node **18** dynamically reacts to link-state and topology changes and maintains a path to each possible destination in the subnet **10** at all times.

[0137] A Partial-Topology Embodiment

[0138] In one partial-topology embodiment, each routing node **14** decides which of its outgoing links (i, j), called "special links," should be disseminated to all nodes in the subnet **10**. This subset of links is maintained in a list L_i. All other outgoing links are sent only one hop (i.e., to all neighbor nodes of node i). Node i sends an update to its neighbor nodes if that update is the addition or removal of a link from the list L_i, or reflects a change in the state of a link in the list L_i.

[0139] Various rules can be used to define the set of special links in the list L_i. For example, one rule defines a link (i, j) to be in L_i only if node j is the parent of node i for some source node other than node j, or if node j belongs to the set children_i(src) for some source node src other than node i. This definition of special links includes enough links to provide minimum-hop paths between any pair of nodes. As a result, this partial-topology embodiment reduces the

amount of control traffic without reducing the quality of the routes. In this embodiment, an update (u, v, c, sn, sp) is augmented to include a, "sp" field (e.g., a single-bit field), which indicates whether the link (u, v) is a special link. Pseudo-code representing an exemplary implementation of the partial-topology embodiment appears in the Appendix A, after the "Partial-Topology 1" header. The procedure Mark_Special_Links(i) is called upon a change to the parent p_i(src) or to the set of children nodes children_i(src).

**[0140]** A Second Partial-Topology Embodiment

**[0141]** In another partial-topology embodiment, each routing node **14**, hereafter node i, maintains a topology table TT_i, a source tree Ti (i.e., computed paths to all destinations), a set of reported links Ri, and a set of neighbor nodes Ni. The entry of TT_i for a link (u, v) is denoted TT_i(u,v) and consists of the tuple (u, v, c, c'), where c is the cost associated with the link and c' is the last cost reported to neighbor nodes for the link. The component c of the entry for link (u, v) is denoted TT_i(u, v).c. In addition, a parent p_i(u) and set of children nodes children_i(u) are maintained for each node u≠node i. The parent p_i(u) is the next node on a shortest path to node u, based on the information in TT_i. The source tree Ti, computed by a lexicographic version of Dijkstra's algorithm, is the set of links that belong to at least one of the computed paths. The set of reported links Ri includes the source tree Ti and any link in TT_i for which an update has been sent but a delete update has not since been sent. In addition, a binary variable pending_i(u) is maintained for each node u≠node i, which indicates that the parent p_i(u) is pending, i.e., that a NEW PARENT(u) message has been sent to p_i(u) but no response has yet been received. In general, each node i reports to neighbor nodes the current states of only those links in its source tree Ti, but sends only part of its source tree Ti to each neighbor node such that no node receives the same information from more than one neighbor node. Pseudo-code representing an exemplary implementation of this partial-topology embodiment of the TBRPF protocol appears in the Appendix A, after the "Partial-Topology 2" header.

**[0142]** Upon receiving an update message, consisting of one or more updates (u, v, c), node i executes the procedure Update( ), which calls the procedure Update_Topology_Table( ), then executes the procedure Lex_Dijkstra( ) to compute the new source tree Ti and the procedure Generate_Updates( ) to generate updates and modify the set of reported links Ri based on changes in link costs and changes to the source tree Ti. Each generated update is then sent to the appropriate children, that is, updates for links with head u are sent to children_i(u). The procedure Update_Parents() is called, which determines any changes in the parent assignment and sends NEW PARENT and CANCEL PARENT messages.

**[0143]** The sending of updates can be accomplished in different ways, depending on whether the subnet **10** consists of point-to-point links, broadcast links, or a combination of both link types. In a network of point-to-point links, each neighbor node k would be sent a message that contains the updates for links (u, v) such that k belongs to children_i(u). If a broadcast capability also exists, links (u, v) for which children_i(u) has more than one member can be broadcast to all neighbor nodes.

**[0144]** The procedure Update_Topology_Table( ) does the following for each update (u, v, c) in the input message (in_message) such that the parent p_i(u) is the neighbor node who sent the message. (Updates received from a node other than the parent are ignored.) If either TT_i does not contain an entry for (u, v) or contains an entry with a different cost than c, then TT_i(u, v) is updated with the new value c and link (u, v) is marked as changed. If the input message is a PARENT RESPONSE, then in addition to updates, the message contains the same list of sources as the NEW PARENT message to which it is responding. For each such source node u such that pending_i(u)=1 and for each link (u, v) in TT_i that is outgoing from source node u but for which the input message does not contain an update, the cost of (u, v) is set to infinity, to indicate that the link should be deleted. In other words, any link that was reported by the old parent but is not reported by the new parent is deleted. Only information from the current parent is considered valid.

**[0145]** The procedure Lex_Dijkstra( ) (not included in Appendix A) is an implementation of Dijkstra's algorithm that computes the lexicographically smallest shortest path LSP(i, u) from node i to each node u, using as path name the sequence of nodes in the path in the reverse direction. For example, the next-to-last node of LSP(i, u) has the smallest node ID among all possible choices for the next-to-last node. Such paths are computed using a modification of Dijkstra's algorithm in which, if there are multiple choices for the next node to label, the one with the smallest ID is chosen.

**[0146]** The procedure Generate_Updates( ) decides what updates to include in the message to be sent to neighbor nodes. A non-delete update is included for any link (u, v) that is in the new source tree Ti and either is marked as changed or was not in the previous source tree (denoted old source tree Ti). In this case, Ti(u, v).c' is set to Ti(u, v).c, and (u, v) is added to the reported link set Ri if not already in the reported link set Ri. A delete update is included for any link (u, v) that is in the reported link set Ri but is not in the source tree Ti, such that TT_i(u, v).c>TT_i(u, v).c'. In this case, (u, v) is removed from the reported link set Ri. Any links with infinite cost are erased from the topology table TT_i.

**[0147]** The procedure Update_Parents( ) sets the new parent p_i(u) for each source node u to be the second node on the shortest path to node u. If there is no path to node u, p_i(u) is null. If the new parent is different from the old parent, then a NEW PARENT message is sent to the new parent (if it is not null) and a CANCEL PARENT message is sent to the old parent (if it is not null and the link to the old parent is still up). The NEW PARENT messages for all source nodes u having the same new parent are combined into a single message, and CANCEL PARENT messages are similarly combined.

**[0148]** The procedure Process_New_Parent( ) is executed when a NEW PARENT message is received from some neighbor node. For each source node u in the NEW PARENT message, the procedure adds the neighbor node to children_i(u) and includes in the PARENT RESPONSE message an update for each link (u, v) in the source tree Ti whose head is source node u, if such a link exists. (Such a link will not exist if node u is a leaf of source tree Ti.) As described above, the PARENT RESPONSE also includes the same list of sources as the NEW PARENT message to which it is responding. (This list is not necessary if the node

sending the NEW PARENT message remembers the list and can match the PARENT RESPONSE to the NEW PARENT message.)

**[0149]** When the cost of a link to a neighbor node j changes, node i sets TT_i(i, j).c to the new cost and calls the procedure Update( ) with k=i and an empty input message. A threshold rule can be used so that TT_i(i, j).c is updated only if the percent difference between the new cost and the old cost is at least some given threshold. If a link to a neighbor node j fails, the same procedure is followed (with the cost changing to infinity), and node j is removed from set of neighbor nodes Ni.

**[0150]** When a link to a neighbor node j comes up, either initially or upon recovering from a failure, node i executes the procedure Link_Up(I, j), which adds neighbor node j to the set of neighbor nodes Ni, sets TT_i(i, j).c to the link cost, and calls the procedure Update( ) with k=i and an empty input message. This may result in a NEW PARENT message being sent to neighbor node j.

**[0151]** To correct errors that may appear in TT_i due to noisy transmissions or memory errors, each node i can periodically generate updates for its outgoing links. Since a received update is ignored unless it has a cost that differs from the entry in the topology table TT_i, the cost of the periodic update should be chosen to be slightly different from the previous update. Alternatively, each update can contain an additional bit b, which toggles with each periodic update.

**[0152]** **FIG. 5** illustrates the operation of the second partial-topology embodiment of the TBRPF protocol when a communication link **142** between nodes B and D in the subnet **10** fails. The minimum-hop-path tree for source node B before the link failure is shown with solid arrows; the minimum-hop-path tree for source node C is shown with dashed arrows. As shown node A selects node B as parent for source nodes B, D, and F, and selects node C as parent for source nodes C, E, and F. Therefore, node B reports link-state changes to node A only for links (B, A), (B, C), (B, D), and (D, F), and node C reports link-state changes to node A only for links (C, A), (C, B), (C, E), and (E, G). Neither nodes B or C would report a link-state change affecting link (F, G) to node A. Thus, unlike the full-topology embodiment of the TBRPF, in which each node **14** has link information for every link in the subnet **10**, the nodes **18** of this partial-topology embodiment have link-state information for less than every link in the subnet **10**.

**[0153]** If link (B, D) fails, as shown in **FIG. 5**, node B reports to nodes A and C that link (B, D) has failed (cost=infinity). Node C reports to node A that link (E, D) **144** has been added to node C's minimum-hop-path source tree. After receiving these updates, node A selects node C as its new parent for source nodes D and F, and sends a NEW PARENT message to node C and a CANCEL PARENT message to node B. Node C responds by sending node A an update only for link (D, F), because link (D, F) is the only link in node C's minimum hop-path source tree with node D or node F as the head of a link. For example, node F is the head of the link (F, G), but the link (F, G) is not in node C's minimum-hop-path source tree and is therefore not reported to node A. Although the minimum-hop-path source tree of node A is modified during the update process, node A does not generate any updates because it has no children for any source other than itself (i.e., node A).

**[0154]** TBRPF Protocol Messages

**[0155]** To disseminate link-state updates to the appropriates nodes in the subnet **10**, neighboring router nodes **14** that have established bi-directional links and performed data link to IPv4 address resolution using TBRPF neighbor discovery (as described below) exchange TBRPF protocol messages. The IPv4 addresses are therefore available for use as node IDs in TBRPF protocol messages.

**[0156]** In one embodiment, the TBRPF protocol messages are sent via the User Datagram Protocol (UDP), which requires an official UDP-service port-number registration. The use of UDP/IPv4 provides several advantages over a data link level approach, including (1) IPv4 segmentation/reassembly facilities, (2) UDP checksum facilities, (3) simplified application level access for routing daemons, (4) IPv4 multicast addressing for link state messages.

**[0157]** TBRPF protocol messages are sent to the IPv4 unicast address of a current neighbor or to the "All_TBRPF_Neighbors" IPv4 multicast address, presuming that an official IPv4 multicast address is assigned to "All_TBRPF_Neighbors." In general, a message is sent to the IPv4 unicast address of a current neighbor node if all components of the message pertain only to that neighbor. Similarly, a message is sent to the All_TBRPF_Neighbors IPv4 multicast address if the message contains components which pertain to more than one neighbor neighbors. Nodes **14** are prepared to receive TBRPF protocol messages sent to their own IPV4 unicast address or the All_TBRPF_Neighbors multicast address.

**[0158]** Actual addressing strategies depend on the underlying data link layer, for example, for data links such as IEEE 802.11, a single, multiple access channel is available for all unicast and broadcast/multicast messages. In such cases, since channel occupancy for unicast and multicast messages is identical, it is advantageous to send a single message to the All_TBRPF_Neighbors multicast address rather than multiple unicast messages, even if the message contains components that pertain to only a subset of the current neighbor nodes. In other cases, in which point-to-point receiver directed channels are available, sending multiple unicast messages may reduce contention on the multiple access broadcast channel.

**[0159]** Atomic TBRPF Message Format

**[0160]** **FIG. 6** shows an exemplary embodiment of an individual (atomic) TBRPF protocol message **160** including a message header **162** followed by a message body **164**. Atomic messages may be transmitted either individually or as components of a compound TBRPF protocol message having multiple atomic messages within a single UDP/IPv4 datagram. TBRPF message headers **162** are either 32-bits or 64-bits in length depending on whether the atomic message is BROADCAST or UNICAST.

**[0161]** The message header **162** includes a type field **166**, a version field **168**, a mode field **170**, a number of sources field **172**, an offset field **174**, a link sequence number field **176**, and a receiver identification field **178**, which is used when the mode is defined as UNICAST.

**[0162]** The type filed **166** (e.g., 4 bits) represents the atomic message type. The following are examples of atomic message types:

| | |
|---|---|
| ACK | 1 |
| NACK | 2 |
| NEW_PARENT | 3 |
| CANCEL_PARENT | 4 |
| HEARTBEAT | 5 |
| END_OF_BROADCAST | 6 |
| LINK_STATE_UPDATE_A | 7 |
| LINK_STATE_UPDATE_B | 8 |
| RETRANSMISSION OF BROADCAST | 9 |

[**0163**] The version field **168** (e.g., 3 bits) represents the TBRPF protocol version and provides a transition mechanism for future versions of the TBRPF protocol. Also, the version **168** can assist the node **18** in identifying false messages purporting to be TBRPF protocol messages.

[**0164**] The mode field **170** (e.g., 1 bit) represents the transmission mode for the atomic TBRPF protocol message **160**; the mode is either UNICAST or BROADCAST. UNICAST refers to an atomic message that must be processed by only a single neighbor node. BROADCAST refers to an atomic message that is to be processed by all neighbor nodes. (For IPv4 subnets, UNICAST implies a specific IPv4 unicast address, whereas BROADCAST implies the All_TBRPF Neighbors IPv4 multicast address.) The following exemplary mode bits are defined:

| | |
|---|---|
| UNICAST | 0 |
| BROADCAST | 1 |

[**0165**] Messages of type ACK, NACK, NEW_PARENT, CANCEL_PARENT, RETRANSMISSION_OF_BROADCAST, and END_OF_BROADCAST are sent as UNICAST.

[**0166**] Messages of type LINK_STATE_UPDATE_A and LINK_STATE_UPDATE_B may be sent as either UNICAST or BROADCAST.

[**0167**] The number of sources field **172** (e.g., 8 bits) represents the number of sources "Num_Sources" included in the atomic message **160**. The field **172** takes a value from 1 to 255 for messages of type: NEW_PARENT, CANCEL_PARENT, LINK_STATE_UPDATE_A, and LINK_STATE_UPDATE_B. All other message types are set Num_Sources=0.

[**0168**] The offset field **174** (e.g., 18 bits) represents the offset (in bytes) from the 0'th byte of the current atomic message header **162** to the 0'th byte of the next atomic message header **162** in the "compound message" (described below.) An offset of 0 indicates that no further atomic messages follow. The 18-bit offset field **174**, for example, imposes a 4-kilobyte length restriction on individual atomic messages.

[**0169**] The sequence number field **176** (e.g., 4 bits) represents the link sequence number ("LSEQ") for this TBRPF protocol message **160**.

[**0170**] The receiver identification field **178** (e.g., 32 bits) represents the IPv4 address of the receiving node which is to process this atomic message **160**. All nodes **18** other than the node identified by the identification field **178** do not process this atomic message **160**. This field **178** is used only if the mode field **170** is set to UNICAST.

[**0171**] Compound TBRPF Protocol Message Format

[**0172**] **FIG. 7** shows the format for a compound TBRPF protocol message **180**, which includes multiple (i.e., "N") atomic TBRPF messages **160**, **160'**, **160"** that are concatenated to form the compound message **180** within a single UDP/IPv4 packet. Atomic message headers **162**, in one embodiment, are aligned on 32-bit boundaries, therefore an atomic message body **164** with a non-integral number of 32-bit words includes 1, 2 or 3 padding bytes **182**, **182'** preceding a subsequent message header **162'**, **162"**, respectively.

[**0173**] TBRPF Atomic Message Body Format

[**0174**] The format of the atomic message body **164** depends on the value in the type field **166** in the corresponding message header **162**. The following are exemplary formats for an atomic message body **164**.

[**0175**] ACK

[**0176**] The ACK message carries a NULL message body. A 4-bit acknowledgment sequence number (from 0 . . . 15) is carried in the LSEQ field **176** of the TBRPF message header **162**.

[**0177**] NACK

[**0178**] Each NACK message is a 16-bit vector. Each bit indicates whether each of the last 16 messages prior to the 4-bit sequence number supplied in the LSEQ field **176** of the TBRPF message header **162** was received or lost. As described above, the LSEQ field **176** is set to the sequence number of the last broadcast message received from the neighbor node to which the NACK is being sent.

[**0179**] NEW PARENT

[**0180**] **FIG. 8A** shows an embodiment of an exemplary format **186** for a NEW PARENT message. The format **186** includes one or more source node identity fields **188**, **188'**, **188"** (generally **188**) and one or more corresponding sequence number fields **190**, **190'**, **190"** (generally **190**). Each source node identity field **188** holds a value (e.g., 32 bits) representing the IPv4 address of that source node. Each sequence number field **190** holds a value (e.g., 16 bits) representing a sequence number for the corresponding source node. The **FIG. 8A** shows the message format for an even number of source nodes. **FIG. 8B** shows an alternative ending **192** for the NEW PARENT message format **186** for an odd number of source nodes.

[**0181**] CANCEL PARENT

[**0182**] **FIG. 9** shows an embodiment of an exemplary format **194** for a CANCEL PARENT message. The format **194** includes one or more source node identity fields **196**, **196'**, **196"** (generally **196**) for including the IPv4 address of each source node for which the CANCEL PARENT message applies.

[**0183**] HEARTBEAT

[**0184**] In one embodiment, the HEARTBEAT message has an eight-bit length and holds a sequence number for the broadcast channel.

[0185] END OF BROADCAST

[0186] In one embodiment, the END_OF_BROADCAST message has an eight-bit length and holds a sequence number for the broadcast channel.

[0187] Link-State Update Messages

[0188] The TBRPF protocol provides two formats for two types of link-state update messages. One type of link-state update message, referred to as type LINK_STATE_UPDATE_A, includes a single sequence number for each source node, and is therefore used only if the updates for all links coming out of the same source have the same sequence number. (For example, periodic updates have this property.) This is done to reduce the message size. The second type of link-state update message, referred to as type LINK_STATE_UPDATE_B, includes a separate sequence number for each link.

[0189] **FIG. 10A** shows an embodiment of an exemplary format **198** for one type of link-state update message, LINK-STATE_UPDATE_A. The format **198** includes one or more link-state updates ("lsuA") **200, 200', 200"** (generally **200**). Each lsuA **200** represents an update message with respect to a particular source node and includes a source node identity field **202, 202', 202"** (generally **202**), a number of neighbor nodes field **204, 204', 204"** (generally **204**), and one or more neighbor node sections **206, 206', 206"** (generally **206**). For each neighbor node listed in a particular lsuA **200**, each neighbor node section **206** includes a neighbor-node identity field **208, 208', 208"** (generally **208**), a sequence number field **210, 210', 210"** for corresponding source nodes and neighbor nodes, and a link metrics field **212, 212', 212"** (generally **212**) for that neighbor node.

[0190] The source node identity field **202** holds a value (e.g., 32-bits) for the IPv4 address of the corresponding source node. The number of neighbor nodes field **204** holds a value (e.g., 16 bits) representing the number of neighbor nodes of the corresponding source node. The neighbor-node identity field **208** holds the IPv4 address of a neighbor node of the corresponding source node. The sequence number field **210** holds a value (e.g., 16 bits) representing a sequence number for the corresponding source and neighbor node. The link metrics field **212** holds a value (e.g., 32 bits) representing the link metrics associated with the neighbor node of the corresponding source node.

[0191] **FIG. 10B** shows an embodiment of an exemplary format **220** for the second type of link-state update message, LINK-STATE_UPDATE_B. The format **220** includes one or more link-state updates ("lsuB") **222, 222', 222"** (generally **222**). Each lsuB **222** represents an update message with respect to a particular source node and includes a source node identity field **224, 224', 224"** (generally **224**), a number of neighbor nodes field **226, 226', 226"** (generally **226**), a sequence number field **228, 228', 228"** (generally **228**), and one or more neighbor node sections **230, 230', 230"** (generally **230**). For each neighbor node listed in a particular lsuB **222**, each neighbor node section **230** includes a neighbor-node identity field **232, 232', 232"** (generally **232**), and a link metrics field **234, 234', 234"** (generally **234**) for that neighbor node.

[0192] The source node identity field **224** holds a value (e.g., 32-bits) for the IPv4 address of the corresponding source node. The number of neighbor nodes field **226** holds a value (e.g., 16 bits) representing the number of neighbor nodes of the corresponding source node. The sequence number field **228** holds a value (e.g., 16 bits) representing a sequence number for the associated with the source and neighbor nodes. The neighbor-node identity field **232** holds the IPv4 address of a neighbor node of the source node. The link metrics field **234** holds a value (e.g., 32 bits) representing the link metrics associated with the neighbor node of the corresponding source node.

[0193] RETRANSMISSION OF BROADCAST

[0194] In brief, a RETRANSMISSION_OF_BROADCAST message provides the retransmission of a compound update message in response to a NACK message. This compound message may contain one or more atomic messages of type LINK_STATE_UPDATE_A or LINK_STATE_ UPDATE_B concatenated together. **FIG. 11** shows an embodiment of an exemplary format **240** of a RETRANSMISSION_OF_BROADCAST message including a message header **162'"** and a compound message **180'**. The message header **162'"**, like the message header **162** of the atomic message format **160** described above, includes a type filed **166'**, a mode field **170'**, a number of sources field **172'**, an offset field **174'**, and a link sequence number field **176'**. The type field **166'** is set to RETRANSMISSION_OF_BROADCAST (e.g., =9), and the number of sources field **172'** is set to 0. The offset field **174'** is the offset (in bytes) from the 0'th byte of the current compound message header to the 0'th byte of the next compound message header **162'** in the RETRANSMISSION_OF_BROADCAST message **240**. A 16-bit offset value enables concatenation of compound messages **180'** up to 64 kilobytes in length.

[0195] As described above, broadcast update messages can be retransmitted on unicast or broadcast channels. For retransmission on a unicast channel, the mode field **170'** is set to UNICAST (e.g., =0) and the atomic message header **162'"** precedes the compound message **180'** The LSEQ field **176'** holds the sequence number corresponding to the unicast channel on which the message is sent. The LSEQ field **176** of each atomic message in the compound message **180'** is the broadcast sequence number that was included in the original (broadcast) transmission of the message. Multiple RETRANSMISSION_OF_BROADCAST messages can be bundled into a compound message **180'** as described above.

[0196] Selecting a Routing Path for Transmitting Packets

[0197] Routing protocols can also be classified according to whether they find optimal (shortest) routes or sub-optimal routes. By not requiring routes to be optimal, it is possible to reduce the amount of control traffic (including routing updates) necessary to maintain the routes. However, optimal routes are desirable because they minimize delay and the amount of resources (e.g., bandwidth and power) consumed. The TBRPF protocol computes optimal routes based on the advertised link states; however, the advertised link states themselves may be approximate in order to reduce the frequency at which each link is updated.

[0198] In the full-topology embodiment of the TBRPF protocol, each routing node **14** has complete link-state information. Each routing node **14** then applies a path selection algorithm to compute preferred paths to all possible destinations, and to update these paths when link states are updated. One exemplary path selection algorithm is to

apply Dijkstra's algorithm to compute shortest paths (with respect to cost, c) to all destinations. In other embodiments, the TBRPF protocol can employ any other path selection algorithm. Once preferred paths are computed, the routing table entry for node u is set to the next node on the preferred path to node u. If minimum-hop routing is desired, then the routing table entry for node u can be set to the parent p_i(u).

[0199] 2. Neighbor Discovery

[0200] Each routing node **14** running the TBRPF protocol uses a neighbor discovery protocol to detect the establishment of new links to new neighbor nodes and the loss of established links to existing neighbor nodes. In general, the neighbor discovery protocol dynamically establishes bi-directional links and detects bi-directional link failures through the periodic transmission of HELLO messages. The neighbor discovery protocol is both automatic and continuous, and may include a data link-to-IPv4 address resolution capability. Because the neighbor discovery protocol is responsible for both link state maintenance and data link-to-IPv4 address resolution in the subnet **10**, the neighbor discovery protocol operates as a data-link-level protocol.

[0201] **FIG. 12** shows an exemplary embodiment of a process **250** used by the nodes **18** to perform neighbor discovery. The process uses the following three types of control messages: HELLO, NEIGHBOR, and NEIGHBOR ACK. This embodiment of the neighbor discovery protocol operates as follows. Every each node i in the subnet periodically transmits (step **252**) a HELLO message at predetermined (e.g., HELLO_INTVL=0.5 seconds) timeout intervals. (The HELLO_INTVL value is common to all nodes **18** within the subnet **10**, but different subnets may use different HELLO_INTVL values.) HELLO messages are sent to the data link level broadcast address and includes the identity of transmitting node i.

[0202] A node j receiving a HELLO message from a new neighbor, node i, responds (step **254**) with a NEIGHBOR message containing the identity of node j, sending the NEIGHBOR message to the data link unicast address of the new neighbor node i. Then, upon receiving the NEIGHBOR message, node i sends (step **256**) a NEIGHBOR ACK to node j using the data link unicast address of node j. The NEIGHBOR ACK message contains the identity of node i. The NEIGHBOR and NEIGHBOR ACK messages also contain the current link-level sequence number for the broadcast channel (discussed below). Thus, a link from node i to node j is established by node i receiving a NEIGHBOR packet from node j, and a link from node j to node i is established by node j receiving a NEIGHBOR ACK packet from node i. The link to an existing neighbor is declared to be down if no traffic (including HELLO messages and ACKs) has been received from the neighbor node within a predetermined time interval (e.g., within the last LINK-DOWN_INTVL=2.0 seconds).

[0203] Implementations of this embodiment of the neighbor discovery protocol should detect the event of a data link-to-IP address mapping change for existing links. This may occur in one of the following instances:

[0204] 1. Two or more nodes in the subnet **10** are using the same IP address.

[0205] 2. An existing node in the subnet **10** has changed its data link layer address.

[0206] 3. A new node is now using the IP address of a former node that may have left the subnet **10**.

[0207] In the first case, the implementation should print some form of "duplicate IP address detected" message to the console. In the second and third instances, the cached link state should be updated to reflect the new data link-to-IPv4 address mapping.

[0208] **FIG. 13** shows an exemplary embodiment of a packet format **260** for the HELLO, NEIGHBOR, and NEIGHBOR ACK neighbor discovery protocol messages on the subnet **10**. The data link header for each message is not shown, since it is specific to the underlying data link layer.

[0209] The eight-bit "Type" field **262** indicates the type of message. For example, each message can be identified by the following examples of values in the Type field **262**:

| | |
|---|---|
| HELLO | 10 |
| NEIGHBOR | 11 |
| NEIGHBOR_ACK | 12 |

[0210] The eight-bit "BCAST Seq# field **264** indicates a sequence number from 0 . . . 15 (4 bits), used in NEIGHBOR and NEIGHBOR ACK messages as described above. The four address fields (sender hardware address **266**; sender protocol address **268**; target hardware address **270**; target protocol address **272**) facilitate the address resolution process. The fields **266**, **268**, **270**, and **272** contain the following examples of values, based on the type of neighbor discovery message:

[0211] Message type is HELLO

[0212] Sender Hardware Address **266**: data link address of sender

[0213] Sender Protocol Address **268**: IPv4 address of sender

[0214] Target Hardware Address **270**: data link broadcast address

[0215] Target Protocol Address **272**: unused

[0216] Message type is NEIGHBOR

[0217] Sender Hardware Address **266**: data link address of sender

[0218] Sender Protocol Address **268**: IPv4 address of sender

[0219] Target Hardware Address **270**: sender H/W Address from received HELLO

[0220] Target Protocol Address **272**: sender IP Address from received HELLO

[0221] Message type is NEIGHBOR ACK

[0222] Sender Hardware Address **266**: data link address of sender

[0223] Sender Protocol Address **268**: IP address of sender

[0224] Target Hardware Address **270**: sender H/W address from NEIGHBOR

[0225] Target Protocol Address **272**: sender IP address from NEIGHBOR

[0226] Usage of the other fields **274**, **276**, **278**, and **280** in the packet **260** are described "An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses To 48.Bit Ethernet Addresses For Transmission On Ethernet Hardware," by David C. Plummer, Request for Comments (RFC) No. 826, November 1982.

[0227] Reduced Overhead Hello Protocol

[0228] Another embodiment of the neighbor discovery protocol, hereafter referred to as Reduced Overhead Hello Protocol (ROHP), is suited for MANETs. As described further below, the ROHP is suited for MANETs because the protocol can operate correctly although an asymmetric (unidirectional) link may exist between any two nodes at any time, link states may change frequently due to node mobility and interference, and the channel may be noisy so that not all transmitted packets are successfully received by all neighbor nodes. An objective of ROHP is to allow each node **18** in the MANET **10** to quickly detect the neighbor nodes with which that node **18** has a direct and symmetric link, (i.e., a bi-directional link such that the node at each end of the link can hear the other node.) The ROHP also detects when a symmetric link to some neighbor no longer exists.

[0229] In brief overview, the ROHP reports each change in the state of a neighbor node (e.g., "heard", "symmetric", or "lost") in only the next K HELLO messages, where K is a small positive integer (e.g., K=3 to 5) such that a node declares a neighbor to be "lost" if it does not receive any complete HELLO message from that neighbor node within a time period equal to K number of time intervals. Each time interval is hereafter referred to as HELLO_INTERVAL, which is for example 0.5 to 2 seconds. In contrast, each HELLO message of conventional neighbor discovery protocols (e.g., as in OSPF and OLSR (Optimized Link-State Routing Protocol)) includes the identities (or addresses) of all neighbors.

[0230] Neighbor Table

[0231] Each node **18** maintains a neighbor table, which has an entry for each known neighbor node and stores state information for that neighbor node. An entry for neighbor node B, for example, contains the following variables:

[0232] state(B): The current state of the link to neighbor node B, which can be "heard", "symmetric", or "lost".

[0233] hold_time(B): The amount of time (in seconds) remaining before state(B) must be changed to "lost" if no further complete HELLO message from B is received.

[0234] counter(B): The number of subsequent HELLO messages that include the identity of the neighbor node B in the list corresponding to state(B).

[0235] The entry for neighbor node B may be deleted from the table if state(B) remains equal to "lost" for a period not less than K*HELLO_INTERVAL.

[0236] Three possible states of a neighbor node B have the following meaning at node A:

[0237] "Heard": A complete HELLO message was received from neighbor node B within the last K*HELLO_INTERVAL seconds, but it is unknown whether neighbor node B can hear node A.

[0238] "Symmetric": Nodes A and B can hear each other.

[0239] "Lost": No complete HELLO message has been received from neighbor node B within the last K*HELLO_INTERVAL seconds.

[0240] Sending HELLO Messages

[0241] Each node **18** sends a HELLO message periodically every HELLO_INTERVAL seconds, possibly with a small jitter to avoid repeated collisions. Because of message size limitations that may be imposed by the MANET **10**, a HELLO message may be too large to send within one packet, in which case, the sending node **18** sends the HELLO message in multiple packets within a period equal to the HELLO_INTERVAL. Depending on the implementation of the ROHP, the receiving node may or may not be able to extract information from a partially received HELLO message.

[0242] A HELLO packet sent by a node includes the following information:

[0243] 1). The identity (e.g., IP address) of the sending node.

[0244] 2). A list of all neighbor nodes that recently changed to the "heard" state. More specifically, a list of identities of neighbor nodes B such that state(B)="heard" and counter(B)>0.

[0245] 3). A list of all neighbor nodes that recently changed to the "symmetric" state. More specifically, a list of identities of neighbor nodes B such that state(B)="symmetric" and counter(B)>0.

[0246] 4). A list of all neighbor nodes that recently changed to the "lost" state. More specifically, a list of identities of neighbor nodes B such that state(B)="lost" and counter(B)>0.

[0247] Whenever a neighbor node B is included in one of the above three lists, counter(B) decrements by 1. As a result, each state change is included in at most K HELLO messages, and in some cases (as described below) is not included in any HELLO message. HELLO messages can also contain other information, as discussed below.

[0248] Receiving a HELLO Message

[0249] **FIG. 14** shows an exemplary embodiment of a process by which each node **18** operating according to the ROHP neighbor discovery processes a received HELLO message. In step **288**, a node (referred to as receiving node A) receives a partial or complete HELLO message. Because a HELLO message must be transmitted within a time interval of length HELLO_INTERVAL, the receiving node A declares the HELLO message to be partial if not all of its parts have been received within a time interval of this length. If the HELLO message is complete and an entry for neighbor node B does not exist in the table, the receiving node A creates (step **290**) such an entry with state(B)="lost". If the

HELLO message is complete, receiving node A also sets (step **292**) the variable hold_time(B) to K*HELLO_INTERVAL. The value of the variable hold_time(B) decreases to 0 (expires) if no HELLO message from neighbor node B is subsequently received within K*HELLO_INTERVAL seconds. When hold_time(B) expires, the receiving node A sets state(B) to "lost" and counter(B) to K. This indicates that the receiving node A is to include the identity of node B is in the list of "lost" neighbor nodes in the transmission of the next K HELLO messages or until state(B) changes again (whichever occurs first).

[**0250**] The receiving node A then performs (step **294**) an action based on whether the received HELLO message is complete, whether the receiving node A appears in a list within the received HELLO message, and if so, which list, and the current state of the neighbor node B (i.e., state(B)). The actions performed by the receiving node A are summarized in Table 3 below.

TABLE 3

| | Action | | | |
|---|---|---|---|---|
| state(B) | Receiving node A is not in any list | Receiving node A is in "lost" list | Receiving node A is in "heard" list | Receiving node A is in "symmetric" list |
| lost | if msg is complete, set state(B) to "heard" and counter (B) to K | if msg is complete, set state(B) to "heard" and counter (B) to K | If msg is complete, set state(B) to "symmetric" and counter (B) to K | If msg is complete, set state(B) to "symmetric" and counter (B) to K |
| heard | No action | No action | Set state(B) to "symmetric" and counter(B) to K | Set state(B) to "symmetric" and counter(B) to K |
| symmetric | No action | Set state(B) to "heard" and counter (B) to 0 | If counter(B) = 0, set counter (B) to K | Set counter(B) to 0 |

[**0251**] Accordingly, upon receiving a complete or partial HELLO message from neighbor node B, the action performed by the receiving node A is as follows.

[**0252**] 1. If state(B)="lost" and the HELLO message is complete, and if the message does not include node A in any list or includes node A in the "lost" list, then set state(B) to "heard" and counter(B) to K. If state(B)="lost" and the HELLO message is complete, and if the message includes node A in the "heard" or "symmetric" list, then set state(B) to "symmetric" and counter(B) to K.

[**0253**] 2. If state(B)="heard" and the message includes node A in the "heard" list, then set state(B) to "symmetric" and counter(B) to K. If state(B)="heard" and the message includes node A in the "symmetric" list, then set state(B) to "symmetric" and counter(B) to 0. (In this case, the receiving node A need not include node B in its HELLO messages, since both nodes A, B already know that the link is symmetric.)

[**0254**] 3. If state(B)="symmetric" and the message includes node A in the "heard" list and counter(B)=0, then set counter(B) to K. If counter(B)>0, then counter(B) need not be set to K, because the "heard" entry is just a repeat of one that was included in a recently received HELLO message from B. If state(B)="symmetric" and the message includes node A in the "symmetric" list, then set counter(B) to 0. (Both nodes know that the link is symmetric.) If state(B)="symmetric" and the message includes receiving node A in the "lost" list, then set state(B) to "heard" and counter(B) to 0. (node B cannot hear node A, but node A can hear node B.) Note that a complete HELLO message must be received in order to create a new entry in the neighbor node table or to change the state of a neighbor node from "lost" to "heard" or to "symmetric." This prevents the creation of a link that has poor quality.

[**0255**] Variations of ROHP

[**0256**] In other embodiments, HELLO messages can be augmented to include enough information to inform each neighbor node of the set of neighbor nodes with which the sending node has symmetric links. This can be accomplished by setting the counter(B) to K, rather than to 0 (see case 2 above), so that node B is included in the "symmetric" list of the next K HELLO messages, even though nodes A and B already know that the link between them is symmetric.

[**0257**] In addition, node A can inform any new neighbor node of the set of neighbor nodes with which node A has symmetric links. Node A can distribute this information by (a) including the set of all neighbor nodes to which symmetric links exist in the next K HELLO messages, whenever the state of the new neighbor node changes to "symmetric"; or (b) sending this information in a separate message that is unicast reliably to the new neighbor node.

[**0258**] HELLO messages can also be augmented to include other information, such as link metrics, sequence numbers, states of non-adjacent links, time stamps, designated routers, special relays, and other data.

[**0259**] The ROHP can be used in conjunction with any routing protocol that uses HELLO messages for neighbor discovery, such as TBRPF (described herein), OSPF, and OLSR. An advantage of the ROHP over existing neighbor discovery protocols, such as those discovery protocols used within OSPF and OLSR, is that ROHP employs HELLO messages that on average are smaller than such neighbor discovery protocols because each neighbor state change observed by a node is included in at most K HELLO messages (unlike OSPF and OLSR), resulting in reduced communications overhead and bandwidth consumption. In addition, because HELLO messages are smaller, they can be sent more frequently, resulting in a faster detection of topology changes.

[**0260**] 3. IPv6-IPv4 Compatibility Address Format

[**0261**] Referring back to the subnet **10** in **FIG. 1**, assume, for example, that the nodes **18** of subnet **10** belong to one domain, and that the gateway **16** is the border gateway **16** for that domain. Assume also that both the IP host A **12** and the gateway **16** are IPv6 nodes **18**, and that the other nodes **18** in the subnet **10** are IPv4 nodes, without any IPv6 routing capability. Any route taken by packets sent by the IP host A **12** to the server **40** on the Internet **30** necessarily traverses IPv4 infrastructure to reach the gateway **16**. To communi-

cate across the subnet **10** with the heterogeneous IP infrastructure, the IP host **12** and the gateway **16** use an aggregatable, global, unicast addresses, hereafter referred to as an "IPv6-IPv4 compatibility address."The use of IPv6-IPv4 compatibility addresses enables IPv6 nodes (1) to forward IPv6 packets across native IPv6 routing infrastructure or (2) to automatically tunnel IPv6 packets over IPv4 routing infrastructure without requiring a pre-configured tunnel state. A routing node **14** with an IPv6-IPv4 compatibility address can serve as a router for nodes **18** with native IPv6 addresses (i.e., IPv6 addresses that are not IPv6-IPv4 compatibility addresses) connected to the same link. On behalf of such native IPv6 nodes, the IPv6-IPv4 routing node **14** can automatically tunnel messages across the IPv4 infrastructure of the subnet **10** to reach the border gateway **16**.

[**0262**] **FIG. 15A** shows an exemplary embodiment of a format **300** for IPv6-IPv4 compatibility addresses. The format **300** includes a 64-bit address prefix **302** and a 64-bit interface identifier **304**. The address prefix **302** specifies a standard 64-bit IPv6 routing prefix, such as that described in the Internet RFC (request for comment) #**2374**. The address prefix **302** includes a 3-bit Format Prefix (FP) **303**, which for all IPv6-IPv4 compatibility addresses is set to "001", and aggregation identifiers **305**. Consequently, the format prefix **303** and the topologically correct aggregation identifiers **305** of the IPv6-IPv4 compatibility addresses are the same as those of IPv6 addresses assigned to IPv6 nodes, enabling IPv6 nodes to route IPv6 packets using IPv6-IPv4 compatibility addresses across IPv6 infrastructure. The 64-bit interface identifier **304** is a specially constructed 64-bit global identifier interface identifier (i.e., 64-bit EUI-64).

[**0263**] **FIG. 15B** shows an embodiment of the interface identifier **304** including a 24-bit company identifier **306** concatenated with a 40-bit extension identifier **308**. In one embodiment, the 24-bit company identifier **306** is a special IEEE Organizationally Unique Identifier (OUI) reserved by the Internet Assigned Numbers Authority (IANA) for supporting the IPv6-IPv4 compatibility addresses. The IEEE Registration Authority (IEEE/RAC) assigns the OUI to an organization and the organization owning that OUI typically assigns the 40-bit extension identifier **308**. In **FIG. 15**B, the string of 'c's represents the company-specific bits of the OUI, the bit 'u' represents the universal/local bit, the bit 'g' represents the individual/group bit and the string of 'm's are the extension identifier bits. Here, when the bit 'u' equals 1, the scope of the address is global and when the bit 'u' equals 0, the scope is local.

[**0264**] To support encapsulation of legacy IEEE EUI-48 (24-bit) extension identifier values, the first two octets of the 40-bit extension identifier **308** (i.e., bits **24** through **39** of the address) are set to 0xFFFE if the extension identifier **308** encapsulates an EUI-48 value. Further, the first two octets of the extension identifier **308** are not set to 0xFFFF, as this value is reserved by the IEEE/RAC. All other 40-bit extension identifier values are available for assignment by the addressing authority responsible for a given OUI. Thus, as described further below, the IPv6-IPv4 compatibility address format **300** enables embedding an IPv4 address in an IPv6-IPv4 compatibility address without sacrificing compliance with the EUI-64 bit format.

[**0265**] **FIG. 15C** shows an embodiment of the interface identifier **304** including an OUI field **306**, a type field **310**, a type-specific extension field (TSE) **312**, a type-specific data field (TSD) **314**. As shown, the OUI field **306** includes the OUI of IANA, (e.g., 00-00-5E), with 'u' and 'g' bits. The type field **310** indicates how the TSE **312** and TSD **314** fields are interpreted; in general, the type field **310** indicates whether the interface identifier **304** encapsulates an IPv4 address that is suitable for automatic intra-subnet IPv6-in-IPv4 tunneling. Table 1 shows the interpretations of TSE and TSD for various values in the type field **310**:

TABLE 1

| TYPE Value | (TSE, TSD) Interpretation |
|---|---|
| 0x00-0xFD | RESERVED |
| 0xFE | (TSE, TSD) together contain an embedded IPv4 address |
| 0xFF: | TSD is interpreted based on the value of TSE as shown in TABLE 2 |

[**0266**]

TABLE 2

| TSE Value | TSD Interpretation |
|---|---|
| 0x00-0xFD | RESERVED for future use (e.g., by IANA) |
| 0xFE | TSD contains 24-bit EUI-48 interface identifier |
| 0xFF | RESERVED (e.g., by IEEE/RAC) |

[**0267**] Thus, if an IPv6-IPv4 compatibility address has TYPE=0xFE, the TSE field **312** is treated as an extension of the TSD field **314**, which indicates that the IPv6-IPv4 compatibility address includes a valid IPv6 prefix and an embedded IPv4 address.

[**0268**] If the IPv6-IPv4 compatibility address has TYPE=0xFF, the TSE field **312** is treated as an extension of the TYPE field **310**. When TSE=0xFE, the TSD field **314** includes a 240 bit EUI-48 interface identifier. Thus, the IPv6-IPv4 compatibility address format **300** conforms to all requirements of a 64-bit global identifier (i.e., the EUI-64 format) and supports encapsulation of EUI-48 interface identifiers (i.e., when TSE=0xFE). For example, an existing IANA EUI-48 format multicast address such as:

[**0269**] **01-00-5**E-**01-02-03**

[**0270**] is written in the IANA EUI-64 format as:

[**0271**] **01-00-5**E-FF-FE-**01-02-03**.

[**0272**] Other values for TYPE and, hence, other interpretations of the TSE and TSD fields **312**, **314** are reserved for future use.

[**0273**] **FIG. 15D** shows a specific example of an IPv6-IPv4 compatibility address **316** for a node **18** with an IPv4 address of **140**.**173**.**189**.**8**. This IPv4 address may be

assigned an IPv6 64-bit address prefix **302** of **3**FFE:**1**a**05**:**510**:**200**::/**64**. Accordingly, the IPv6-IPv4 compatibility address **316** for this IPv4 node is expressed as:

[**0274**] **3**FFE:**1**a**05**:**510**:**200**:**0200**:**5**EFE:**8**CAD:**8108**

[**0275**] In an alternative form, the IPv6-IPv4 compatibility address **316** with the embedded IPv4 address is expressed as:

[**0276**] **3**FFE:**1**a**05**:**510**:**200**:**0200**:**5**EFE:**140**.**173**.**189**.**8**

[**0277**] Here, the least significant octet of the OUI (**02**-**00**-**5**E) in the interface identifier **304** is 0x02 instead of 0x00 because the bit 'u' is set to 1 for global scope.

[**0278**] Similarly, the IPv6-IPv4 compatibility addresses for the link-local and site-local (i.e., within the subnet **10**) variants, respectively, of are:

[**0279**] FE**80**::**0200**:**5**EFE:**140**.**173**.**189**.**8**

[**0280**] FEC**0**::**200**:**0200**:**5**EFE:**140**.**173**.**189**.**8**

[**0281**] As previously noted, the IPv6-IPv4 compatibility address format **300** enables IPv6 nodes to tunnel IPv6 packets through a one-time IPv6-in-IPv4 tunnel across IPv4 routing infrastructure. **FIG. 15E** shows an embodiment of a packet header **320** used for tunneling IPv6 packets using IPv6-IPv4 compatibility addresses across IPv4 routing infrastructure. In this embodiment, the header **320** includes a 20-byte IPv4 header **322** and a 40-byte IPv6 header **324**. The IPv6 header **324** includes an IPv6 address **329** of the node that is the source of the IPv6 packet and an IPv6-IPv4 compatibility address **316** associated with the final IPv6 destination node. The IPv4 header **322** includes the IPv4 address **326** of the dual-stack node that "re-routes" the IPv6 packet by tunneling the IPv6 packet through the IPv4 routing infrastructure. The IPv4 header **322** also includes the IPv4 address **328** of an IPv4 destination node that typically is the same as the IPv4 address embedded within the IPv6 destination address' IPv6-IPv4 compatible interface identifier **304**. Alternatively, the IPv4 address **328** can be the IPv4 address **328** of the next-hop IPv6 gateway that has a path to the final IPv6 destination address and, therefore, can forward the IPv6 packet towards the final IPv6 destination node.

[**0282**] Upon receiving the tunneled IPv6 packet, the IPv4 destination node determines that the IPv6 header **324** includes an IPv6-IPv4 compatibility address **316** and can route the IPv6 packet to the IPv6 destination node identified by that IPv6-IPv4 compatibility address.

[**0283**] Address Aggregation

[**0284**] One advantage of embedding an IPv4 address in the interface identifier **304** of an IPv6 address is that large numbers of IPv6-IPv4 compatibility addresses **316** can be assigned within a common IPv6 routing prefix **302**, thus providing aggregation at the border gateway **16**. For example, a single 64-bit IPv6 prefix **302** for the subnet **10**, such as **3**FFE:**1**a**05**:**510**:**2418**::/**64**, can include millions of nodes **18** with unique IPv4 addresses embedded in the interface identifier **304** of the IPv6-IPv4 compatibility addresses. This aggregation feature allows a "sparse mode" deployment of IPv6 nodes throughout a large Intranet comprised predominantly of IPv4 nodes.

[**0285**] Globally and Non-Globally Unique IPv4 Addresses

[**0286**] Another advantage is that IPv6-IPv4 compatibility addresses **316** support subnets that use globally unique IPv4 address assignments and subnets that use non-globally unique IPv4 addresses, such as when private address assignments and/or network address translation (NAT) are used.

[**0287**] Non-globally Unique IPv4 Addresses

[**0288**] IPv4 addresses need not be globally unique but may be allocated through a private network-addressing scheme that has meaning only within the context of that domain. IPv6-IPv4 compatibility addresses for private IPv4 addresses set the 'u' bit to 0 for local scope. For example, a node with the private, non-globally unique IPv4 address **10**.**0**.**0**.**1** can be assigned the IPv6-IPv4 compatibility address of

[**0289**] **3**FFE:**1**a**05**:**510**:**200**:**0000**:**5**EFE:**10**.**0**.**0**.**1**,

[**0290**] which uses the same example IPv6 64-bit prefix and IANA OUI (**00**-**00**-**5**E) described above with the 'u' bit in the EUI-64 interface identifier indicating that this is a local address

[**0291**] Routing with IPv6-IPv4 Compatibility Addresses

[**0292**] By embedding an IPv4 address in the interface identifier **304** of an IPv6-IPv4 compatibility address **316**, IPv6 packets can be routed globally over the IPv6 infrastructure or tunneled locally across portions of the IPv4 infrastructure of the subnet **10** that have no IPv6 routing support. Thus, the compatibility-addressing scheme supports heterogeneous IPv6/IPv4 infrastructures in transition with incremental deployment of IPv6 nodes within the subnet **10**.

[**0293**] Intra-domain Routing

[**0294**] **FIG. 16** shows an exemplary embodiment of an intra-domain routing process **330** by which a routing node **14**, configured with IPv6 and IPv4 routing tables, routes a packet having the IPv6-IPv4 compatibility address. Upon receiving the packet, the routing node **14** has IPv6 node software that checks (step **332**) for the special IETF OUI **306** and the type field **310** encapsulated in the interface identifier **304** If the software finds the special OUI **306** and the value of 0×FE in the type field **310**, this means that the received packet has an IPv6 prefix and an embedded IPv4 address.

[**0295**] The routing node **14** then determines (step **334**) if any IPv6 routing information leads to the destination node **18**; that is, if the IPv6 routing table has an entry for 'default' (i.e., the default gateway) or for the IPv6 prefix of the destination node **18** If such an entry is found, the router **14** determines (step **336**) whether there is a path through IPv6 routing infrastructure to the gateway **16** for the IPv6 prefix of the destination node **18** If there is such an IPv6 path, then the router **14** sends (step **338**) the packet as an IPv6 packet to the IPv6 gateway **16** for that IPv6 prefix.

[**0296**] If no such IPv6 path to the gateway **16** through IPv6 routing infrastructure exists, the routing node **14** construes (step **340**) the last four bytes of the extension identifier **308** as an IPv4 address embedded in the IPv6-IPv4 compatibility address. The routing node **14** then determines (step **342**) if the IPv4 routing table includes an entry for a prefix of the embedded IPv4 address of the destination.

[0297] Upon finding such an entry, the routing node **14** encapsulates (step **342**) the IPv6 packet for tunneling through the IPv4 routing infrastructure using the embedded IPv4 address as the destination for the tunneled packet. (The general format for an encapsulated packet is shown in **FIG. 15E**.) One technique for automatically tunneling the IPv6 packet is described in "Transition Mechanism for IPv6 Hosts and Routers," by R. Gilligan and E. Nordmark, draft-ietf-ngtrans-mech-04.txt (work in progress). This technique can also be applied to the IPv6-IPv4 compatibility address. This implies that the gateway **16** also uses IPv6-IPv4 compatibility addresses.

[0298] Inter-domain Routing

[0299] Globally Unique IPv4 Addresses without Privacy Concerns

[0300] Where nodes **18** within an heterogeneous IPv6/IPv4 subnet **10** use globally unique IPv4 addresses and where no privacy concerns exist regarding exposure of internal IPv4 addresses to the public Internet, messages may be routed across domain boundaries using the same routing process **330** described above in **FIG. 16**.

[0301] Globally Unique IPv4 Addresses without Privacy Concerns

[0302] One advantage of the IPv6-IPv4 compatibility address format **300** is that the format **300** does not necessarily expose the true identification of the sending node, if an administrative authority for the subnet **10** wishes to enforce a policy of not exposing internal IPv4 addresses outside of the subnet **10**. To accomplish this, the administrative authority configures the border gateway **16** of the subnet **10** to perform a type of "reverse network address translation," which transforms the IPv6-IPv4 compatibility address interface identifier **304** with embedded IPv4 address of the sending node into an anonymous ID for inter-domain routing outside the subnet **10**. Within the subnet **10**, the fully qualified IPv6-IPv4 compatibility address interface identifier **304** with the embedded IPv4 address of the sending node is still used to enable automatic IPv6-in-IPv4 tunneling, and the intra-domain routing of IPv6 packets follows the process **330** described above.

[0303] In one embodiment, the border gateway **16** advertises an IPv6 prefix **302** of **2002**::/**16** and the IPv6 prefix **302** of **2002**:V4ADDR/**48** where'V4ADDR' is the globally unique embedded IPv4 address of the border gateway **16**. IPv6-IPv4 compatibility addresses within the subnet **10** are constructed as the concatenation of a **2002**:V4ADDR/**48** prefix, a 16-bit SLA ID, and a 64-bit EUI64 interface identifier **304** as described above.

[0304] For example, if the IPv4 address of the border gateway is **140**.**173**.**0**.**1**, the IPv4 address of the IPv4 node within the subnet **10** is **140**.**173**.**129**.**8** and the node resides within SLA ID 0x001, the IPv6-IPv4 compatibility address **316** within the subnet is constructed as:

[0305] **2002**:**8**CAD:**1**:**1**:**0200**:**5**EFE:**8**CAD:**8108**,

[0306] where the '2002:' is a predetermined prefix associated with the reverse network address translation; the '8CAD:1:' is the IPv4 address (**140**.**173**.**0**.**1**) of the border gateway **16**; the second '1:' is the SLA ID; the '0200:5EFE' is the IANA-specific OUI (with the 'u' bit set to global scope and the type field **310** indicating that the compatibility address includes an embedded IPv4 address; and the '8CAD:8108' is the embedded IPv4 address (**140**.**173**.**129**.**8**) of the internal IPv4 node.

[0307] The border gateway **16** performs "reverse network address translation" using an identifier not vulnerable to eavesdropping. The border gateway **16** maintains a mapping of the identifier to the actual IPv4 address of the IPv4 node in order to map messages from destinations back to the actual IPv4 node within the subnet **10**. For example, if the border gateway **16** replaced the IPv4 address **140**.**173**.**129**.**8** with the identifier value: 0x00000001, the IPv6-IPv4 compatibility address outside the subnet **10** is constructed as:

[0308] **2002**:**8**CAD:**1**:**1**:**0000**:**5**EFE:**0**:**1**

[0309] Here: again the least significant octet of the EUI-64 interface identifier **304** has the 'u' bit set to 0 to indicate that the embedded IPv4 address is not globally unique.

[0310] The IPv6-in-IPv4 tunneling for inter-domain routing then derives the IPv4 source address from the IPv4 address of the numerous separate tunnel transitions for an IPv6 packet traveling from a sending node to a destination node. The transitions include (1) intra-domain tunnels from the IPv6 sending node through routers along the path to the border gateway for its domain, (2) inter-domain tunnels from the sending node's border gateway through other transit routers along the path to a border gateway for the destination, and (3) intra-domain tunnels from the destination node's border gateway through intra-domain routers along the path to the destination node itself. Thus, IPv4 addresses within the subnet are exposed across the public Internet **30**.

[0311] Non Globally Unique IPv4 Addresses

[0312] Embodiments of the subnet **10** that use private, non-globally unique IPv4 addresses require a border gateway **16** that implements an inter-domain routing function as described above. For example, if the IPv4 address of the border gateway **16** is **140**.**173**.**0**.**1**, the IPv4 address of an IPv4 node within the subnet **10** is **10**.**0**.**0**.**1**, and the IPv4 node resides within SLA ID 0×001, the IPv6-IPv4 compatibility address within the subnet **10** is constructed as:

[0313] **2002**:**8**CAD:**1**:**1**:**0000**:**5**EFE:**0A00**:**1**,

[0314] where again the least significant octet of the EUI-64 interface identifier **304** has the 'u' bit set to 0 to indicate that the embedded IPv4 address '0A00:1' (**10**.**0**.**0**.**1**) is not globally unique.

[0315] The administrative authority for such embodiments of the subnet **10** may institute a policy that permits exposing non-globally unique IPv4 addresses to the public Internet **30**. In this case, the reverse network address translation is unnecessary, but might be used to protect against eavesdropping on the non-globally unique addresses.

[0316] Additional Routing Considerations

[0317] In a different embodiment than that described in **FIG. 16**, each host **12** or router **14** that sends an IPv6 packet to an IPv6-IPv4 compatibility destination address follows the following process:

[0318] If the 64-bit IPv6 prefix of the IPv6-IPv4 compatibility destination address matches the 64-bit

IPv6 prefix of one of the network interfaces, tunnel the packet through IPv4.

[0319] Otherwise, route the packet through IPv6.

[0320] From the above sending process, a sending node that does not have an interface which shares a common 64-bit routing prefix with the packet's IPv6-IPv4 compatibility destination address sends the packet to the next-hop gateway determined by an IPv6 routing table lookup. In short, when a sending node does not have an interface which shares a common 64-bit (site-level) routing prefix with an IPv6-IPv4 compatibility destination address, the sending rule is identical to that for a native IPv6 destination address. This decision is independent of whether the sending node has an IPv6-IPv4 compatibility address itself, or whether the sending node even comprises a dual-stack configuration. The sending node can be a native IPv6 node with no legacy IPv4 support.

[0321] When a sending node has an interface which shares a common 64-bit routing prefix with an IPv6-IPv4 compatibility destination address, the sending node must assume that the destination is not directly reachable at the data-link level, although the shared site-level routing prefix implies otherwise. Instead, if the sending node comprises a dual-stack configuration, it automatically tunnels the IPv6 packet to the IPv4 address embedded within the IPv6-IPv4 compatibility destination address' interface identifier. If the sending node is an IPv6-only node that does not comprise a dual-stack configuration, however, it has no means for automatically tunneling the packet via IPv4. In this case:

[0322] If the sending node is the host that originates the packet, the sending node sends the packet to a router that lists the 64-bit prefix in its router advertisements. If no such router exists, the sending node should drop the packet and return a "No route to host" error indication to the originating application. If the sending node is a router that forwards the packet, the sending node drops the packet and sends an ICMPv6 "Destination Unreachable" message to the source

[0323] By implication, the scheme breaks down if a packet with an IPv6-IPv4 compatibility destination address reaches an IPv6-only router that has an interface that shares a common 64-bit routing prefix with the IPv6-IPv4 compatibility destination address. Additional mechanisms to address this issue may be possible, such as allowing dual-stack routers to advertise 96-bit prefixes which incorporate the special 32-bit EUI-64 interface identifier prefix: **0200**:**5**EFE. A sending node can then interpret such an advertisement to mean that the advertising router comprises a dual stack and is capable of intra-site IPv6-in-IPv4 tunneling.

[0324] Incremental IPv6 Deployment Examples

[0325] When deploying an IPv6 node in a subnet that is predominantly IPv4, the embedded IPv4 address within an IPv6-IPv4 compatibility assigned to that IPv6 node does not need to be globally unique. The embedded IPv4 address needs only be topologically correct for and unique within the context of that subnet **10**. Also, when deployed in a predominantly IPv4 subnet, the deployed IPv6 node is unlikely to share a common multiple access data-link with an IPv6 router **14** in the subnet. Because the IPv6 node does not share a common multiple access data-link with the IPv6 router, no router advertisements are available. IPv6-IPv4 compatibility addresses enable the IPv6 node to join the global IPv6 network (i.e., on the Internet **30**) by automatically tunneling IPv6 messages through the intra-site IPv4 routing infrastructure. For this purpose, the deployed IPv6 node requires two pieces of static configuration information: the 64-bit IPv6 network prefix for the subnet **10** and the IPv4 address of the dual-stack IPv6 gateway **16** servicing the subnet **10**. No other pre-configured tunnel state information is required.

[0326] For example, consider a researcher who wishes to configure IPv6 on his existing IPv4-based workstation, but the network administrators for the subnet **10** have not yet configured an IPv6 router for the workstation's LAN. The researcher is aware of a dual-stack IPv6 router elsewhere within the subnet **10** (which may be several IPv4 router hops away from his workstation's LAN) and sets the 64-bit IPv6 address prefix and IPv4 address of the router as configuration information on his workstation.

[0327] This configuration information is used to construct two IPv6-IPv4 compatibility addresses. One is the concatenation of the IPv6 prefix and the IPv4 address of the router to construct the IPv6-IPv4 compatibility address for the router. The researcher's workstation uses this IPv6-IPv4 compatibility address of the router as its default IPv6 gateway address. The second address is the concatenation of the IPv6 prefix and the IPv4 address of the researcher's workstation to construct the IPv6-IPv4 compatibility address which the workstation uses as its own IPv6 source address. The researcher's workstation can now access the global IPv6 Internet **30** by first tunneling messages through the subnet-local IPv4 routing infrastructure to the IPv6 router. The IPv6 router then routes the IPv6 messages. No static configuration information is needed on the IPv6 router on behalf of the researcher's workstation.

[0328] As another example, a network administrative authority wishes to configure IPv6 on an existing IPv4 subnet under their jurisdiction, but the subnet is separated from the IPv6 border gateway **16** for the subnet by other IPv4 subnets, which are not ready for IPv6 deployment. The administrator configures a dual-stack IPv6 router (or routers) for his administrative domain by arranging for SLA (site-level aggregation)-based subnet allocation(s) from the owner of the IPv6 border gateway for the subnet. The administrator further sets the 64-bit IPv6 address prefix and IPv4 address of the border gateway as configuration information on his router. The router(s) for the administrative domain can now access the global IPv6 Internet by first tunneling messages through the site-local IPv4 routing domain to the IPv6 border gateway for the site. Hosts and/or other IPv6 routers which share a common multiple access data-link with the router receive router advertisements from which they can construct native IPv6 addresses with topologically-correct 64-bit prefixes and interface identifiers via address auto-configuration. The IPv6 border gateway for the site need only have routing information that points to the router(s) for the SLA-based subnet allocations.

[0329] Automatic Deprecation

[0330] As seen in the above deployment examples, the IPv6-IPv4 compatibility address format enables incremental IPv6 deployment for hosts and routers within sites that have

incomplete or "sparse" IPv6 coverage at the network infrastructure level. In general, IPv6-IPv4 compatibility addresses are intended for use by nodes **18** that do not receive router advertisements because such nodes **18** do not share a common multiple access data-link with an IPv6 router. When router advertisements become available, such as when an IPv6 router is deployed on a common multiple access data-link shared by the node **18**, the node **18** can discontinue use of its IPv6-IPv4 compatibility address and adopt an IPv4 unicast address using address auto-configuration for a prefix discovered through router discovery. In this way, IPv6-IPv4 compatibility addresses can gradually and automatically disappear as IPv6 nodes become widely deployed within the subnet **10**. The following automatic deprecation rule for hosts and routers using IPv6-IPv4 compatible addresses can be used to transition from the use of IPv6-IPv4 compatibility addresses:

> [**0331**] While no IPv6 router advertisements are received, continue to use the IPv6-IPv4 compatibility address. If router advertisements ensue, discontinue use of the IPv6-IPv4 compatibility address and construct a native IPv6 address based on prefix information carried in the router advertisements.

[**0332**] Address Selection

[**0333**] To ensure efficient routing within the destination's subnet when multiple IPv6 destination addresses alternatives are available, a "second-tier" address selection policy is used for choosing between an IPv6-IPv4 compatibility addresses and a native IPv6 address. If multiple alternatives remain after address selection has been applied on the 64-bit routing prefixes, and if at least one of the remaining alternatives is constructed with a native IPv6 interface identifier (one that does not contain an embedded IPv4 address), select a native IPv6 address. Otherwise, select an IPv6-IPv4 compatible address.

[**0334**] 4. Updating Information Upon Resuming Interrupted Communications

[**0335**] Referring again to **FIG. 1**, assume that the mobile node **12**, hereafter "client 12", and the server **40** are communicating over a route or path through the subnet **10** that includes one or more wireless links. Movement by the client **12** or by another node **14** in the subnet **10** may cause the client **12** to move in and out of communication range of the subnet **10**. For example, the client **12** may move to a new position in the subnet **10** (as indicated by arrow **27**) or to the foreign subnet **20** (as indicated by arrow **29**). While moving, the client **12** may break current a communication link (e.g., link **24**) to the subnet **10** and be out of range of all routing nodes **14** within the subnet **10**. As another example, the node B may move out of range of the client **12**, placing the client **12** out of range of the subnet **10** if the client **12** is not within range of another routing node **14** in the subnet **10**. Consequently, the client **12** is not communicating with the server **40** and may not access information, particularly updated information, from the server **40**. The inability to obtain updated, timely information may cause resources associated with the client **12** to be inefficiently used and adversely affect the operation of the client **12**.

[**0336**] To lessen any adverse impact of client movement, the client **12** and the server **40** can (1) use message queues to store communications affected by an interruption for subsequent transmission if communications between the client **12** and the server **40** are resumed; and (2) use bandwidth adaptation techniques to maintain a persistent connection between the client **12** and the server **40** although a route between the client **12** and the server **40** is momentarily lost.

[**0337**] Message Queues

[**0338**] The client **12** may register an interest in certain data on server **40**. In one embodiment, the data are an object. Objects, as will be appreciated by those skilled in the art, are generally programming units that include data and functionality, and are instances of classes. For example, the client **12** may be interested in updated information pertaining to particular objects. In one embodiment, server **40** may also include meta-objects, which are objects that have no physical representation, and are classes with methods and attributes that serves as a factory to create new objects. Meta-objects may not be instantiated, (i.e., meta-objects generally do not provide a representation of a physical object). Instead, meta-objects may serve as templates from which objects that represent physical objects are constructed.

[**0339**] The client **12** maintains local copies of objects on the server **40** and updates these objects, as necessary, when communicating with the server **40**. Relevant objects associated with the server **40** may be replicated, (i.e., databases associated with server **40** may be replicated), on the client **12** to provide the client **12** with the local copies of objects. Local copies provide the client **12** with access to relatively up-to-date information should the client **12** move out of the communications range of the subnet **10**, interrupting communications with the server **40**. While the link **24** is broken, the local copies of the objects, which are active entities, can continue to run on the client **12**. Then when the client **12** moves back into the communications range of subnet **10** or the subnet **20** and reestablishes communications with the server **40** over the same or a different route, the server **40** can provide the client **12** with updated, or current, information. That is, the server **40** may update the local copies of the objects that are present on the client **12** in, for example, a local cache.

[**0340**] In general, because communications between the server **40** and the client **12** are synchronous, the server **40** is aware of all objects that are associated with the client **12** Server **40** may then be able to save state information associated with the client **12** Therefore, server **40** may restore the current state of the client **12** as appropriate (e.g., when lost link **24** is re-established). It should be appreciated, however, that server **40** is generally not aware of any semantics with regards to objects. Rather, the server **40** is only aware that objects have been updated, and, further, that the corresponding updates should be forwarded to the client **12** as appropriate.

[**0341**] Server **40** includes an object list that is a list of all objects associated with the server **40** and which are to be updated. In other words, the object list is a queue of object updates. The client **12** may communicate substantially with the server **40** after the client **12** is registered with respect to server **40**. That is, client **12** may send commands to server **40**. In one embodiment, such commands include lists of topics in which the client **12** is interested. The server **40** may send update messages to the client **12** to indicate that certain

objects on the client **12** should be updated such that the states of the objects on the client **12** are consistent with the states of the corresponding objects on the server **40**. The client **12** also includes an object list, (i.e., a client object list), that contains substantially all objects that are associated with the client **12** In general, the new client object list contains all objects, which are associated with the server **40** and which the client is "interested" in.

[**0342**] The client **12** communicates with the server **40** over a route (or path) through the subnet **10** or subnet **20** determined by the routing nodes **14**. The client **12** may transmit data to the server **40** directly or through a message queue. The client **12** queues data on the message queue when, for example, data has been modified and is to be sent to the server **40**. Specifically, when the client **12** creates or modifies data, the data is sent to the server **40** through the message queue. The communications between the client **12** and the message queue may, in the described embodiment, be performed using a potentially unreliable communications link (e.g., wireless link), while the communications between the message queue and server **40** are typically more reliable, (e.g., wired link).

[**0343**] Data is placed on the message queue by the client **12**, and is removed from the message queue by the server **40** or, more specifically, communications software associated with server **40**. Data is removed from the message queue after the data has been successfully received by the server **40**.

[**0344**] When the client **12** creates data (e.g., objects), the client **12** typically associates that data with a unique identifier that is used by the client **12** and the server **40** to identify that data. One example of a unique identifier is a timestamp. The associated timestamp is updated each time the data are updated or modified by the client **12**. A timestamp essentially prevents data conflicts from arising when more than one client attempts to modify that data at a given time. Specifically, timestamps are monotonically increasing such that substantially no data conflicts between unique identifiers can arise. Other embodiments use different mechanisms to uniquely identify data, such as an authorization level that is associated with the users of a particular client; a priority level that is associated with the particular type of data, and the order in which the data are received (e.g., LIFO, FIFO).

[**0345**] Similarly, the server **40** can communicate directly to the client **12** or through a message queue(s), or lists, for storing objects in which the client **12** has indicated an interest. The message queue(s) can be part of or separate from the server **40**. Hence, data may be transmitted to the client **12** from the server **40** through such message queues. In other words, the server **40** may use substantially the same heuristics as the client **12** for sending data. Data is placed on the message queue by the server **40** and removed from the message queue by the client **12**. Again, data is removed from the message queue when the client **12** has successfully received previously removed data.

[**0346**] 5. Adaptive Use of Network Bandwidth

[**0347**] In general, within the subnet **10**, which includes wireless links, a variety of different failures can interrupt or cause a communications outage. For example, a failure may be due to a hardware problem at either the client **12** or the server **40**. A failure may also be the result of a software problem, (e.g., data may be successfully received but acknowledgement of the receipt may fail). Failures may also occur because of problems with links, as mentioned previously. Such failures may include a failure of any link on a route between the client **12** and the server **40**. It should be appreciated that in some cases, more then one failure may occur at any given time.

[**0348**] To adaptively handle interruptions in communications between the client **12** and the server **40**, the internetworking system **2** may run diagnostics to determine the cause of the interruption. After the cause is determined, the system **2** makes corrections that restore communications. Depending upon current system parameters, such adaptive corrections include, but are not limited to attempting (1) to reestablish the same interrupted connection between the client **12** and the server **40**, (2) to establish a connection between the client **12** to a redundant server, or (3) to establish a new connection between the client **12** with the server **40**. Other techniques used alone or in combination with the aforementioned corrections include varying and/or increasing the waiting period between unsuccessful attempts to establish a connection the client **12** and the server **40** and adjusting the length of transmitted packets. Such techniques can be used in response to current bandwidth conditions in the subnet **10**.

[**0349**] When a "network dropout" occurs in the subnet **10**, (e.g., when the client **12** or the server **40** appears to be out of communication with the subnet **10**), standard client-server systems, such as those based upon TCP/IP, typically operate under the assumption that the failure is due to network congestion. As will be understood by those skilled in the art, although a network dropout in a low-bandwidth, wireless subnet may indeed occur as a result of network congestion, the network dropout may also occur for a variety of other reasons including, but not limited to, packet loss due to coverage problems.

[**0350**] Packet loss associated with a network typically involves either the failure of transmission of a packet of data or the loss of some of the data transmitted in a packet. Although packet losses can occur for any number of reasons, packet losses often occur when the client **12** is at least temporarily out of range of the subnet **10** or when a communication link in the route between the client **12** and the server **40** is temporarily interrupted.

[**0351**] By counting the number of packets sent and as the total number of packets acknowledged, the packet loss in a system may be determined. Measuring packet loss enables the manner in which packets are resent or rebroadcast to be dynamically changed such that the resending of packets is substantially optimized with respect to the network.

[**0352**] The internetworking system **2** can use the measure of packet loss to determine the length of packets that are transmitted between the client and the server **40**. For example, if packets with lengths of 1000 bytes experience a 15% packet loss, and packets with lengths of 100 bytes experience a 1% packet loss, then the client **12** and server **40** can tune the length of transmitted packets to minimize the percentage of packets that fail to reach their destination. A factor in determining the packet length is the tradeoff between data throughput and the percentage of packet loss. That is, the smaller the packet length, the greater the percentage of packets that reach their destination, but the

lower the percentage of payload (i.e., data) transmitted in each packet because each packet also carries a number of overhead bits.

[**0353**] Also, the client **12** and the server **40** can dynamically adjust the packet length based upon packet loss measurements that are taken periodically. The client **12** and/or the server **40** can make such packet length adjustments. Further, the client **12** can use a packet length that differs from the packet length used by the server **40**; packets transmitted from the client **12** to the server **40** may take a different route with different bandwidth capabilities than packets transmitted from the server **40** to the client **12**

[**0354**] When a network dropout occurs due to network congestion, repeated attempts may be made to reconnect a "dropped out" client **12** or server **40** to the subnet **10**. If a network dropout occurs due to packet loss and attempts are made to reconnect the client **12** or the server **40** to the subnet **10**, the overall performance of the subnet **10** may degrade to a point where the overall performance of the subnet **10** is unacceptable. That is, attempting to initiate a connection that in fact has not been lost may preclude other connections from being made, thereby preventing the transmission of data which would be made across those connections.

[**0355**] Although a variety of different methods may be used to actually determine if a network dropout is the result of network congestion or of packet loss, such a determination may be made using ongoing statistical measurements. Alternatively, the speed at which data is transmitted may be changed. Typically, when a network dropout is due to packet loss, changing the speed of data transmission often solves the network dropout. However, when network dropout is due to network congestion, changing the speed of data transmission may have no effect and may worsen the throughput.

[**0356**] In order to enable communications to be optimized to reflect actual network conditions, the client-server system may measure the roundtrip time for packet transmission. That is, the amount of time that elapses while a packet of data is transmitted from the client **12** to the server **40**, or vice versa, may be measured. Although the measurements may be used for substantially any purpose, the measurements are often used to characterize the quality of a connection or route between the client **12** and the server **40**. By way of example, for certain networks the duration of a roundtrip can indicate whether a connection is good; short roundtrips are associated with good connections, while long roundtrips are associated with poor connections. The measurements of roundtrip times for a variety of different packets may further be used to statistically determine how long to wait between attempts to resend an unsuccessfully sent packet.

[**0357**] **FIG. 17** shows an embodiment of a process used by the client **12** and the server **40** establish and maintain a persistent connection in a dynamically changing network environment using the above-described bandwidth adaptation techniques. Although the process is described generally from the perspective of the client **12** sending messages to the server **40**, the process also applies to when the server **40** sends messages to the client **12** The process begins (step **350**) by establishing communications between the client **12** and the server **40**. Attempts to establish a connection with the server **40** can begin when the client **12** comes within range of the subnet **10**. The client **12** sends a packet and awaits a reply from the server **40**. The client **12** then waits a specified period of time. If that period elapses without a receiving a response, the client **12** attempts again to establish a connection with the server **40** by sending another packet. Again, the client **12** waits a specified period of time, but the current waiting period is longer than the previous waiting period. By waiting for a longer period (i.e., "backing off") on the subsequent connection attempt, the client **12** is accommodating the dynamic and intermittent quality of mobile wireless networks by giving any response from the server **40** additional time to arrive at the client **12** If the new waiting period also times out, the client **12** sends the packet to the server **40** yet again and waits a still longer period for the reply from the server **40** that establishes the connection.

[**0358**] Under some circumstances, numerous clients **12** (e.g., 200), may arrive within range of the subnet **10** simultaneously, each attempting to establish a connection with the server **40**. For example, consider a military "invasion" scenario in which each participant is equipped with a portable computer capable of establishing a wireless connection to the subnet **10** and thus of communicating with the server **40**. These computers are used to coordinate the military invasion and to assist in pinpointing the position of each individual during the operation. An onslaught of connection attempts could overwhelm the subnet **10** and the server **40** with packets such that only a portion of the computers are able to successfully establish a connection with the server **40**. If each computer then backed off for approximately the same period of time before attempting again to connect to the server **40**, the outcome might be the same; namely, another onslaught of connection packets that impedes some of the computers from establishing a connection. Thus, in one embodiment, the computers are configured so that the back-off period is not the same for each of computers, causing the attempts to connect to the server **40** to be staggered. That is, some computers **12** wait for longer periods than other computers before sending another connection packet to the server **40**.

[**0359**] After communications are established over a route through the subnet **10** that includes one or more wireless links, the client **12** identifies (step **354**) a packet of data that is to be sent to the server **40** as having been sent. After identifying the packet as having been sent, the client **12** transmits (step **358**) the packet through the subnet **10** over a route determined by the routing nodes **14** The packet may be queued in a message queue and sent to the server **40** based on prioritization within the message queue. Similarly, in some embodiments, if a packet is being sent from the server **40** to the client **12**, the packet may also be added to a message queue and sent to the client **12** as determined by priorities assigned within the message queue.

[**0360**] When the packet that is sent includes data that is to be updated, the data may be sent in a variety of different forms. That is, within an object based system, when an object is modified, either the entire object may be sent in a packet, or substantially only the changes to the object may be sent in a packet. By way of example, when an object has a size that is smaller than a predetermined threshold, the entire object is sent in a packet. Alternatively, when the object is larger than that threshold, the updates or changes to that object alone may be sent in a packet, although the entire object may also be sent.

[**0361**] The client **12** then determines (step **362**) whether it has received an acknowledgement from the server **40** indi-

cating that the server **40** received the packet. The client **12** may make the determination after a predetermined amount of time has elapsed. Receipt of the acknowledgment indicates that the packet has been successfully transmitted and received. Hence, the client **12** identifies (step **366**) the packet as being successfully sent and received, and the process of sending data is completed.

[**0362**] If the client **12** instead determines that no acknowledgement of the packet has been received, then this is an indication that there may have been a failure in the network that prevented the server **40** from receiving the packet. Such failures may include, but are not limited to, failures such as a failure of the client **12**, of the server **40**, and a communication link in a route between the client **12** and the server **40**. The failures may also be due to packet loss, and not to a physical failure of any component of the overall system.

[**0363**] When the packet has not been successfully received, then a determination is made (step **370**) as to whether a maximum number of resend tries has been exceeded. The maximum number of attempts to send a packet between the client **12** and the server **40** may generally be widely varied, and is typically determined using statistical models based upon the measured behavior of the overall system. The maximum number of resend tries may be updated at any suitable time during the operation of the overall system. By way of example, the maximum number of resend tries may be calculated and, if necessary, revised, whenever the accumulation of statistical information reaches a certain level.

[**0364**] When the client **12** determines that the maximum number of resend tries has not been exceeded, another attempt is made (step **358**) to send the packet. As mentioned above, the amount of time to wait between resend tries may be based upon statistical calculations based upon information that includes the average roundtrip time for a transmitted packet. An attempt to resend a packet can be successful when the initial failure in sending the packet was the result of packet loss.

[**0365**] On the other hand, if it is determined that the maximum number of resend tries has been exceeded, then attempts to send the packet over the potentially existing link are aborted. When the maximum number of resend tries has been exceeded, and acknowledgement of packet receipt still has not been received, then it is likely that there has been an interruption of the communications link, and that the unsuccessfully sending of data was likely not due to packet losses. Accordingly, an attempt is made (step **374**) to reestablish communications between the client **12** and the server **40**.

[**0366**] A determination is then made (step **378**) as to whether communications between the client **12** and the server **40** have been successfully reestablished. If the determination is that communications have been successfully reestablished, the packet is sent (step **358**). Alternatively, when it is determined that communications between the client **12** and the server **40** have not been reestablished, then another attempt is made (step **374**) to reestablish communications. The number of attempts to reestablish communications between the client **12** and the server **40** may be limited in some cases. In one embodiment, attempts to re-establish communications may be aborted after a predetermined number of attempts have been reached. In other embodiments, when the number of attempts to re-establish communications is limited, the number of attempts that are made may be substantially dynamically determined based on statistical information gathered during the course of communications between client **12** and the server **40**. In still another embodiment, after the number of attempts to establish a connection with the server **40** is reached, the attempts to establish a connection with the client **12** can continue with a different server upon which the data are replicated.

[**0367**] By limiting the number of times attempts made to send data and, further, by not first attempting to re-establish communications which may not actually have been interrupted, the amount of available communications bandwidth in a system may be substantially optimized. The bandwidth may be allocated to making actual connections which are required, rather than wasting the bandwidth by immediately attempting to re-establish communications when such reestablishment is not necessary.

[**0368**] In one embodiment, when an attempt is made to send data from the client **12** to the server **40**, the data is queued on a message queue such that the data is prioritized for transmission to the server. Generally, a single message queue may be shared between multiple servers.

[**0369**] **FIGS. 18A and 18B** are a diagrammatic representation of the updating of an embodiment of a message queue **380** in accordance with an embodiment of the invention. As mentioned above, when data are created or modified, a timestamp accompanying the data is set or modified, respectively. A message queue **380** is shown at time t**3** as including objects that were previously modified and have not yet been accepted by, (i.e., sent to and successfully received by), the server **40**. At time t**3**, at the head **382** of the message queue **380** is an object "obj 1" that was modified at time t**1**, followed by an object "obj 6" that was modified at time t**2**, and an object "obj 9" that was modified at time t**3**. In the described embodiment, the queue **380** is prioritized in a first-in-first-out (FIFO) manner, although priority can instead be based on a variety of other factors.

[**0370**] At time t**5**, the queue **380** further includes an object "obj 3" that was modified at time t**4**. Also at time t**5**, object "obj 6" is being modified such that its corresponding timestamp is updated accordingly. Further, at time t**6**, an object "obj 4" is modified. In one embodiment, the object "obj 6" that was modified at time t**2** is superceded by a version of object "obj 6" that is updated at time t**5**. That is, the object "obj 6" at timestamp t**2** has been replaced with the object "obj 6" at timestamp t**5**. As shown in **FIG. 18**B, at time t**6**, the message queue **380** no longer includes the object "obj 6" that was modified at time t**2** and, instead, includes object "obj 6" that was modified at time t**5**. Within the queue **380**, object "obj 6" in one embodiment does not take the priority of object "obj6" at time t**2**, which has been removed. Instead, object "obj 6" takes a chronological position within the message queue **380**, after object "obj 3" and before after the modification at time t**5** of object "obj 6" . A variety of techniques for prioritizing objects within the message queue **380** is described in co-pending patent application entitled "Method and Apparatus for Updating Information in a Low-Bandwidth Client/Server Object-Oriented System", Ser. No. 09/518,753, which is incorporated by reference herein in its entirety for all purposes.

[**0371**] While the invention has been shown and described with reference to specific preferred embodiments, it should

be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention as defined by the following claims. For example, although the described embodiments illustrate the principles of the invention with respect to wireless networks, such also apply to wire-line networks.

What is claimed is:

**1**. In a multi-hop network including a plurality of nodes, a method for disseminating topology and link-state information over the multi-hop network, comprising:

maintaining a path tree for each source node in the network that can produce an update message, each path tree having that source node as a root node, a parent node, and zero or more children nodes;

receiving an update message from the parent node in the path tree maintained for the source node that originated the received update message, the update message including information related to a link in the network; and

determining whether to forward the update message to children nodes, if any, in the path tree maintained for the source node that originated the update message in response to the information in the received update message.

**2**. The method of claim 1 wherein the information related to the link indicates whether the update message is to be forwarded to other nodes.

**3**. The method of claim 1 wherein the path tree associated with each source node is a minimum-hop-path tree.

**4**. The method of claim 1 further comprising obtaining link-state information from one or more nodes in the path tree maintained for a given source node for use in developing the path tree to that source node.

**5**. The method of claim 1 wherein the link is a wireless communication link.

**6**. The method of claim 1 further comprising sending a new parent message to a node selecting that node as a new parent node for the source node originating the update message.

**7**. The method of claim 6 further comprising receiving from the new parent node in response to the new parent message link-state information associated with the source node that originated the update message.

**8**. The method of claim 7 wherein the new parent message included a serial number and the link-state information received in response to the new parent message is associated with update messages having serial numbers that are greater than the serial number included in the new parent message.

**9**. The method of claim 1 further comprising:

determining that a path through a new parent node for the source node originating the update message has the same number of node hops as the path through the current parent node, and

maintaining the current parent node as the parent node for the given source node.

**10**. The method of claim 1 further comprising:

determining that a path to the source node originating the update message ceases to exist; and

maintaining the current parent node as the parent node for the source node.

**11**. The method of claim 1 further comprising broadcasting the update message to the children nodes if the number of children nodes exceeds a predefined threshold when forwarding the update message to children nodes.

**12**. The method of claim 1 further comprising transmitting the update message to each child node using a unicast mode if the number of children nodes is less than a predefined threshold when forwarding the update message to children nodes.

**13**. The method of claim 1 further comprising:

computing a parent node for each neighbor node and source node; and

determining which neighbor nodes are children nodes for a given source node;

**14**. A network, comprising:

a plurality of nodes in communication with each other over communication links, each node maintaining a path tree for each source node in the network that can produce an update message, each path tree having that source node as a root node, a parent node, and zero or more children nodes,

wherein one of the nodes (i) receives an update message from the parent node in the path tree maintained for the source node that originated the received update message, the update message including information related to a link in the network, (ii) and determines whether to forward the update message to children nodes, if any, in the path tree maintained for the source node that originated the update message in response to the information in the received update message.

* * * * *