IW 7535866

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office

June 29, 2015

THIS IS TO CERTIFY THAT ANNEXED IS A TRUE COPY FROM THE RECORDS OF THIS OFFICE OF THE FILE WRAPPER AND CONTENTS OF:

APPLICATION NUMBER: *09/629,577*
FILING DATE: *July 31, 2000*
PATENT NUMBER: *6,732,147*
ISSUE DATE: *May 04, 2004*

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

EXHIBIT
Ex. 1002
Volume 3

M. TARVER
Certifying Officer
PART (3) OF (6) PART(S)

US005946316A

United States Patent [19]

Chen et al.

[11] **Patent Number: 5,946,316**

[45] **Date of Patent: Aug. 31, 1999**

[54] **DYNAMIC DISTRIBUTED MULTICAST ROUTING PROTOCOL**

[75] Inventors: **Xiaoqiang Chen**, Eatontown; **Vijay Pochampalli Kumar**, Freehold, both of N.J.; **Cauligi Srinivasa Raghavendra**, Pullman, Wash.; **Ramanathan Venkateswaran**, Holmdel, N.J.

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

[21] Appl. No.: **08/785,625**

[22] Filed: **Jan. 17, 1997**

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

[52] **U.S. Cl.** **370/408**; 370/256; 370/432

[58] **Field of Search** 370/256, 390, 370/400, 408, 432, 395; 340/825.02

[56] **References Cited**

U.S. PATENT DOCUMENTS

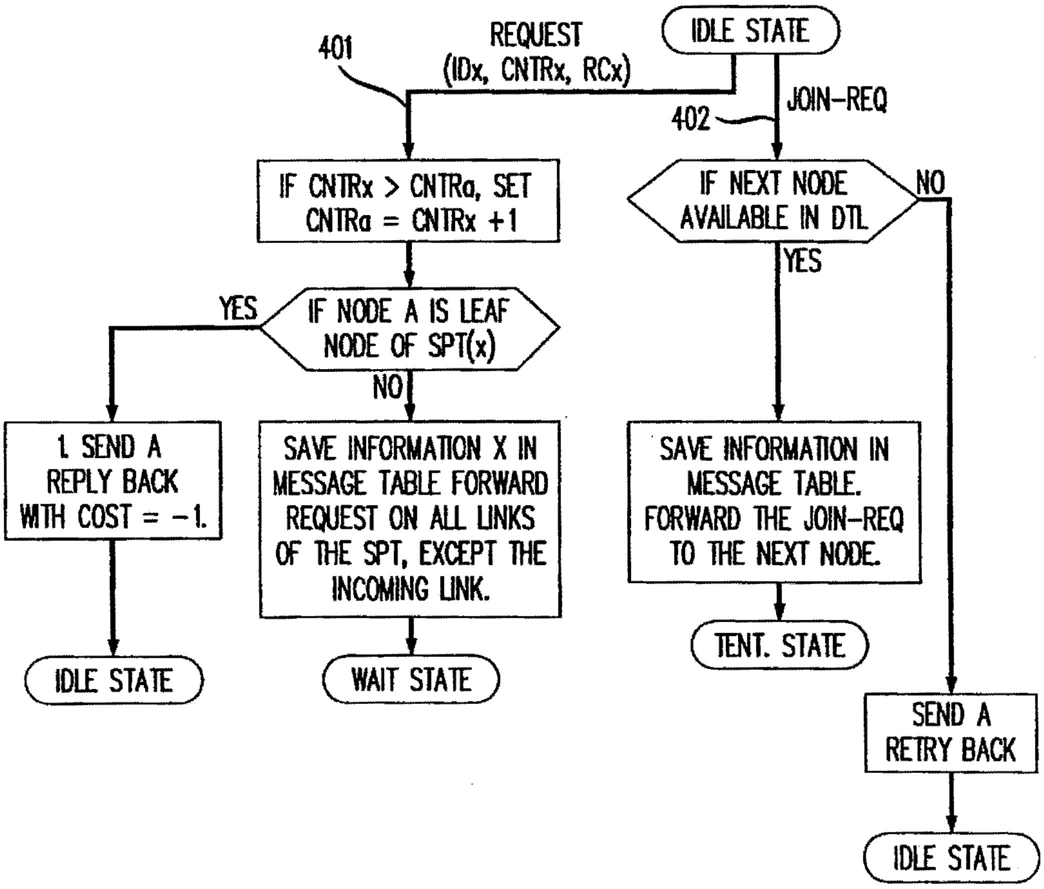
5,291,477	3/1994	Liew	370/408
5,351,146	9/1994	Chan et al.	370/408
5,541,927	7/1996	Kristol et al.	370/408
5,671,222	9/1997	Chen et al.	370/388

Primary Examiner—Alpus H. Hsu
Assistant Examiner—Ricky Q. Ngo
Attorney, Agent, or Firm—Frederick B. Luludis; Daniel J. Piotrowski

[57] **ABSTRACT**

The distribution of multicast information in a communications network formed from a plurality of communications nodes, e.g., ATM switches, is enhanced by providing an efficient mechanism for routing a request to join a multicast connection to an originator of the multicast and an efficient mechanism for then connecting the requester to the multicast connection.

12 Claims, 11 Drawing Sheets



IDLE STATE
401
REQUEST (IDx, CNTRx, RCx)
402
JOIN-REQ
IF CNTRx > CNTRa, SET CNTRa = CNTRx +1
IF NEXT NODE AVAILABLE IN DTL
NO
YES
YES
IF NODE A IS LEAF NODE OF SPT(x)
NO
1. SEND A REPLY BACK WITH COST = −1.
SAVE INFORMATION X IN MESSAGE TABLE FORWARD REQUEST ON ALL LINKS OF THE SPT, EXCEPT THE INCOMING LINK.
SAVE INFORMATION IN MESSAGE TABLE. FORWARD THE JOIN-REQ TO THE NEXT NODE.
TENT. STATE
IDLE STATE
WAIT STATE
SEND A RETRY BACK
IDLE STATE

FIG. 1

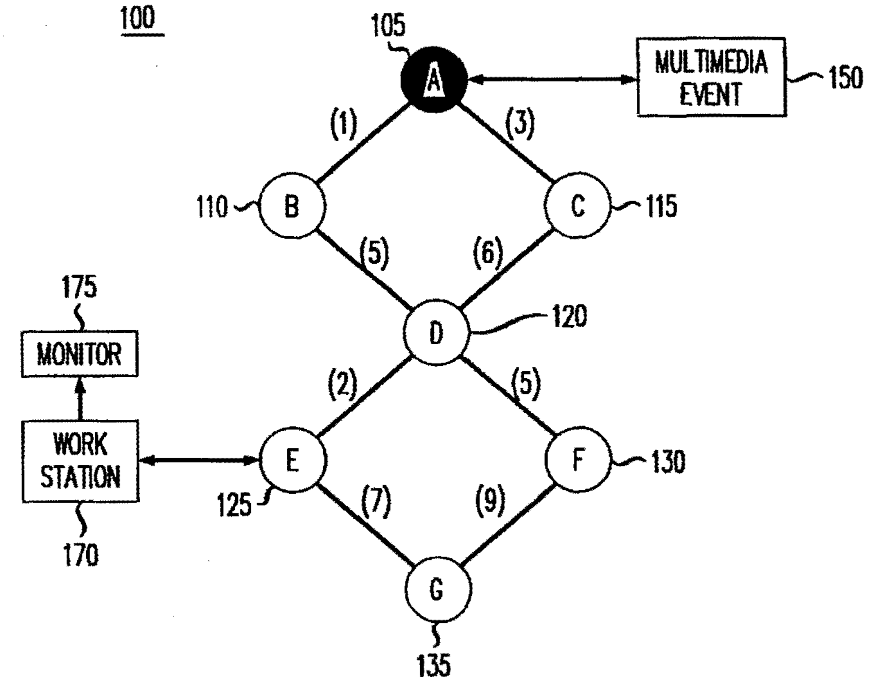

100
105
A
MULTIMEDIA EVENT
150
(1)
(3)
110
B
C
115
(5)
(6)
175
MONITOR
D
120
(2)
(5)
WORK STATION
E
F
130
125
(7)
(9)
170
G
135

FIG. 2

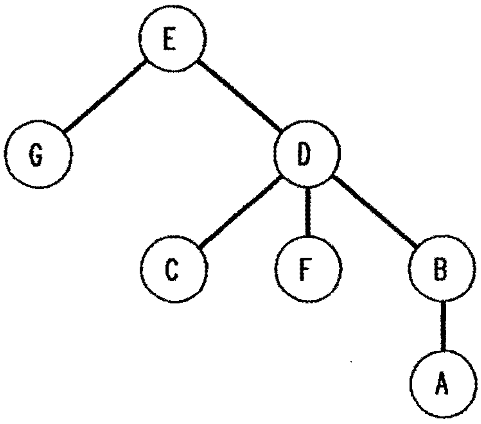

E
G
D
C
F
B
A

FIG. 3

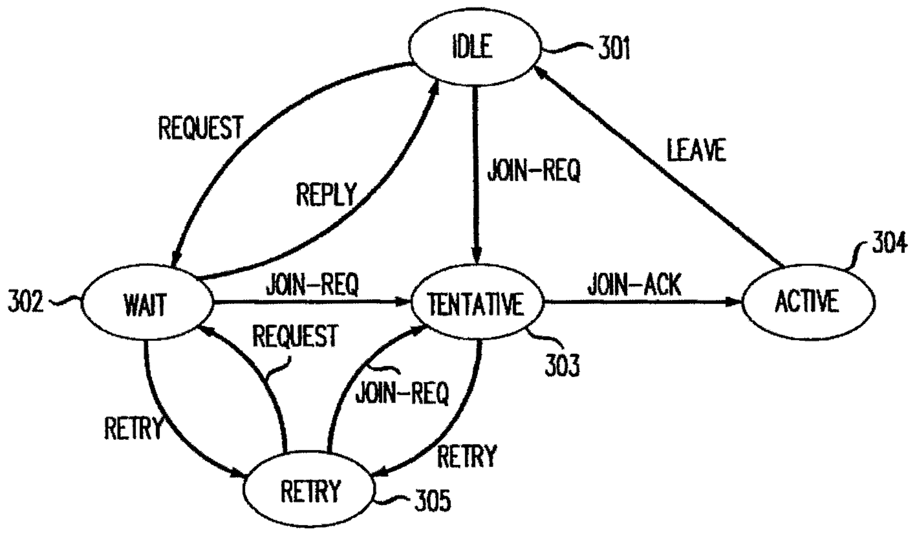

IDLE
301
REQUEST
REPLY
JOIN-REQ
LEAVE
304
302
WAIT
JOIN-REQ
TENTATIVE
JOIN-ACK
ACTIVE
REQUEST
303
JOIN-REQ
RETRY
RETRY
RETRY
305

FIG. 4

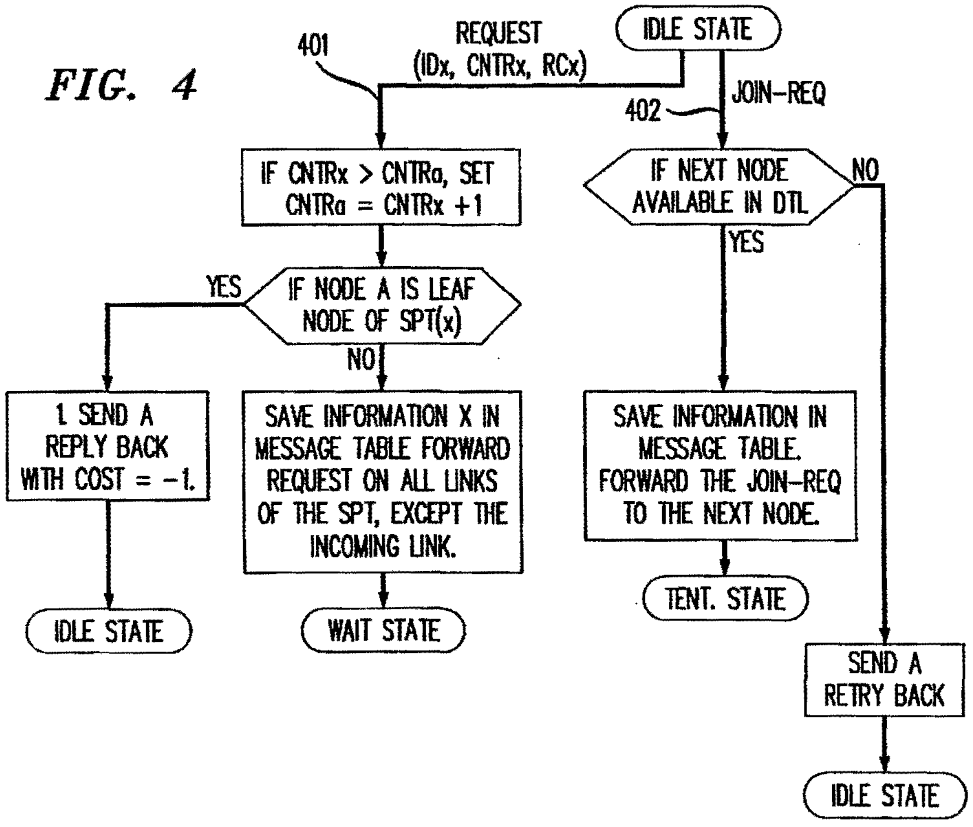

401
REQUEST (IDx, CNTRx, RCx)
IDLE STATE
JOIN-REQ
402
IF CNTRx > CNTRa, SET CNTRa = CNTRx +1
IF NEXT NODE AVAILABLE IN DTL
NO
YES
YES
IF NODE A IS LEAF NODE OF SPT(x)
NO
1. SEND A REPLY BACK WITH COST = -1.
SAVE INFORMATION X IN MESSAGE TABLE FORWARD REQUEST ON ALL LINKS OF THE SPT, EXCEPT THE INCOMING LINK.
SAVE INFORMATION IN MESSAGE TABLE. FORWARD THE JOIN-REQ TO THE NEXT NODE.
TENT. STATE
IDLE STATE
WAIT STATE
SEND A RETRY BACK
IDLE STATE

FIG. 5A

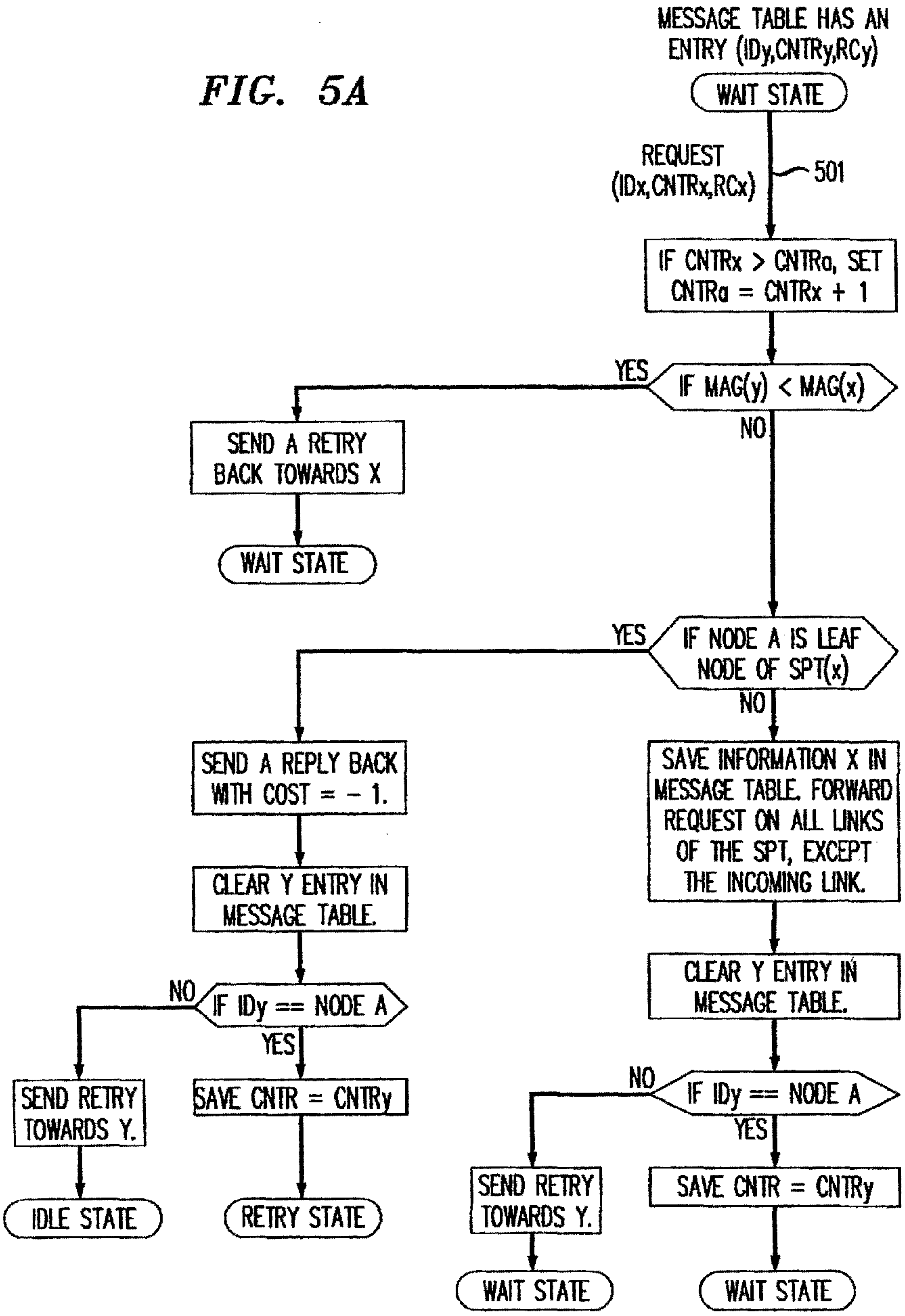

MESSAGE TABLE HAS AN ENTRY (IDy,CNTRy,RCy)
WAIT STATE
REQUEST (IDx,CNTRx,RCx)
501
IF CNTRx > CNTRa, SET CNTRa = CNTRx + 1
YES
IF MAG(y) < MAG(x)
NO
SEND A RETRY BACK TOWARDS X
WAIT STATE
YES
IF NODE A IS LEAF NODE OF SPT(x)
NO
SEND A REPLY BACK WITH COST = – 1.
SAVE INFORMATION X IN MESSAGE TABLE. FORWARD REQUEST ON ALL LINKS OF THE SPT, EXCEPT THE INCOMING LINK.
CLEAR Y ENTRY IN MESSAGE TABLE.
CLEAR Y ENTRY IN MESSAGE TABLE.
NO
IF IDy == NODE A
YES
SEND RETRY TOWARDS Y.
SAVE CNTR = CNTRy
NO
IF IDy == NODE A
YES
IDLE STATE
RETRY STATE
SEND RETRY TOWARDS Y.
SAVE CNTR = CNTRy
WAIT STATE
WAIT STATE

FIG. 5B

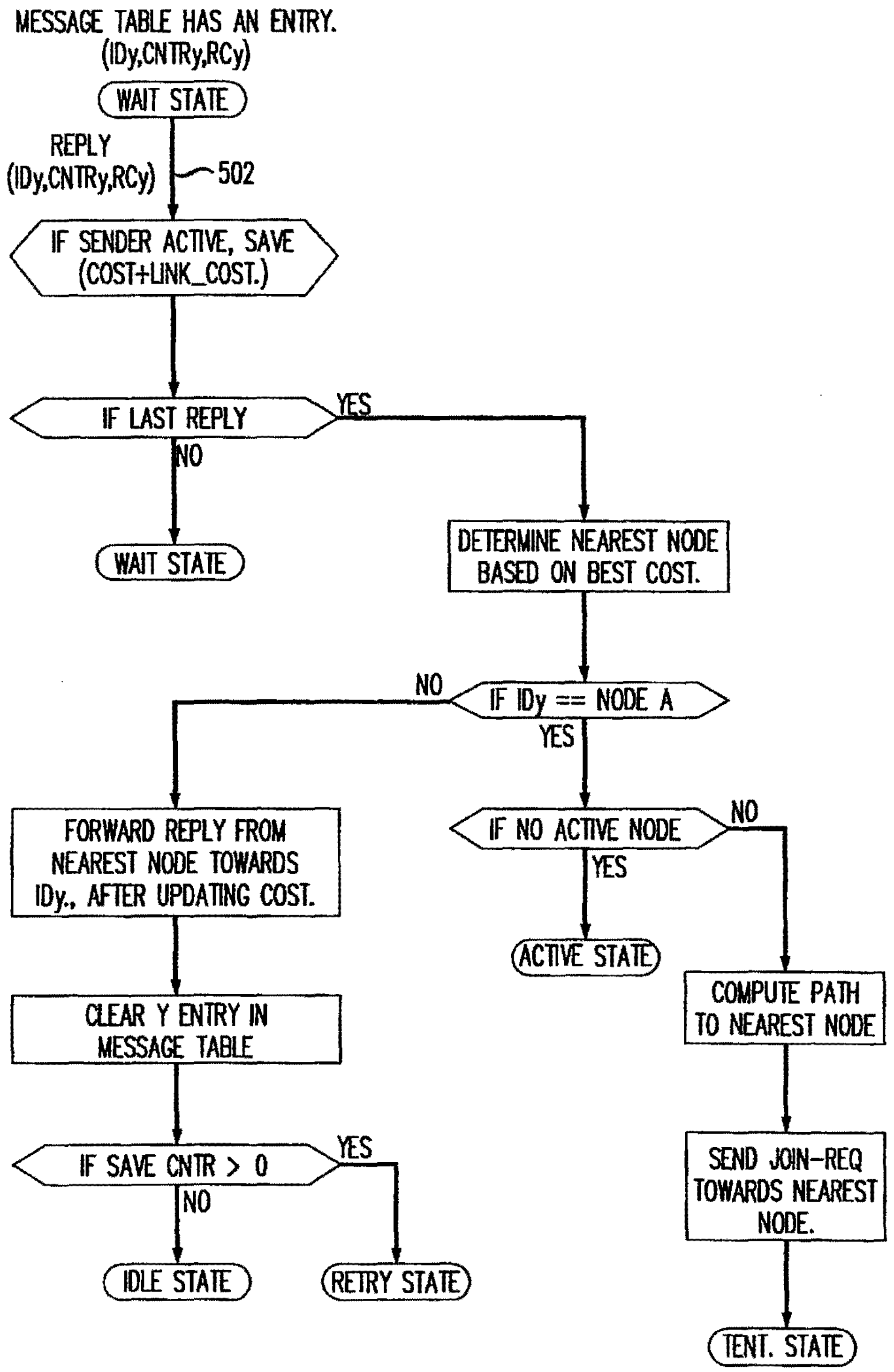

MESSAGE TABLE HAS AN ENTRY.
(IDy,CNTRy,RCy)
WAIT STATE
REPLY
(IDy,CNTRy,RCy)
502
IF SENDER ACTIVE, SAVE (COST+LINK_COST.)
IF LAST REPLY
YES
NO
WAIT STATE
DETERMINE NEAREST NODE BASED ON BEST COST.
NO
IF IDy == NODE A
YES
FORWARD REPLY FROM NEAREST NODE TOWARDS IDy., AFTER UPDATING COST.
IF NO ACTIVE NODE
NO
YES
ACTIVE STATE
COMPUTE PATH TO NEAREST NODE
CLEAR Y ENTRY IN MESSAGE TABLE
SEND JOIN-REQ TOWARDS NEAREST NODE.
IF SAVE CNTR > 0
YES
NO
IDLE STATE
RETRY STATE
TENT. STATE

FIG. 6

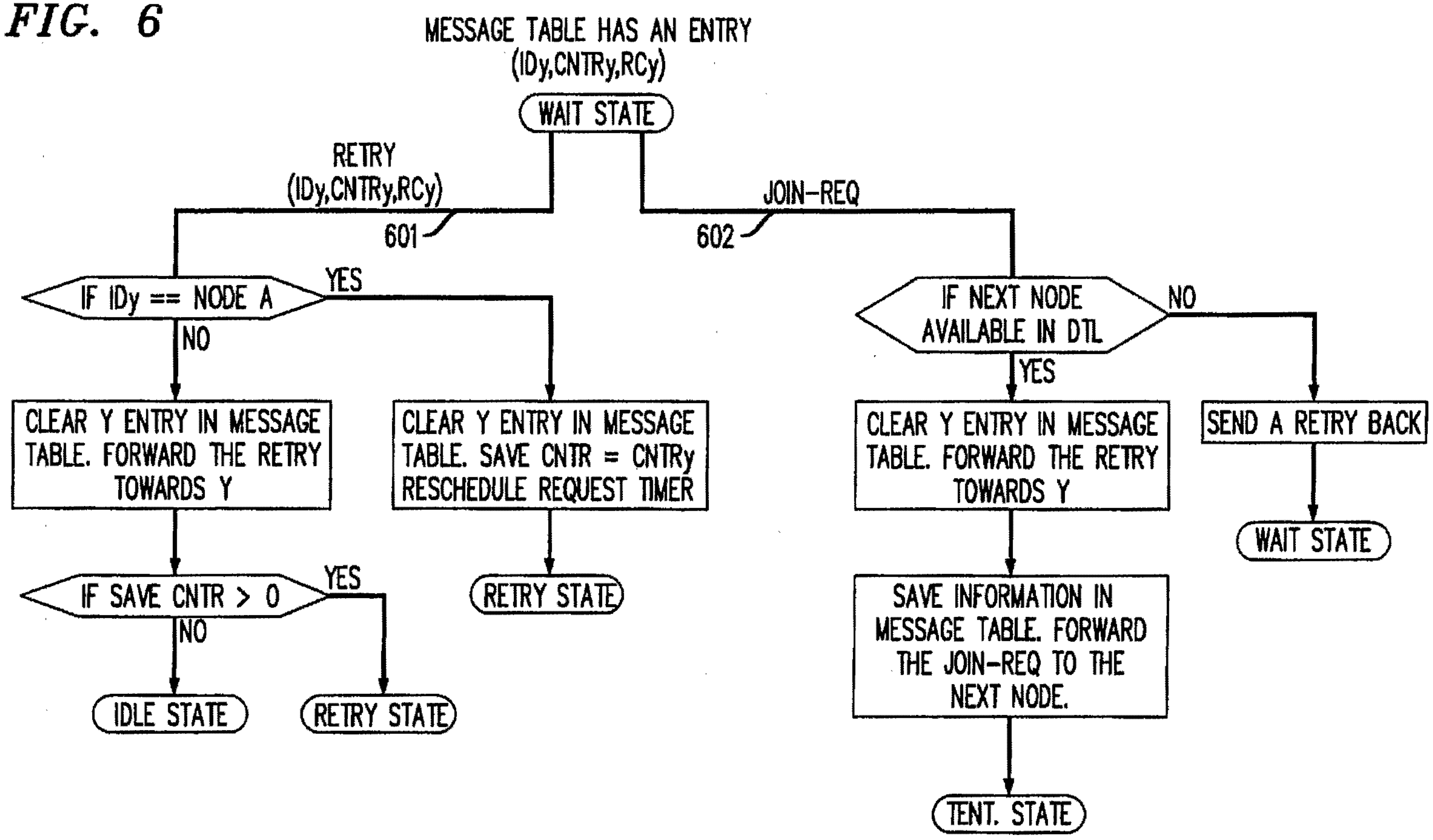

MESSAGE TABLE HAS AN ENTRY
(IDy,CNTRy,RCy)
WAIT STATE
RETRY
(IDy,CNTRy,RCy)
601
JOIN-REQ
602
IF IDy == NODE A
YES
NO
CLEAR Y ENTRY IN MESSAGE TABLE. FORWARD THE RETRY TOWARDS Y
CLEAR Y ENTRY IN MESSAGE TABLE. SAVE CNTR = CNTRy RESCHEDULE REQUEST TIMER
IF SAVE CNTR > 0
YES
NO
IDLE STATE
RETRY STATE
RETRY STATE
IF NEXT NODE AVAILABLE IN DTL
NO
YES
CLEAR Y ENTRY IN MESSAGE TABLE. FORWARD THE RETRY TOWARDS Y
SEND A RETRY BACK
WAIT STATE
SAVE INFORMATION IN MESSAGE TABLE. FORWARD THE JOIN-REQ TO THE NEXT NODE.
TENT. STATE

FIG. 7

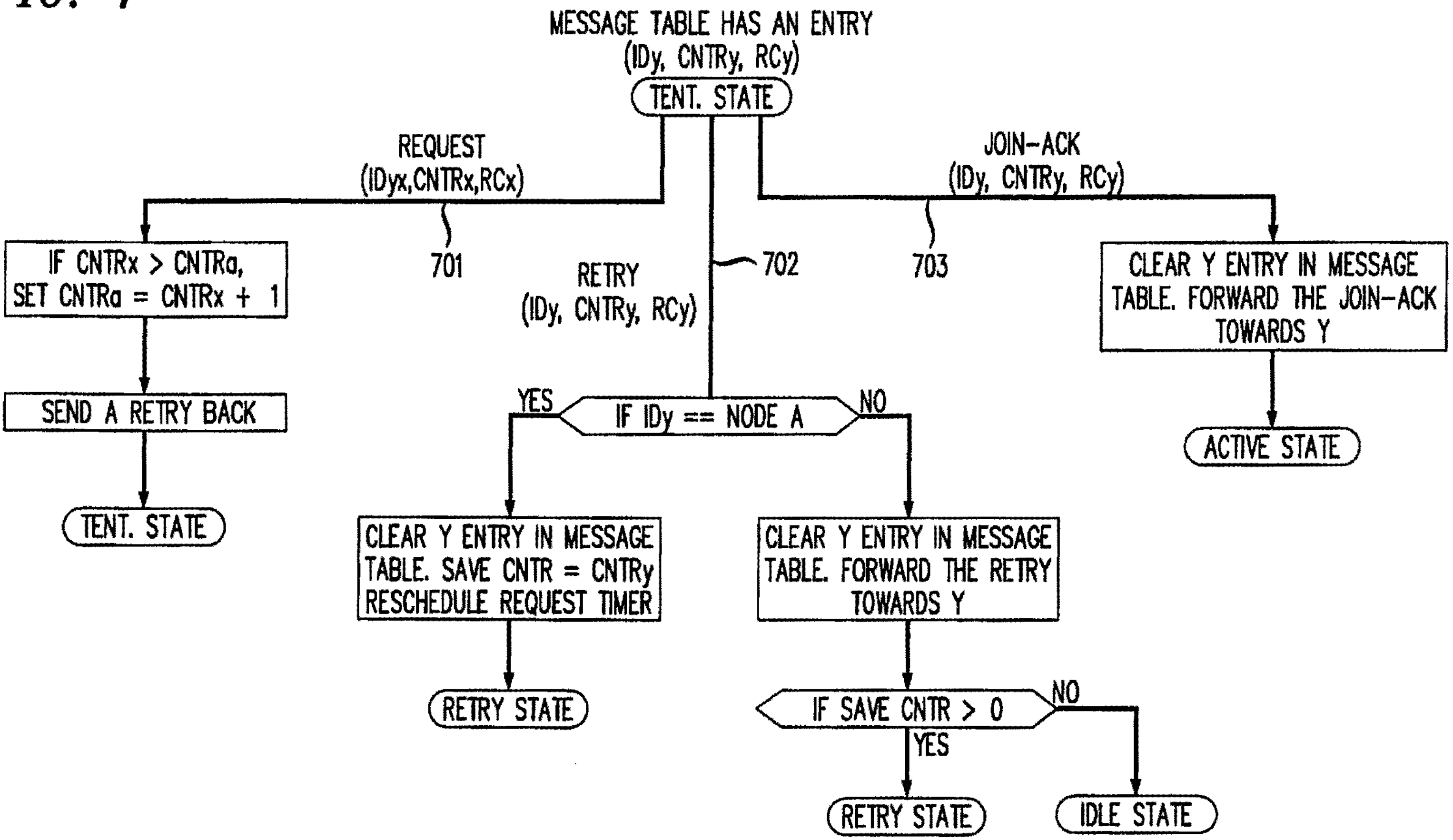

MESSAGE TABLE HAS AN ENTRY
(IDy, CNTRy, RCy)
TENT. STATE
REQUEST
(IDyx,CNTRx,RCx)
701
IF CNTRx > CNTRa,
SET CNTRa = CNTRx + 1
SEND A RETRY BACK
TENT. STATE
RETRY
(IDy, CNTRy, RCy)
702
YES
IF IDy == NODE A
NO
CLEAR Y ENTRY IN MESSAGE TABLE. SAVE CNTR = CNTRy RESCHEDULE REQUEST TIMER
RETRY STATE
CLEAR Y ENTRY IN MESSAGE TABLE. FORWARD THE RETRY TOWARDS Y
IF SAVE CNTR > 0
NO
YES
RETRY STATE
IDLE STATE
JOIN-ACK
(IDy, CNTRy, RCy)
703
CLEAR Y ENTRY IN MESSAGE TABLE. FORWARD THE JOIN-ACK TOWARDS Y
ACTIVE STATE

FIG. 8

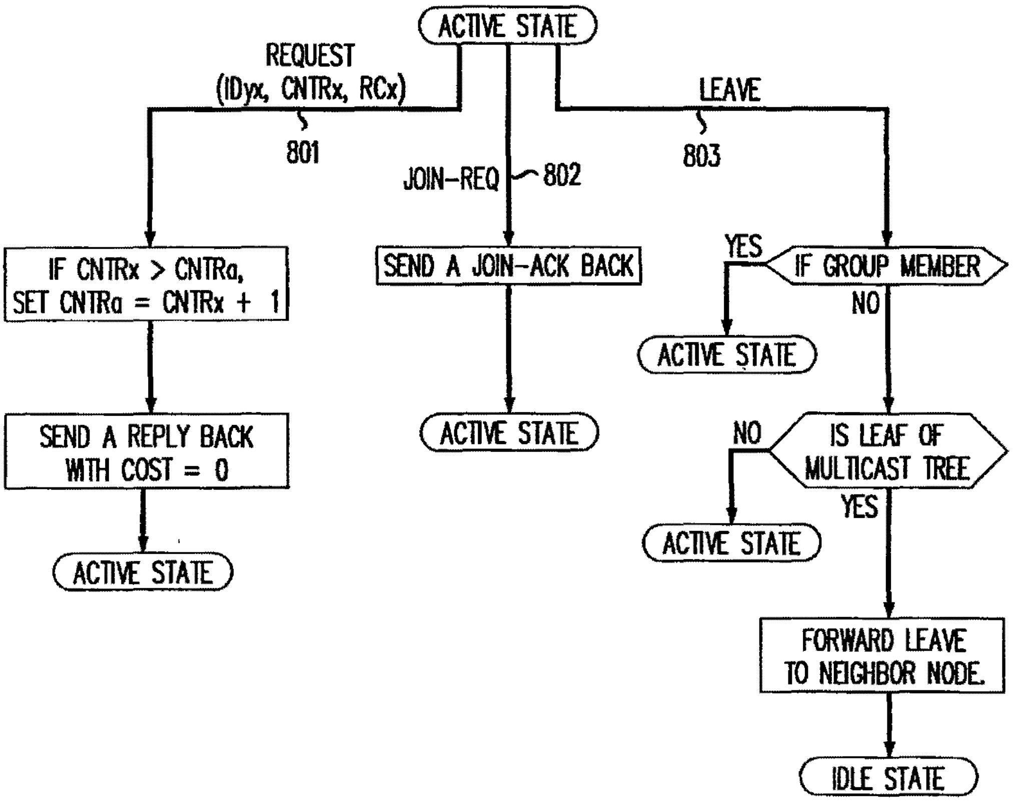

ACTIVE STATE
REQUEST (IDyx, CNTRx, RCx)
801
JOIN-REQ
802
LEAVE
803
IF CNTRx > CNTRa, SET CNTRa = CNTRx + 1
SEND A REPLY BACK WITH COST = 0
ACTIVE STATE
SEND A JOIN-ACK BACK
ACTIVE STATE
IF GROUP MEMBER
YES
ACTIVE STATE
NO
IS LEAF OF MULTICAST TREE
NO
ACTIVE STATE
YES
FORWARD LEAVE TO NEIGHBOR NODE.
IDLE STATE

FIG. 9

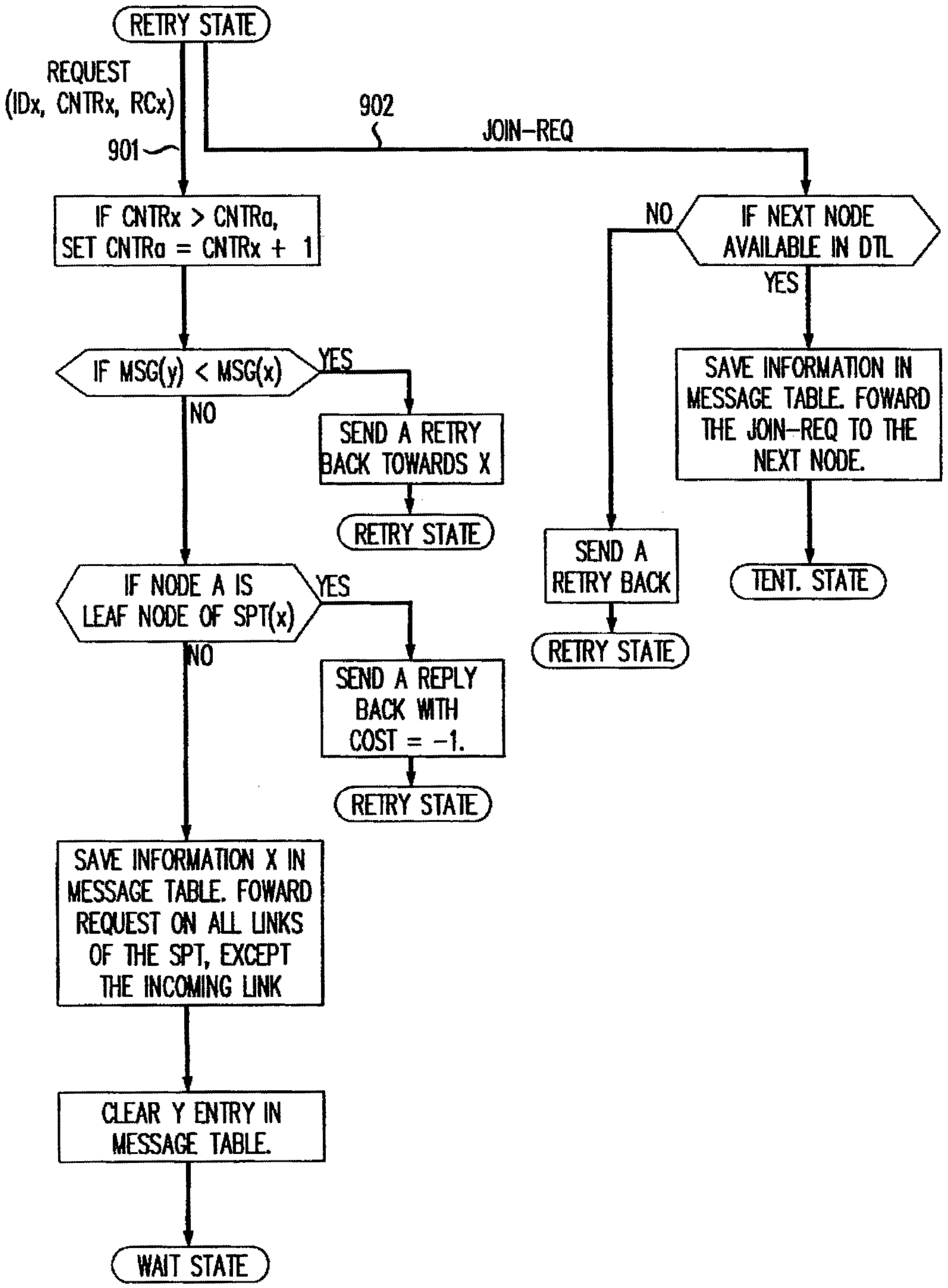

RETRY STATE
REQUEST (IDx, CNTRx, RCx)
901
902
JOIN-REQ
IF CNTRx > CNTRa, SET CNTRa = CNTRx + 1
IF MSG(y) < MSG(x)
YES
NO
SEND A RETRY BACK TOWARDS X
RETRY STATE
IF NODE A IS LEAF NODE OF SPT(x)
YES
NO
SEND A REPLY BACK WITH COST = -1.
RETRY STATE
SAVE INFORMATION X IN MESSAGE TABLE. FOWARD REQUEST ON ALL LINKS OF THE SPT, EXCEPT THE INCOMING LINK
CLEAR Y ENTRY IN MESSAGE TABLE.
WAIT STATE
IF NEXT NODE AVAILABLE IN DTL
NO
YES
SAVE INFORMATION IN MESSAGE TABLE. FOWARD THE JOIN-REQ TO THE NEXT NODE.
TENT. STATE
SEND A RETRY BACK
RETRY STATE

FIG. 10

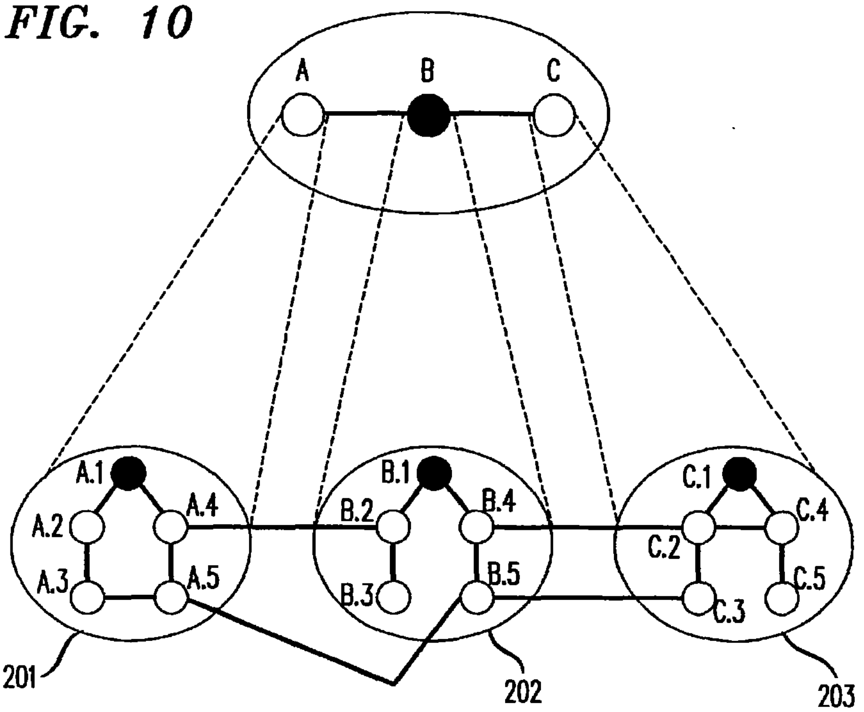

A
B
C
A.1
A.2
A.3
A.4
A.5
B.1
B.2
B.3
B.4
B.5
C.1
C.2
C.3
C.4
C.5
201
202
203

FIG. 11

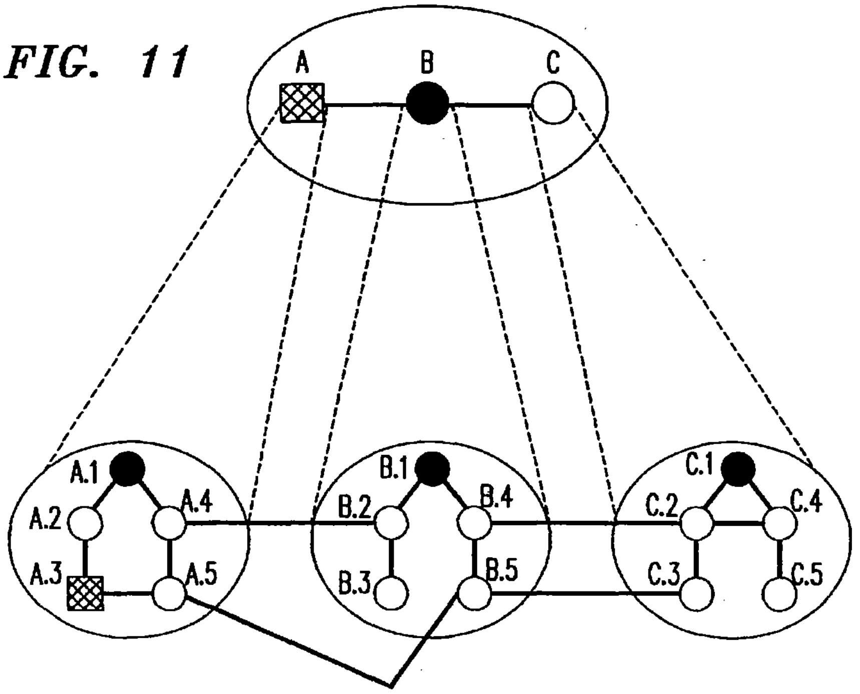

A
B
C
A.1
A.2
A.3
A.4
A.5
B.1
B.2
B.3
B.4
B.5
C.1
C.2
C.3
C.4
C.5

FIG. 12

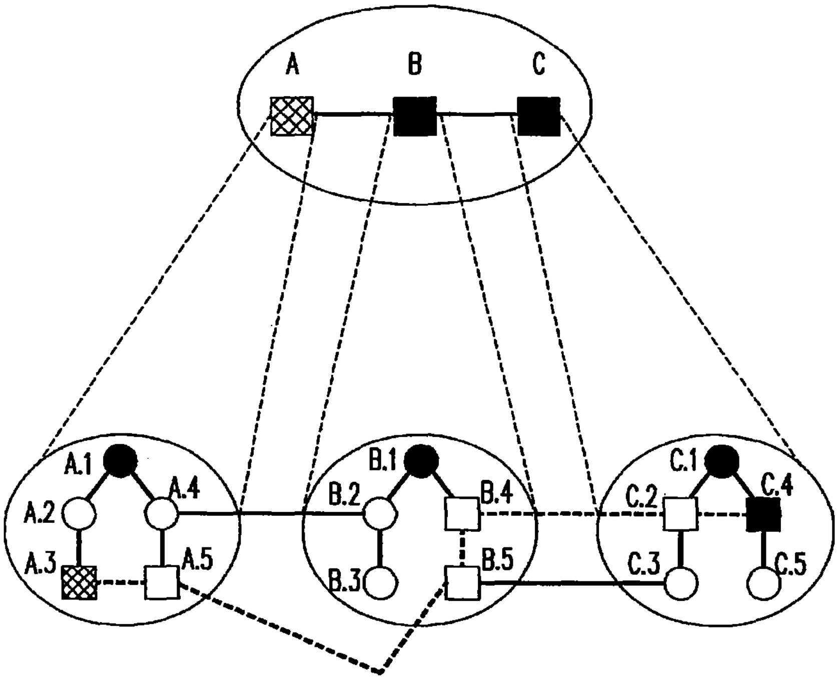

A
B
C
A.1
A.2
A.4
A.3
A.5
B.1
B.2
B.4
B.3
B.5
C.1
C.2
C.4
C.3
C.5

FIG. 13

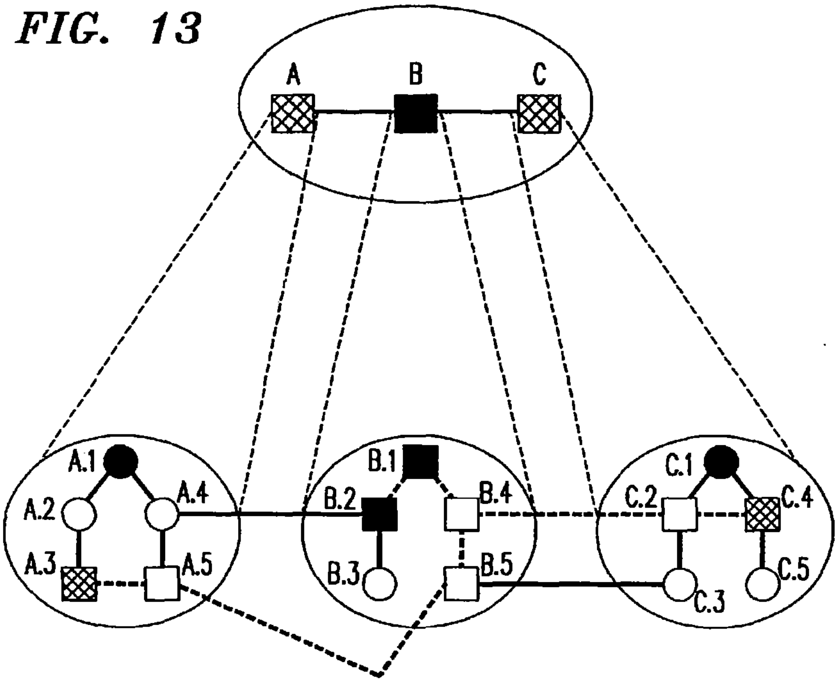

A
B
C
A.1
A.2
A.4
A.3
A.5
B.1
B.2
B.4
B.3
B.5
C.1
C.2
C.4
C.3
C.5

FIG. 14

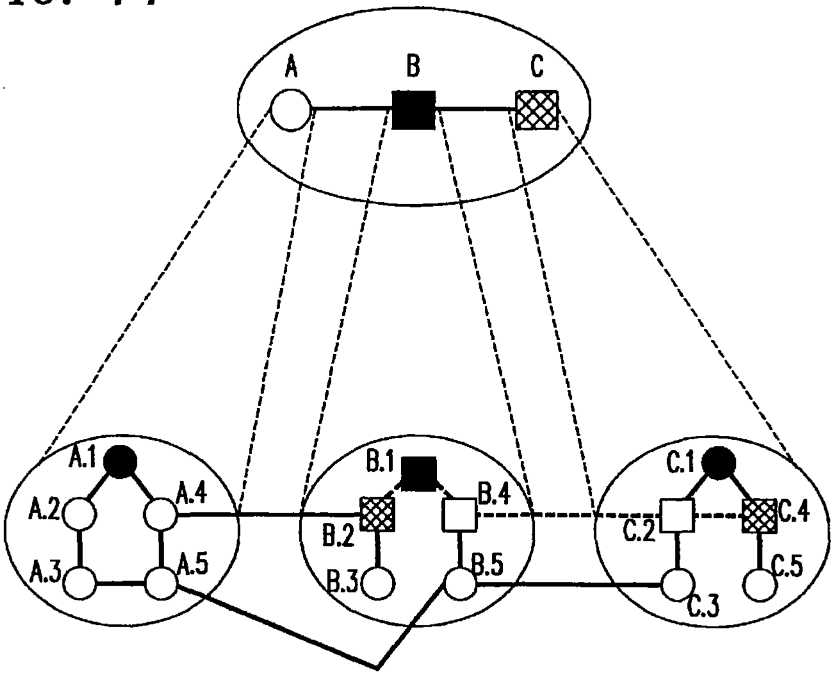
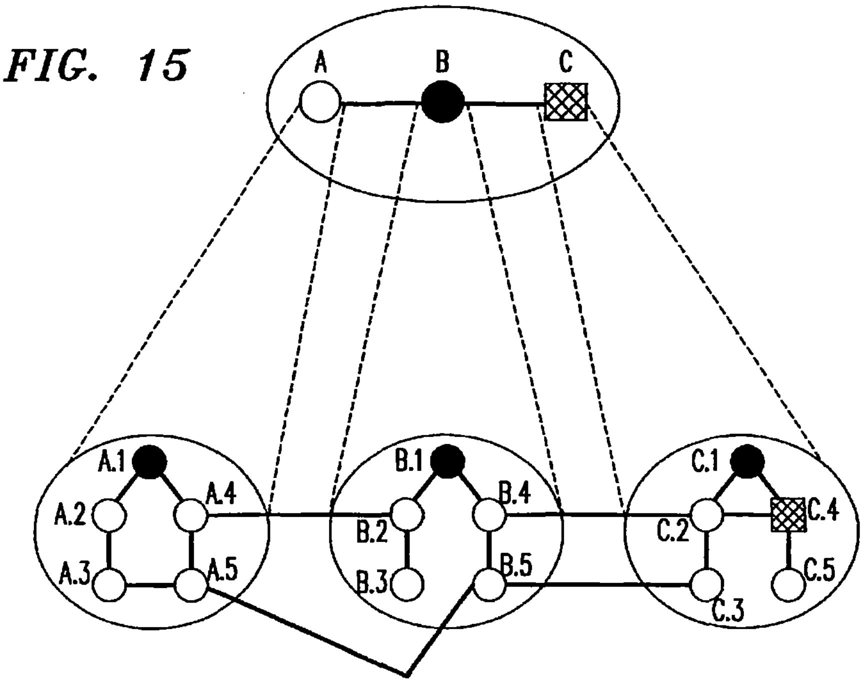

A
B
C
A.1
A.2
A.3
A.4
A.5
B.1
B.2
B.3
B.4
B.5
C.1
C.2
C.3
C.4
C.5

FIG. 15


A
B
C
A.1
A.2
A.3
A.4
A.5
B.1
B.2
B.3
B.4
B.5
C.1
C.2
C.3
C.4
C.5

DYNAMIC DISTRIBUTED MULTICAST ROUTING PROTOCOL

FIELD OF THE INVENTION

The invention relates to data networks and more particularly relates to a multicast protocol for locating, joining and leaving a multicast group that is receiving particular information.

BACKGROUND OF THE INVENTION

Presently, a user associated with a multimedia terminal, e.g., a conventional workstation or PC, may enter a request via the terminal to participate in a particular event, e.g., a lecture, audio/video conference, televised speech, etc., that is being provided by a node, e.g., a packet switch, in a digital network. The event is typically associated with some type of identifier, i.e., group identifier (e.g., 800 number or conference bridging number in conventional telephone networks) so that a user who wants to participate in the event may identify the event in a request that the user submits to his/her terminal. The user's terminal then forwards the request to an associated serving node within the digital network. A group identifier may represent a collection of users who are interested in particular information associated with the event. (Note that group identifier is different from an identifier that is used to identify a particular user.) Typically, the membership associated with a group identifier may be dynamic such that a member may join and leave the group at any time. Also, there is no restriction on the physical location of multicast group members who may or may not be at the same physical location. The network maintains information relating to the group and typically does this by constructing a network directory which is used to track the connectivity of active members. That is, a serving node submits the group identifier to the network directory. The directory, in turn, returns the address of the nearest node which it can join. The serving node then sends a request to join the multicast group to the identified node. A node that joins or leaves a multicast group must notify the directory so that it can update the membership information. Unfortunately, membership may change so frequently that management of such a directory may become costly and inefficient.

The network may also maintain such membership by constructing a multicast server in the network. Data that is sent by a user is then first delivered to the server, which, in turns, delivers the information to all other members in the multicast group. Although this approach appears to be simple to implement it, nevertheless, may lead to a single point of failure in the event that the server fails. Moreover, this approach does not use the network resources efficiently, since user data has to be sent to the server.

SUMMARY OF THE INVENTION

The foregoing is addressed and the relevant art is advanced by providing an efficient and scaleable mechanism for a data network to maintain multicast group information in a distributed fashion in which any node in the network may automatically locate, join and leave a multicast group. The mechanism is distributed so that a single node has to maintain complete information about the other participants of the multicast group. In particular, in accordance with an illustrative embodiment of the invention, a network node enters a request to join a multicast group, by constructing at least one routing tree formed from selected paths to each of a number of other nodes identified in the tree and sending a find message identifying the multicast group to the selected nodes in accordance with the routing tree. In response to receipt of a message identifying the nearest node connected to the multicast group, the requesting node then simply sends a join message to the identified node to join the multicast group.

These and other aspects of the claimed invention are disclosed in the ensuring detailed description, accompanying drawings and claims.

BRIEF DESCRIPTION OF THE DRAWINGS

In the drawing:

FIG. 1 shows a communications network in which the principles of the invention may be practiced;

FIG. 2 illustrates a routing tree rooted at a particular one of the nodes of FIG. 1;

FIG. 3 shows a state diagram illustrating the operation of a node in accordance with the principles of the invention;

FIGS. 4–9 are respective expanded versions of the operations states shown in FIG. 3, and

FIGS. 10–15 illustrate the operation of the principles of the invention in an illustrative hierarchical network.

DETAILED DESCRIPTION

Our inventive protocol supports what we call a "Participant-Initiated Join", in which a node initiates a request to join a multicast group and, in doing so, determines the network path that is to be used to join the multicast. The protocol has two phases. We call the first phase the "Find" phase, since it is the requesting node that determines the identity of the nodes that are already on the multicast tree. We call the second phase the "Join" phase, since the requesting node attempts to join the nearest node that is already on the multicast tree. In the Find phase, a node that wants to join the multicast group originates a REQUEST message and sends it to all of its neighbors on the shortest path tree rooted at the requesting node. A neighbor that receives the message then forwards it downstream to its neighbors along the links of the shortest path tree rooted at the originating node. This process continues until the REQUEST message reaches either a node that is already on the multicast tree or a leaf node of the shortest path tree rooted at the originating node (i.e., the node that has the sought-after multicast information.) Each node that is already on the multicast tree replies to the request, with the cost parameter set to **0**. The leaf nodes that are not on the multicast tree reply to the request with the cost parameter set to −1. On receiving all of the replies to the message, a node determines the nearest node and forwards the REPLY message from the nearest node upstream towards the originator of the REQUEST. Before doing so, the node updates the cost function to reflect the cost from this node to the nearest node. The originator receives all the replies and determines the nearest node. This ends the Find phase. In the Join phase, the requesting node determines the path to the nearest node and sends a JOIN-REQ message. The determined path is inserted in the message so that an intermediate node does not have to make the same determination. The nearest node (having the sought-after information) replies with a JOIN-ACK message and all the nodes in the path become a branch of the multicast tree.

An illustrative example of the foregoing is shown in FIG. **1**, in which sought-after multicast information is assumed to be a multimedia event **150**, e.g., a televised lecture. Multimedia signals characterizing the multimedia event are supplied to node **105**, which may operate as a conventional video server for the purpose of supplying the video signal in the form of digital signals to a user who has entered a request to receive a copy of such signals via network **100** formed from data nodes **105, 110, 115, 120, 125, 130** and **135**. In an illustrative embodiment of the invention, each of the nodes may be, for example, a conventional ATM switch. Assume that a user associated with workstation **170** in a conventional manner enters a request via a keyboard (not shown) to view the event on monitor **175**, in which the request includes an identifier associated with the multimedia event. The request, however, does not contain the identity (e.g., address) of the source of the multicast information. Workstation **170**, in response to receipt of the request, converts the request into a form expected by associated network node **125**, also shown as the E node. As an aspect of the invention, network **100** does not include a directory identifying the information that is respectively supplied by the network **100** nodes. Accordingly, to locate the node that has the multicast information, node **125** will poll each of the other network **100** nodes. Before doing so, however, node **125** constructs a tree formed from selected paths to each of the other nodes, in which the selection is based on predetermined parameter, for example, cost, number of hops, etc. In an illustrative embodiment of the invention, cost is characterized by a weight value. Such weight values are shown in parentheses in FIG. 1 and are used, as mentioned, to identify a least cost path/route. For example, if node **120** (D) needs to send a message to node **105** (A), then node **120** will likely send the message via node **110** (B) since the sum of the weights associated with the links in that path is less than the sum of the weights associated with the links in the via node **115** (C) path, i.e., (5)+(1)<(6)+(3).

Thus, in accordance with known techniques, node **125**, in the "find"phase, constructs a path tree and uses that tree to control the routing of a message to locate a node that has the requested multicast information, i.e., the televised event **150**. An example of such a tree is illustrated in FIG. 2. Node **120** thus sends a "Find" message to node **135** (G) and to node **120**(D) in accord with the tree. The "Find" message includes, inter alia, the address of the message originator, identifier associated with the sought-after information, a request for the identity of the node that has the sought-after information. The message may also include information characterizing the constructed path tree. Accordingly, then, upon receipt of the message, node **135** returns a Reply message with the cost parameter set to −1. Although node **120** does not have the sought-after information, it does not discard the message since the path tree indicates that the message should be individually routed to nodes **110, 115** and **130**, as shown in FIG. 2. Similarly, when the message reaches node **110**, the node does not discard the message (even though it does not have the sought-after information), but forwards it to node **105** (A). Since, it is assumed that node **105** has the sought after information, then it responds to the received message by returning to node **125** a reply message containing the identity (address) of node **105** and with the cost parameter set to 0.

Upon receipt of the latter message, node **125** enters the "Join" phase, as discussed above. In this phase, node **125** constructs the shortest path to node **105** and sends a Join message containing the node **125** address with a request to join the multicast of the identified event.

Referring to FIG. 2, it is seen that such path would be via nodes **110** and **120**, rather than via nodes **115** and **120**. Node **105** returns a Join-ACK and then starts supplying the televised event to node **125** via the determined path. When nodes **110** and **120**, which are in the path of the multicast, start receiving the multicast information then they note in their respective internal memories that they also have such information.

Assume at this point that a user associated with node **130** also wishes to receive the multicast and enters a request for that information. Similarly, node **130** constructs a least-cost path tree and launches its own find message in accordance with the tree (not shown). When the message reaches node **120**, it responds with a reply message to node **130** indicating that it has the information. As discussed above, node **130** accumulates the reply messages and then determines from those message which of the nodes having the sought-after information is closest to node **130**. In the present illustrative example, the closest node for the desired purpose would be node **120**. Accordingly, then node **130** may join the multicast by sending a Join message to node **120**, rather than to node **105**. Thus, when node **120** receives a packet of such information from node **110**, it sends a copy to node **125** and a copy to node **130**.

If at this point, the user at workstation **170** enters a request to terminate receipt of the multicast, then node **125** forms a "Leave" message and sends the message to node **120** in accordance with the constructed path tree rooted at node **125**. When node **120** receives the message, it stores the message in local memory and terminates the supplying of the multicast information to node **125** but will continue supplying that information as it is received to node **130**. If prior to the end of the televised event, node **130** launches a "Leave" message, then node **120**, in response to that message and in response to having no other receiver for the multicast, sends a message to node **110** to terminate the supplying of the multicast to node **120**. Similarly, node **110** sends a similar message to node **105** if it has no other receiver for the multicast information.

A situation could arise in which a node receives, at about the same time (concurrently), Find messages from a plurality of nodes, e.g., two nodes. To handle this situation and preserve the correctness of the distributed protocol, a node, e.g., node **125**, appends a time stamp to a Find message. In this way, if a node, e.g., node D, receives concurrently "Find" message from nodes **125** and **130**, then it processes the Find message bearing the earliest time stamp and returns a Reply message to the node, e.g., node **130**, that launched the Find message bearing the latest time stamp. Then, when the least cost multicast path is established from node **105** to node **120**, then node **130** launches a Find message after waiting a random (or a predetermined) period of time and then proceeds as discussed above.

In an illustrative embodiment of the invention, the principles of the invention are embodied in a state machine which is implemented in each network node for each multicast group. An illustrative example of such a state machine is illustrated in FIG. 3 and is composed of five major states, namely IDLE **301**, WAIT **302**, TENTATIVE **303**, ACTIVE **304** and RETRY **305**. Before discussing FIG. 3, it would be best to review the different message(s) that may be sent by a node attempting to join a multicast and the message(s) that are sent by the other nodes in the network in response to a request to join a multicast session. Briefly, a node that wants to join a multicast tree/session sends a REQUEST message, in the manner discussed above. The header of the request message contains, inter alia, the identity of the multicast information, sender ID, and time-stamp. (It may also contain a retry count.) A request message can only be originated by a node that is in the IDLE state. A node sends a REPLY message in response to receipt of a REQUEST message. A

REPLY message includes the header information of the REQUEST message and an associated cost parameter which identifies the cost of the shortest path from the current node to the nearest node that is already on the tree. The cost is updated as the REPLY moves from one node to another node. The cost is set to −1 if there is no ACTIVE node on that particular path. A node sends a RETRY message if it determines that another request is already being processed, as discussed above. The request with the earlier time-stamp is allowed to continue while a RETRY message is sent to the originator of the request having the later time stamp, as mentioned above. A node sends a JOIN-REQ message when it wants to join the multicast session/tree. The path to the nearest node is encoded within this message. An ACTIVE node sends a JOIN-ACK message in response to receiving a JOIN-REQ message. The receipt of this message indicates that a node is on the multicast tree. A node sends a LEAVE to an ACTIVE neighbor node when it wants to leave the multicast session/tree.

Turning then to FIG. 3, Briefly, in the IDLE state a node has no information pertaining to the multicast information (or tree). A node enters the WAIT state after it has sent/forwarded a REQUEST message and is waiting for a response. A node enters the TENTATIVE state after it has sent/forwarded a JOIN-REQ message towards the nearest node that is already on the multicast tree. In this state, the node is waiting for a JOIN-ACK message. A node enters the ACTIVE state when it joins a multicast session (i.e., it is on the multicast tree. A node reaches this state when it receives a JOIN-ACK message from a node that is already on the tree. A node in the ACTIVE state may also maintain the tree-based information regarding all the links that are incident on it and belong to the tree. However, the node does not maintain any state information about new requests to join the tree. A node reaches the RETRY state when it is the originator of a REQUEST to join a multicast tree and receives a RETRY message via one of the links of the tree. In this state, a node does not have to maintain any state information relating to other requests. Also, the node waits for a random period of time before re-sending the original REQUEST. It may also track the number of REQUEST messages that it sends.

An expanded version of the IDLE state is shown in FIG. 4. As mentioned above a node initiates a REQUEST message when it is in the IDLE state.

A node in this state only can initiate a REQUEST message. A REQUEST message is identified using the tuple<senderID,time-stamp,retry-count>, where senderID is the ID of the node that is the originator of the message, time-stamp is the value of a counter at the senderID and retry-count is the number of attempts the node has made to join the tree. The retry-count is initially 0. The initial REQUEST message is forwarded on all the links via the shortest-path tree rooted at the originating node. The node then enters the WAIT state. The following actions are taken when the node receives either a REQUEST message (action path **401**) or JOIN REQ message (action path **402**). For example, assume that a REQUEST message is received from node B, and therefore contains the tuple<ID_B,$CNTR_B$, RC_B>. Then the receiving node takes the following actions:

If the message is not on the shortest path, then send a REPLY with cost=−1 towards the originator. Else, update the time stamp (local counter).

Save the state information of the message.

Forward the REQUEST message. Change State=WAIT.

If no available link, send REPLY with cost=−1. State=IDLE.

If a JOIN-REQ message is received from node B, then the receiving node proceeds as follows:

Forward JOIN-REQ message to next node. Change State=TENTATIVE.

If current node is the final destination noted in the message, then return a RETRY message to the originator. (Note that this condition may arise if the node changes its state from ACTIVE to IDLE during the time between the sending of the REQUEST message and JOIN-REQ message to node B.)

An expanded version of the WAIT state is shown in FIGS. 5A, 5B and 6. For FIGS. 5A and B, assume that a node received a REQUEST message containing the tuple <ID_A, $CNTR_A$,RC_A>from node A.

If the node also receives a REQUEST message from node B, <ID_B,$CNTR_B$,RC_B>. Then the node takes the following actions (path **501**):

Update the local time stamp counter.

If the message had not been received via the shortest path, then send a REPLY to B, setting cost to −1.

If the message is received via the shortest path and A precedes B in time, then send a RETRY message to B.

If the message is received via the shortest path and B precedes A in time, then send a RETRY message to A. Clear state information pertaining to A. Save information pertaining to B and forward REQUEST message.

If the node receives a REPLY message via one of its links, then the node takes the following actions (path **502**):

Ignore the REPLY message if it does not contain state information.

If not the last reply, then store the cost information and wait for other reply messages.

If the last REPLY, then determine best message by minimizing the (cost field of message+link cost). Then forward the best message towards the originator of the REQUEST after updating cost.

Change state=IDLE.

If originator of REQUEST message, then compute the shortest path to the nearest node. Send JOIN-REQ message towards that node via the shortest path. Change state=TENTATIVE.

If the node receives a RETRY message via one of its links, then the node takes the following actions (path **601**, FIG. **6**):

Ignore the RETRY message if it does not contain state information.

Clear the state information and forward the RETRY message towards the originator of REQUEST.

Change state=IDLE.

If originator of REQUEST, then change state=RETRY. Increment retry count.

Wait for a random period of time, then resend the REQUEST message with the updated retry count value. (Note: The same counter value (CNTR) will be used to ensure fairness.)

If the node receives a JOIN-REQ message from a node, e.g., node B, then the node proceeds as follows (path **602**, FIG. **6**):

Change state=TENTATIVE.

If next node information is available, then forward the JOIN-REQ message to the next node and send a RETRY message to node A.

If no next node, then send a RETRY message towards the originator of the JOIN-REQ message.

An expanded version of the TENTATIVE state of FIG. **3**, is shown in FIG. **7**, in which a node enters the TENTATIVE

State after it receives a JOIN-REQ message originated by another node, e.g., node A. If at that time, the node receives a REQUEST message from node B containing the tuple<ID$_B$,CNTR$_B$, RC$_B$>, then the receiving node proceeds in the following manner (path **701**):

If a REQUEST message is not received via the shortest path, then return a REPLY with cost =–1 to the originator. Else,

Update local counter value.

Send RETRY message towards the originator of the message.

If, on the other hand, the node receives a RETRY Message, then the node takes the following actions (path **702**):

If RETRY matches JOIN-REQ, forward RETRY message back towards node A.

Change state=IDLE.

Ignore if RETRY message does not match JOIN-REQ.

If the node otherwise receives a JOIN-ACK Message, then the node takes the following actions (path **703**):

Change State=ACTIVE.

Forward Join-Ack to node A.

Clear all state information.

Update multicast tree information to include the link over which the JOIN-REQ was received.

An expanded version of the ACTIVE state of FIG. **3**, is shown in FIG. **8**, in which a node is already on the multicast tree. In that case, the node only responds to new requests and join requests. Other messages may be ignored.

If a node in the active state receives a REQUEST message from node B a containing the tuple<ID$_B$,CNTR$_B$, RC$_B$>, then the receiving node takes the following action (path **801**):

If the REQUEST message is not received via the shortest path,

then send REPLY message with cost=–1 to the originator. Else,

Update local counter value.

Send a REPLY with cost=0.

If, on the other hand, the node receives a JOIN-REQ message from node B, then the node returns a JOIN-ACK message to B (path **802**). If the node otherwise receives a LEAVE message from node B, then the node takes the following actions:

Update multicast tree information.

If only one ACTIVE neighbor and the node is not member of multicast group, then send LEAVE message to neighbor.

Change state=IDLE.

An expanded version of the RETRY State is shown in FIG. 9. A node enters the RETRY state after sending a REQUEST message. Assume that node A is in the retry state and the associated tuple information is—<ID$_A$,CNTR$_A$, RC$_A$>, and that node A receives a REQUEST message from node B whose tuple information is<ID$_B$,CNTR$_B$,RC$_B$>. For that case, node A takes the following actions (path **901**):

If message is not on the shortest path, send a REPLY back with cost set to –1. Update local counter.

If A precedes B in time, then send a RETRY message towards node B.

If B precedes A in time, then save B's state information and forward the REQUEST message. Save A's information and stop A's retry timer. (A cannot join the multicast group until B has joined).

Change state=WAIT. (A is now waiting for B)

If, on the other hand, a JOIN-REQ message is received from node B, then the node takes the following actions (path **902**):

Change state=TENTAITVE. Forward JOIN-REQ message to next node on the path.

Save A's information and wait. (A cannot resend REQUEST message until B has joined multicast group).

If Retry timer has expired, then increment Retry count and send REQUEST message containing the following tuple; <ID$_A$,CNTR$_A$,RC$_A$>.

The old value of CNTR$_A$ is used to ensure fairness.

The foregoing may be readily implemented in a network formed from a plurality of so-called Peer groups, in which a peer group is a group of network nodes forming a smaller network. In particular, a large network formed from a large number of nodes may be logically restructured to improve the efficiency of the number. Such restructuring entails forming nodes into groups each operating as a smaller network, in which connection management functions are logically extended to each level of the network hierarchy. The state information is maintained at each physical node. In addition, each group is represented as a logical node at a higher level of the network hierarchy by a peer group leader (PGL). The PGL maintains separate state information for that logical node at that hierarchical level. The logical nodes perform the same state transitions as the physical nodes at the lowest level of the network hierarchy. In that sense, our inventive protocol may be applied to such a network with the following modifications (extensions).

Find Phase

i) A Joining Node prompts (requests) its PGL to enter the "Find" phase at the next higher level of the hierarchy. At the logical level, pre-established virtual connections between logical nodes are used to send the REQUEST and REPLY messages. During the "Find" phase, the logical nodes may make a transition from the IDLE state to the WAIT state and RETRY state similar to the physical nodes.

ii) If the PGL is already active at the higher level (which may mean that some other ACTIVE node exists in the peer-group), then this information is returned to the requesting node. On receipt of the information, the requesting node executes the protocol to determine the nearest ACTIVE node within the peer-group. The ID of the nearest ACTIVE node (physical or logical) is returned to the lower level requesting node. If the current level is the lowest level, then the requesting node enters the "Join" phase.

iii) Otherwise, the PGL recursively executes the above steps at higher levels until it reaches the top level of the hierarchy. At that level, the PGL enters the "Find" phase. At the end of the "Find" phase, the requesting node has the ID of the nearest ACTIVE node. (Note that this ID may either be the physical ID of the node within the same peer-group (single peer group case) or the logical ID of a higher level node in a logical peer-group (multiple peer group case).

Join Phase

Based on the available topological information, a requesting node determines the path to the nearest node. The path may not be complete if the nearest node is a logical node in a higher level, logical peer-group. The determined path is stored in the form of a Designated Transit List (DTL), as discussed in the Private Network-Network Interface (PNNI) specification version 1.0, available from the ATM Forum, 2570 West El Camino Real, Suite **304**, Mountain View, Calif. 94040-1313, which is hereby incorporated by reference. The DTL is embedded in the JOIN-REQ message before it is forwarded to the nearest node. On receiving a JOIN-REQ message, a node executes the following protocol:

1) If the node is already ACTIVE, it returns a JOIN-ACK message. The JOIN-ACK is returned along the same path (in the reverse direction) to originating node.

ii) Else, if the DTL stack is not empty, then the JOIN-REQ message is forwarded to the next node based on the information available in the DTL. (Note that new paths may be added to the DTL by ingress nodes, as explained in the above-mentioned reference. Also note that the JOIN-REQ and JOIN-ACK messages may be sent on network signaling channel. When the JOIN-REQ or JOIN-ACK messages are sent across border links, the logical node at the appropriate level has to be informed so that the logical node can make a state transition to the TENTATIVE or ACTIVE state respectively. This is done by sending a copy of the JOIN-REQ or JOIN-ACK message to the appropriate logical node.

iii) If the DTL stack is empty, then the node enters the "Find" phase to determine the nearest ACTIVE node. The nearest node information is then encoded as DTL and the message is forwarded to the nearest node.

Similar actions are taken when a participating node leaves the multicast group. In this case a LEAVE message is sent to an ACTIVE neighbor node, provided that the participant has only one ACTIVE neighbor. The participant node then enters the IDLE state. The neighbor node may then forward the LEAVE message to its nearest neighbor if it is not a participant and has only one ACTIVE neighbor. When a LEAVE message is sent across a border link, then a copy of the message is sent to the appropriate logical node, so that the global state of the node can be changed to the IDLE state.

The extended protocol described immediately above may be further appreciated when discussed in conjunction with the examples illustrated in FIGS. **10** through **13**, in which each FIG. illustrates a network formed from three peer groups of network nodes (e.g., ATM switches) **201**, **202** and **203**. The nodes in peer group (a) **201** are respectively designated A.1 through A.5, (b) **202** are respectively designated B.1 through B.5 and (c) **203** are respectively designated C.1 through C.5. The dark (or filled in) node serves as the PGL for the respective peer group in each of FIGS. **10** through **13**. Assume for FIG. **10** that node A.3 wants to join a multicast group, and, therefore sends a request to that effect to its PGL (node A), which causes that PGL to enter the "Find" phase at the higher level as noted by the ellipse containing logical PGLs A, B and C. Since PGL A enters the highest level, it sends a REQUEST message to the other members of its peer-group to determine who has the requested multicast information. Assume that PGL A determines that there are no members of the multicast in peer-groups B and C. This information is passed on to node A.3 and the state of node A.3 becomes ACTIVE. The multicast tree now consists of a single node A.3. Globally, the state machine for logical node A also makes a transition to the ACTIVE state. The ACTIVE nodes at each level are shown as boxes in FIG. **11**. The participant nodes at each level are shown as lightly shaded boxes.

Assume that node C.4 now wants to join the multicast group, and, therefore sends a request to its PGL (node C) to enter the "Find" phase. Node C, in response thereto, sends a REQUEST message to logical node B and logical node A. (Note that node C may enter the RETRY state if there is a concurrent Join request from logical node B.) If there are no other requests, then only logical node A is ACTIVE and this information is sent to node C.4 by logical node C. This action completes the "Find" phase for node C.4.

In the "Join" phase, node C.4 encodes a path to peer-group A as a DTL and forwards a JOIN-REQ message to peer-group A. The message enters peer-group B, across the border link (C.2—B.4). Here, node B.4 is the ingress node and the DTL stack is not empty (step (ii) of the "Join" phase). Node B.4 identifies a path through its peer-group to peer-group A and appends the path to the DTL. It then forwards the message towards peer-group A. Node B.4 also sends a copy of the JOIN-REQ message to the PGL node B.1, which maintains the global state machine for logical node B. Node B.1 updates the global state machine of node B to TENTATIVE state. Assume that the message enters peer-group A across link (B.5—A.5). The DTL stack is now empty (step (iii) of the "Join" phase). Node A.5 upon receipt of the message enters the "Find" phase to determine the nearest node that should receive the message, i.e., node A.3. It then appends the path to node A.3 to the DTL and forwards the JOIN-REQ to node A.3. Node A.3, being in the ACTIVE state, replies to receipt of the message by returning a JOIN-ACK message (step (i) of the "Join" phase). The JOIN-ACK messages retraces the path followed by the JOIN-REQ message. When the JOIN-ACK crosses the border link (A.5—B.5), the state machine of logical node B is updated to the ACTIVE state. Logical node C becomes ACTIVE when the JOIN-ACK message is sent across link (B.4—C.2). The resulting multicast tree is shown as dotted lines in FIG. **12**. Node B.2 can now join the multicast tree by joining node B.4, as shown in FIG. **13**.

If participant A.3 leaves the multicast group, then it sends a LEAVE message to node A.5, which propagates the message to node B.5. Node A.5 also sends a copy of the LEAVE message to node A.1, so that the global state machine of logical peer-group A may be changed to the IDLE state. Node B.5 also sends a LEAVE message to node B.4. However, the propagation of the latter message stops at node B.4 since that node has two ACTIVE neighbors, nodes B.1 and C.2. The resulting tree is shown in FIG. **14**. Assume now node B.2 similarly leaves the multicast group. In this instant the LEAVE message propagates to node C.4. Recall that when the LEAVE message is sent across link B.4—C.2, a copy is also sent to node B.1 so that the global state machine for the peer-group may be changed to IDLE state. The final tree consisting of only node C.4 is shown in FIG. **15**.

The foregoing is merely illustrative of the principles of the invention. Those skilled in the art will be able to devise numerous arrangements, which, although not explicitly shown or described herein, nevertheless embody those principles that are within the spirit and scope of the invention.

We claim:

1. A method of joining a multicast connection originating from a source node distributing multicast information, said multicast connection being established in a network composed of a plurality of communications nodes, said method comprising the steps of:

at one of said nodes, responsive to receipt of a request to receive said multicast information, constructing at least one routing tree formed from selected paths to each of a number of other nodes identified in the tree and sending a message identifying said multicast information to the selected nodes in accordance with the routing tree, and

at said one node, responsive to receipt of a message identifying said source node as being the originator of the identified multicast information, sending a message containing a request to join multicast distribution of the identified multicast information.

2. The method of claim 1 further comprising the steps of

at the source node, responsive to receipt of a message containing a request to join the multicast distribution of the identified information, returning an acknowledgment message to the originator of the message.

3. The method of claim 1 further comprising the steps of

at the source node, responsive to receipt of a message containing a request to join the multicast distribution of the identified information, constructing a network path tree rooted at the source node and extending to the node originating the message, and

supplying the multicast information via the path tree rooted at the source node.

4. The method of claim 3 wherein said multicast tree includes at one intermediate node between the source node and the node originating the message and wherein said method further comprises the steps of

at the intermediate node, responsive to receiving from another one said nodes a message containing a request for said multicast information, returning to said other one of said nodes a reply message indicating that said intermediate node can supply said multicast information, and

at said other one of said nodes accumulating all reply messages to its request for the multicast information and determining which of nodes returning a reply message is closest to said other one of said nodes, and

sending a message to join the multicast to the determined closest one of the nodes that returned a reply message to said other one of said nodes.

5. The method of claim 4 further comprising the step of

at the closest one of the nodes, responsive to receipt of the join message and thereafter responsive to receipt of multicast information via the path tree rooted at the source node, sending a copy of the received multicast information to the other one of said nodes.

6. The method of claim 5 wherein said intermediate node is said closest one of the nodes and wherein said method further comprises the steps of

at said intermediate node, responsive to receiving from said one node a message addressed to said source node and requesting a termination of the sending of the multicast information to said one node, terminating the sending of the multicast information to said one node and continuing to supply the multicast information as it is received to said other one of said nodes.

7. The method of claim 5 wherein said intermediate node is said closest one of the nodes and wherein said method further comprises the steps of

at the intermediate node, responsive to receiving from said other one of said nodes a message addressed to said source node and requesting a termination of the sending of the multicast information to said other one of said nodes, terminating the sending of the multicast information to said other one of said nodes, and responsive to having no other recipient for said multicast information, sending the received termination message to the source node.

8. A method of joining a multicast connection originating from a source node distributing multicast information, said multicast connection being established in a network composed of a plurality of communications nodes, said method comprising the steps of

at a network node, responsive to receiving from first and second other ones of the network nodes respective message requests to join the multicast, determining as a function of a predetermined parameter a first one and a second one of the first and second nodes, said first one is to be connected first to the multicast connection and sending a wait message to said second one,

forwarding the message received from said first one over a path contained in that message, and

thereafter, responsive to receipt of multicast information from a source via a path rooted at the source, supplying the multicast information to said first one.

9. The method of claim 8 further comprising the steps of

at said other one of said first and second nodes, response to receipt of the wait message, waiting a predetermined period of time, and

at the expiration of that period of time, again sending a message requesting receipt of the multicast information.

10. The method of claim 9 further comprising the step of

at said network node, responsive to receipt of the request message from said other one of said first and second nodes, supplying said multicast information to said other one of said first and second nodes as it is received from the source.

11. The method of claim 9 further comprising the steps of

at said network node, responsive to receiving from said one node of said first and second modes a message addressed to said source node and requesting a termination of the sending of the multicast information to said one of said first and second nodes, terminating the sending of the multicast information to that node and continuing to supply the multicast information as it is received to said other one of said first and second nodes.

12. The method of claim 9 further comprising the steps of

at said network node, responsive to receiving from said other one of said first and second nodes a message addressed to said source node and requesting a termination of the sending of the multicast information to said other one of said first and second nodes, terminating the sending of the multicast information to said that node, and responsive to having no other recipient for said multicast information, sending the received message requesting termination to the source node.

* * * * *

US006252884B1

(12) **United States Patent**
Hunter

(10) **Patent No.: US 6,252,884 B1**
(45) **Date of Patent: Jun. 26, 2001**

(54) **DYNAMIC CONFIGURATION OF WIRELESS NETWORKS**

(75) Inventor: **Wesley G. Hunter**, Doraville, GA (US)

(73) Assignee: **NCR Corporation**, Dayton, OH (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/045,263**

(22) Filed: **Mar. 20, 1998**

(51) **Int. Cl.**^{7} **H04B 7/212**

(52) **U.S. Cl.** **370/443**; 370/552

(58) **Field of Search** 370/552, 443, 370/400, 410, 426, 522, 377, 373, 374, 378, 384, 329, 348, 349, 335, 395, 232, 230, 524, 401, 338; 455/522

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,088,094	*	2/1992	Grauel et al.	370/443
5,732,360	*	3/1998	Jarett et al.	455/552
5,737,319	*	4/1998	Croslin et al.	370/255
6,034,966	*	3/2000	Ota	370/443
6,049,535	*	4/2000	Ozukturk et al.	370/335

* cited by examiner

Primary Examiner—Wellington Chin
Assistant Examiner—Premell Jones

(74) *Attorney, Agent, or Firm*—Needle & Rosenberg, P.C.

(57) **ABSTRACT**

The present invention is directed to a system and process for automatically and dynamically configuring a wireless computer network. Each computer that is to participate in the dynamic network continuously broadcasts its address to any other computer within range of the wireless network hardware. When a computer receives a broadcast message from a machine it is not currently connected to, it can then use any standard communications protocol (i.e., TCP/IP) to establish a connection to the broadcasting machine. Once the connection is established, a message is sent to the broadcasting machine notifying it of the new connection. This allows for either client/server, peer-to-peer, or other communications strategies to be implemented, depending on the application. Upon establishing a new connection between a pair of computers, a data synchronization protocol is employed to exchange data, applications, or configure services. To avoid having many disconnects, reconnects, and data synchronizations happening, a connection degradation strategy is used.

14 Claims, 5 Drawing Sheets

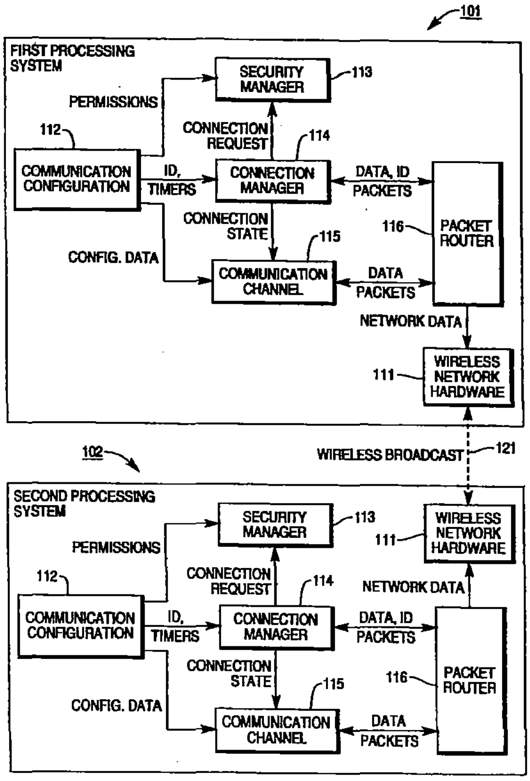


FIG. 1

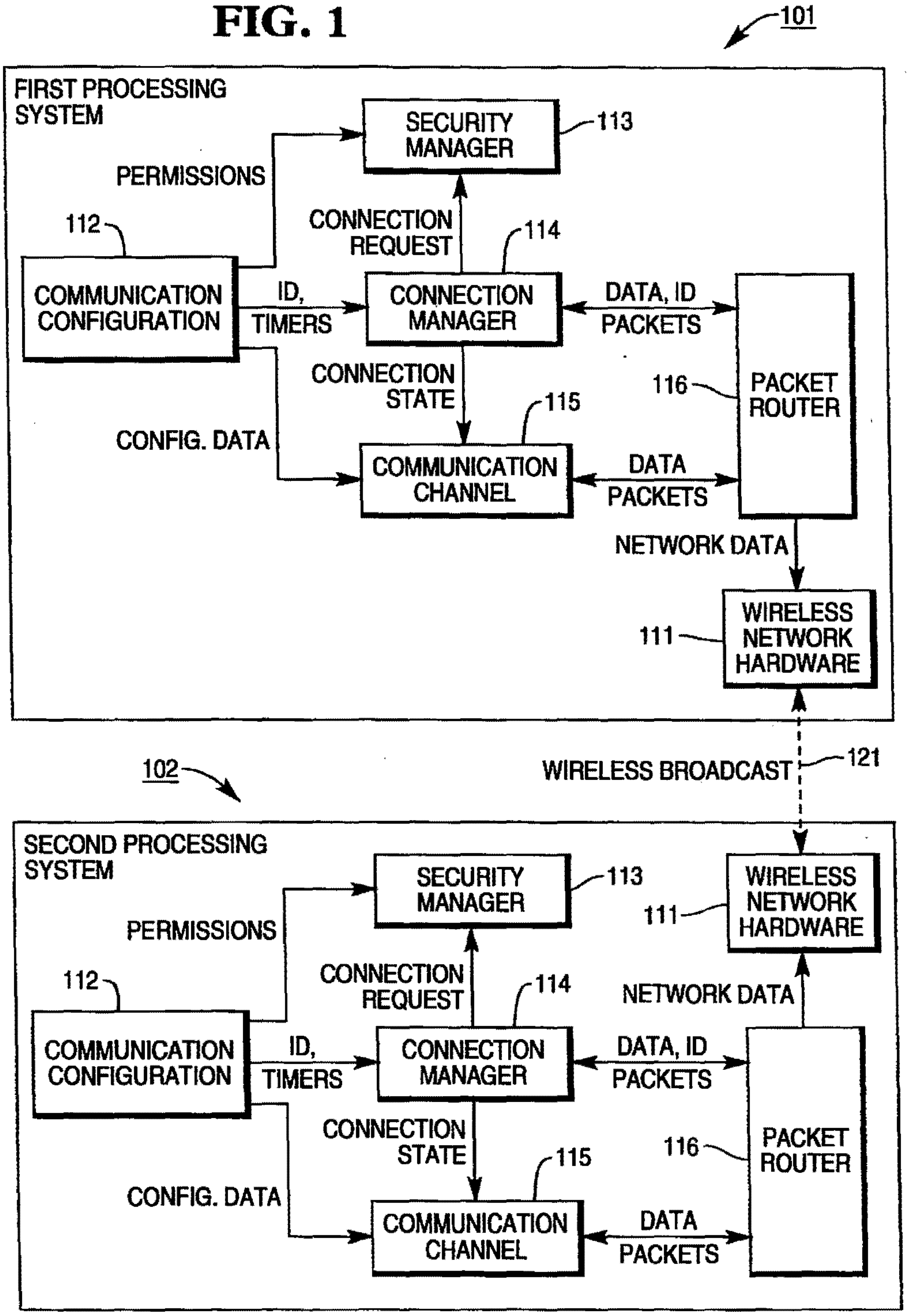
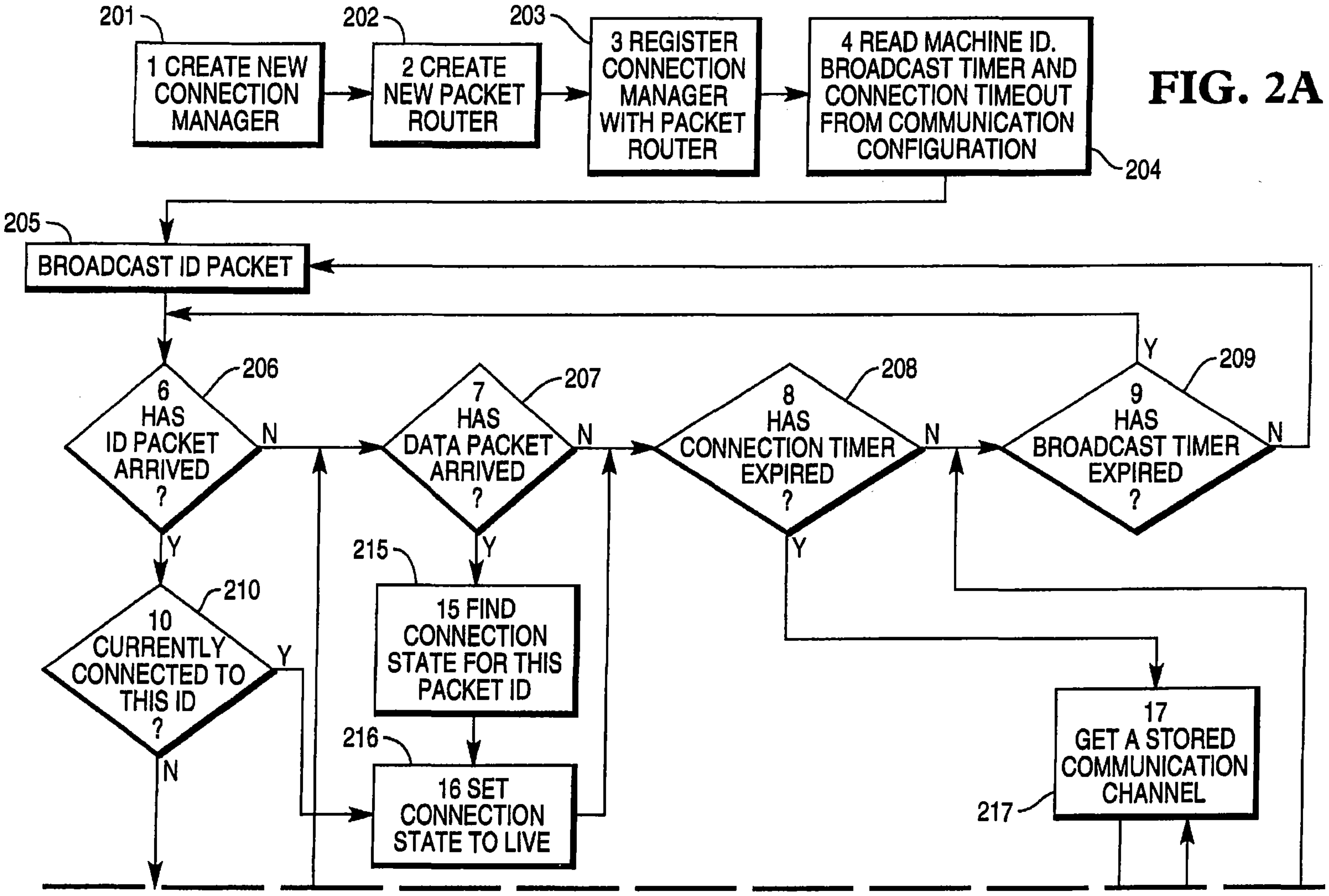

101
FIRST PROCESSING SYSTEM
SECURITY MANAGER
113
PERMISSIONS
CONNECTION REQUEST
114
112
COMMUNICATION CONFIGURATION
ID, TIMERS
CONNECTION MANAGER
DATA, ID PACKETS
CONNECTION STATE
116
PACKET ROUTER
115
CONFIG. DATA
COMMUNICATION CHANNEL
DATA PACKETS
NETWORK DATA
111
WIRELESS NETWORK HARDWARE
102
WIRELESS BROADCAST
121
SECOND PROCESSING SYSTEM
SECURITY MANAGER
113
WIRELESS NETWORK HARDWARE
111
PERMISSIONS
CONNECTION REQUEST
114
NETWORK DATA
112
COMMUNICATION CONFIGURATION
ID, TIMERS
CONNECTION MANAGER
DATA, ID PACKETS
CONNECTION STATE
116
PACKET ROUTER
115
CONFIG. DATA
COMMUNICATION CHANNEL
DATA PACKETS

FIG. 2A


201
1 CREATE NEW CONNECTION MANAGER
202
2 CREATE NEW PACKET ROUTER
203
3 REGISTER CONNECTION MANAGER WITH PACKET ROUTER
4 READ MACHINE ID. BROADCAST TIMER AND CONNECTION TIMEOUT FROM COMMUNICATION CONFIGURATION
204
205
BROADCAST ID PACKET
206
6 HAS ID PACKET ARRIVED ?
N
Y
207
7 HAS DATA PACKET ARRIVED ?
N
Y
208
8 HAS CONNECTION TIMER EXPIRED ?
N
Y
209
9 HAS BROADCAST TIMER EXPIRED ?
Y
N
210
10 CURRENTLY CONNECTED TO THIS ID ?
Y
N
215
15 FIND CONNECTION STATE FOR THIS PACKET ID
216
16 SET CONNECTION STATE TO LIVE
17 GET A STORED COMMUNICATION CHANNEL
217

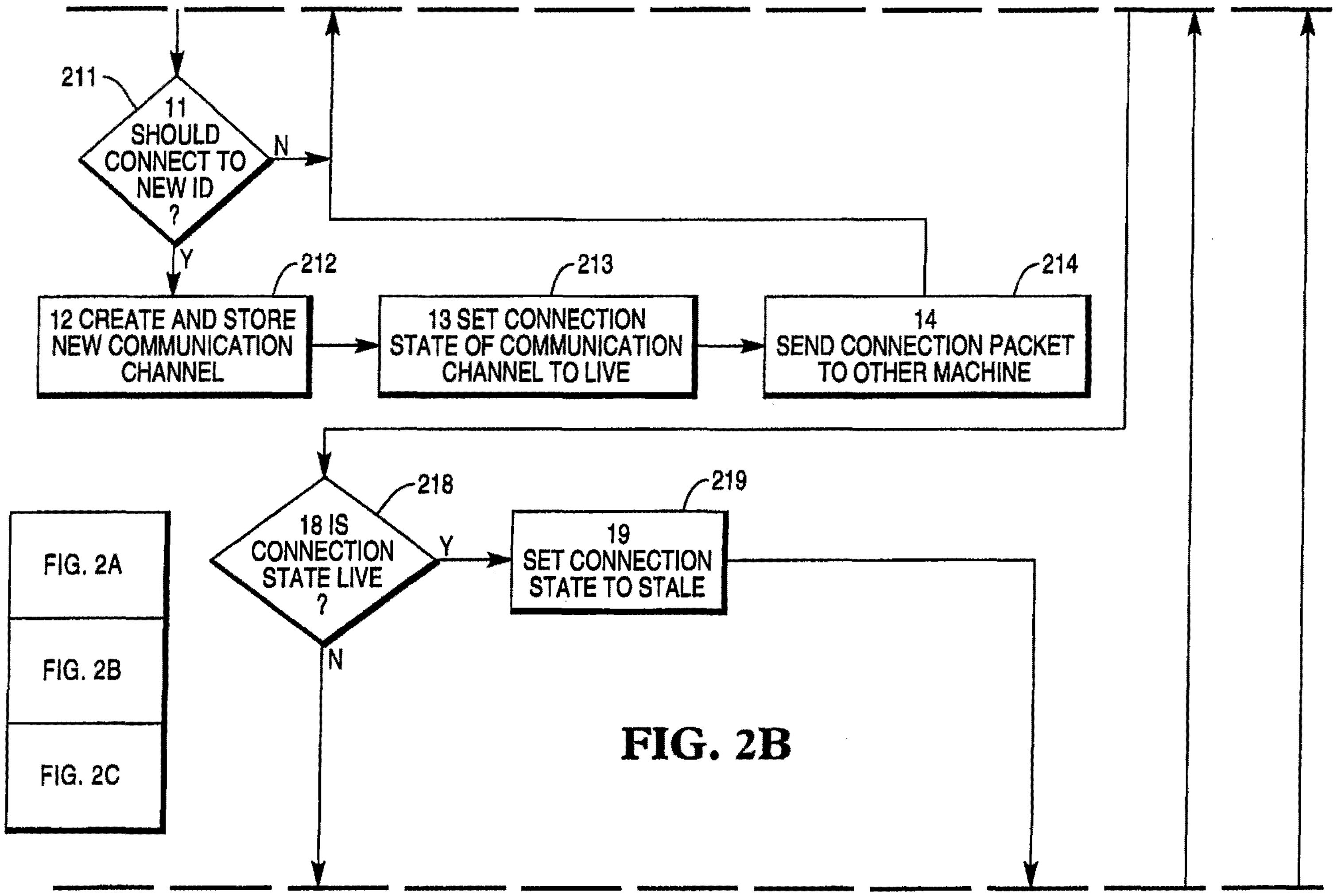
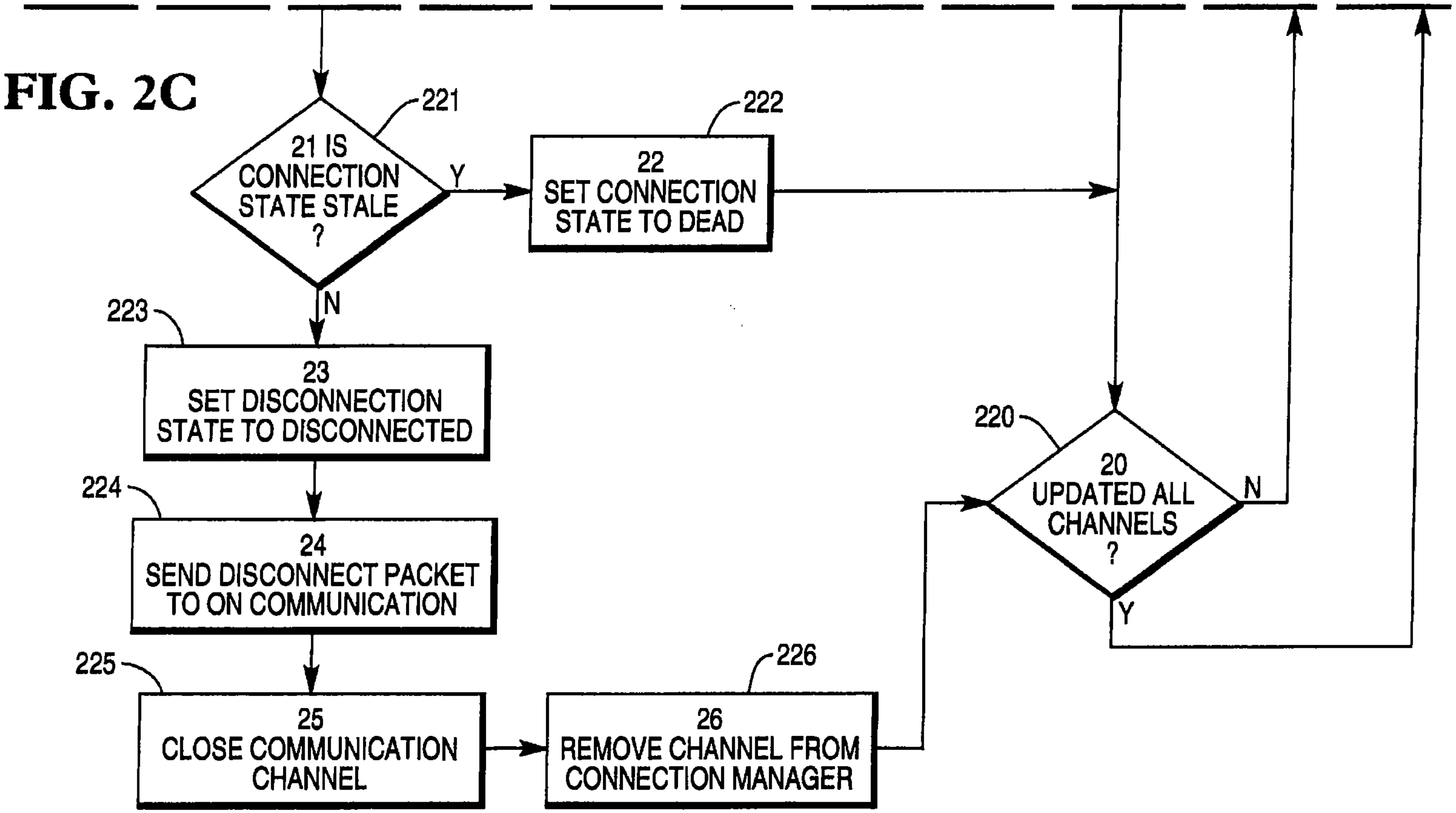

211
11 SHOULD CONNECT TO NEW ID ?
N
Y
212
12 CREATE AND STORE NEW COMMUNICATION CHANNEL
213
13 SET CONNECTION STATE OF COMMUNICATION CHANNEL TO LIVE
214
14 SEND CONNECTION PACKET TO OTHER MACHINE
218
18 IS CONNECTION STATE LIVE ?
Y
N
219
19 SET CONNECTION STATE TO STALE
FIG. 2A
FIG. 2B
FIG. 2C

FIG. 2B

FIG. 2C


21 IS CONNECTION STATE STALE ?
221
Y
N
22 SET CONNECTION STATE TO DEAD
222
223
23 SET DISCONNECTION STATE TO DISCONNECTED
224
24 SEND DISCONNECT PACKET TO ON COMMUNICATION
225
25 CLOSE COMMUNICATION CHANNEL
226
26 REMOVE CHANNEL FROM CONNECTION MANAGER
220
20 UPDATED ALL CHANNELS ?
N
Y

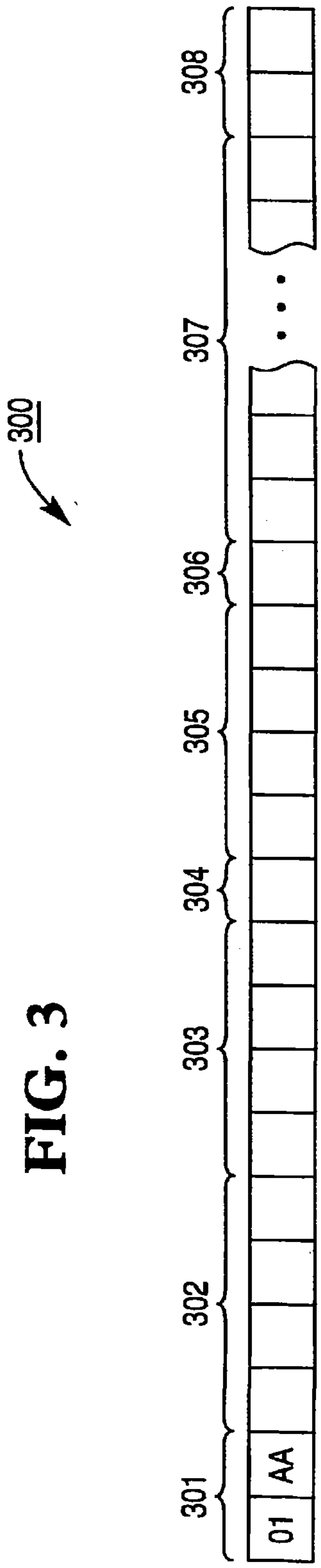


FIG. 3

DYNAMIC CONFIGURATION OF WIRELESS NETWORKS

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to the management of computer devices within a wireless network, where the computer devices are apt to appear and disappear on a regular basis.

2. Description of the Prior Art

Wireless networks connecting computer devices are a necessary component of distributed, collaborative, and portable applications. However wireless networks pose a unique set of software problems to be solved. Traditional wired networks are relatively static. When a new computer or branch of the network is added, it can be assumed that it will exist for a lengthy period of time. In a dynamic, volatile user environment such as emergency medical care, people and computer components can be arriving and departing on a minute by minute basis. If the computer components are running applications that wish to share data, or if the computers wish to share applications themselves, it is necessary to automatically manage the appearance and disappearance of network connections.

Unfortunately, existing networking methodologies are primarily designed to assume that the various computer devices are static—e.g., that they will always be in range and connected to the network. Of course, this may not always be the case, especially in certain applications. For example, with respect to the emergency medical care environment mentioned previously, medical technicians and medical vehicles may each be outfitted with computer devices for collecting and sharing medical information, etc. Because such computer devices may be brought into use with each other in a dynamic fashion, it cannot be assumed that any of these devices will be present, nor in any specific configuration. Therefore, prior art wireless network maintenance systems are simply unsuited to such environments.

There is therefore a significant need in the art for a system for dynamically managing a wireless network of computer devices, so as to ensure that each device will properly be connected to the network.

SUMMARY OF THE INVENTION

In the present invention, each computer that is to participate in the dynamic network continuously broadcasts its address to any other computer within range of the wireless network hardware. To minimize the overhead on the available communications bandwidth, this broadcast only contains a number identifying this message as an address broadcast and another number representing the address of the sending machine. This message must be sent as often as the network is expected to change (for example, once per minute for a highly dynamic network).

When a computer receives a broadcast message from a machine it is not currently connected to, it can then use any standard communications protocol (i.e., TCP/IP) to establish a connection to the broadcasting machine. Depending on the application requirements, a set of rules might be consulted to decide whether to connect to a particular machine or not. Once the connection is established, a message is sent to the broadcasting machine notifying it of the new connection. This allows for either client/server, peer-to-peer, or other communications strategies to be implemented, depending on the application.

Upon establishing a new connection between a pair of computers, a data synchronization protocol is employed to exchange data, applications, or configure services. Each machine in the newly connected pair sends a set of messages to the other. Each of these messages contains an identifier for the data object, application, or service the computer can supply along with the status of the data object, application, or service (the status could be the last date and time a particular data object was updated, for example). When a machine receives a data synchronization message, it can look at the identifier and status to decide whether to send a request to its partner machine. For example, if it notices that a data object it has interest in has a more recent update time stamp, it can request a new copy of the data object. The number and type of data synchronization messages and the response to those messages can vary to satisfy specific application requirements.

In a mobile environment, it is likely that a computer could move out of communication contact for a brief time (seconds or minutes) and then come back into range. To avoid having many disconnects, reconnects, and data synchronizations happening, a connection degradation strategy is used. When a connection is first established or when any data is received from a connection (including a broadcast message), that connection is marked as LIVE. At regular timed intervals, all the connections a machine has are downgraded one level. From LIVE, a connection moves to STALE; from STALE, to DEAD; and from DEAD, to DISCONNECTED. The amount of time between downgrades should be closely tied to the broadcast message rate. A simple implementation is to downgrade before each broadcast, if it can be assumed that each machine participating in the dynamic network is broadcasting at the same rate. If an application attempts to send data on a connection that is any state other than LIVE, the data is queued up and not transmitted. As soon as the connection becomes LIVE again, any queued data can be sent according to any scheduling rules that are in place. Once a connection has reached the DISCONNECTED state, any termination required by the underlying communications protocol can be done. No data can be sent to or received from the machine at the other end of a DISCONNECTED connection until a broadcast message has caused a reconnect and data synchronization to happen.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram depicting one instantiation of a dynamic network of the present invention.

FIG. 2 is a diagram depicting the process performed by the system of FIG. 1.

FIG. 3 is a diagram depicting an example of the format of a data packet that may be transmitted between the processing systems of FIG. 1.

DETAILED DESCRIPTION OF THE INVENTION

A preferred embodiment of the invention is now described in detail. Referring to the drawings, like numbers indicate like parts throughout the views.

FIG. 1 is a block diagram depicting one instantiation of a dynamic network of the present invention, linking a first processing system **101** to a second processing system **102** via wireless radio frequency (RF) connection **121**. Each processing system may consist of a hardware module called Wireless Network Hardware **111** and the software modules/objects called Communication Configuration **112**, Security Manager **113**, Connection Manager **114**, Communication Channel **115**, and Packet Router **116**.

The Connection Manager 114 is the main software module controlling the dynamic network. It is responsible for monitoring incoming network data packets, detecting new processing systems on the network, creating new communication channels within the wireless broadcast 121, detecting when processing systems leave the network, and closing communication channels. There is one Connection Manager 114 running on each processing system 101, 102, etc.

The Security Manager 113 controls which processing systems are allowed to connect and which data is allowed to be accessed by these processing systems. There is one Security Manager 113 running on each processing system.

A Communication Channel module 115 is used to send application data from one processing system to another processing system (e.g., from system 101 to system 102) via the wireless network 121. It is responsible for ensuring data delivery in the proper sequence without error. There can be many Communication Channel modules 115 on each processing system.

The Packet Router 116 receives data from the Wireless Network Hardware 111 and forwards it to other software modules that have registered for the data. It understands the format of application data packages and constructs these packets by reading network data bytes from the Wireless Network Hardware 111. One Packet Router 116 is running in each processing system.

The Communication Configuration module 112 allows users to edit and store parameters that control the actions taken by the other software modules. For this portion of the dynamic network, valid parameters include processing system identification, connection and data transmission permissions, broadcast and connection degradation timeout values, and any configuration parameters needed for a communication channel. There is also one Communication Configuration module 112 on each processing system.

The Wireless Network Hardware 111 is the interface between processing systems (via wireless RF broadcast 121). In one embodiment, hardware 111 may comprise a WaveLAN system, available from AT&T, but may also be implemented with cellular phones and modems, PCS systems, satellite telephones, CB radios, or any other wireless communication system.

FIG. 2 depicts the flow of control for the dynamic network configuration of FIG. 1. In a preferred embodiment, the process of FIG. 2, as well as the processes performed by software modules/objects 112–116, may be programmed for IBM-compatible PCs operating under the Windows environment, available from Microsoft Corporation, or equivalent. Other suitable environments include Unix, Macintosh, or any other programming environment.

In summary, steps 201–204 are initialization steps. Steps 205–209 form the main processing loop for the Connection Manager 114. In steps 210–214, the Connection Manager 114 responds to connection requests coming from other processing systems. (received via wireless connection 121 through the Wireless Network Hardware 111). Steps 215–226 form the basis for detecting when a processing system has left the network. 7569

FIG. 2 will now be described in further detail. In steps 201–204, when each processing system is started, it first creates a new Connection Manager 114 and a new Packet Router 116. For example, if Smalltalk is used as a programming language, the message new is sent to the ConnectionManager class and the result saved in a variable for future reference. In C++, the new operator would be invoked for the ConnectionManager class and the result saved in a variable. In C, the malloc( ) function would be used to allocate a ConnectionManager data structure and the result saved in a variable.

The Connection Manager 114 then informs the Packet Router 116 that it wants to receive all data packets that arrive from wireless connection 121 via the Wireless Network Hardware 111. The Packet Router 116 stores this request in a suitable internal table. The Connection Manager 114 also reads and stores the identifier for this processing system from the Communication Configuration 112. It also reads values for the broadcast timer and the connection degradation timer and starts these two timers running. In one embodiment, these timers may be implemented by using standard Microsoft Windows functions to create and start operating system timers that send periodic messages to the software modules/objects.

In step 205, the Connection Manager 114 creates a network data packet containing the identifier for this processing system and sends it to the Packet Router 116, instructing the Packet Router 116 to broadcast the packet through the Wireless Network Hardware 111 and connection 121 to all other processing systems within transmission range.

In one embodiment, all network data packets 300 may consist of a series of bytes in a common format, as depicted in FIG. 3. For example, the first two bytes 301 may always be the number 01 followed by the number AA (base 16). These two bytes identify the start of a data packet 300. The next four bytes 302 represent the integer identifier of the sending processing system. These four bytes are ordered with the most significant byte of the identifier first and the least significant byte last. Then follow four bytes 303 in the same order representing the integer identifier of the receiving processing system. Next is a single byte 304 encoding the packet sequence number.

Each Communication Channel between two processing systems maintains a Current Packet Sequence Number that can be used to reorder the network packets into their proper sequence if intermediary network hardware or software either lost or delivered packets out of the intended order. The Current Packet Sequence Number is incremented and stored at 304 in each packet 300 before it is transmitted. Once the value exceeds the maximum integer that can be stored in a single byte (255), it is reset to 0.

Following the sequence number are four bytes 305 that store an integer representing the size of the entire network packet 300. These four bytes are ordered with the most significant byte of the size first and the least significant byte last. Then follows a single byte 306 that identifies the message being sent from one processing system to another. For example, a value of 1 indicates that the packet is an identification packet, a value of 2 for a connection packet, a value of 3 for a data received packet, a value of 4 for an error packet, and values of 5 and higher for application specific data.

After the message ID byte 306 are a set of data bytes 307. The number of data bytes and their meaning is dependent on the message ID byte 306. An identification packet has no data bytes 307 since the processing system identification is already encoded in the network packet. A connection packet would also have no data bytes 307 since that is needed to be known is that a connection occurred. An error packet could have one data byte 307 that indicated what error occurred. The last two bytes 308 in the network packet are checksum bytes.

Before the packet 300 is transmitted, all the bytes in the packet (excluding the two checksum bytes) are added together and the result truncated to fit into two bytes. The resulting truncated sum is stored, most significant byte followed by least significant byte, at the end 308 of the network packet. These two bytes 308 are used to detect errors in the data transmission. The receiving processing system computes the checksum for the bytes it received and compares it to the transmitted checksum 308 generated by the sending processing system. If the two values match, no error has presumably occurred.

In steps **206–209**, the Connection Manager **114** then waits for either a network packet to come in from the Packet Router **116** or for one of the timers to trigger. This loop repeats until the processing system is shut down and the Connection Manager **114** is destroyed. If the identification timer triggers (step **209**), the processing system repeats the broadcast of the identification packet (step **205**), restarts the timer, and continues looping. Otherwise, it continues waiting for timers or packets.

In step **210**, if an identification packet has arrived (step **206**), the respective processing system knows that another processing system is present on the network. The Communication Channel Module **115** looks at the identifier in this packet and tries to find an active communication channel (one whose Connection State is anything but DISCONNECTED) for the identifier. These flags may be represented as integers (for example, 0=LIVE, 1=STALE, 2=DEAD, 3=DISCONNECTED) and may be stored in a variable maintained by the Communication Channel object. If an active channel already exists, its Connection State is updated to LIVE (Step **216**) and the Connection Manager **114** continues to wait for another packet or timer (step **208**). Since all processing systems are repeatedly broadcasting identification packets, this will serve to keep a channel alive even if it is not being used to carry application data.

In step **211**, if no active communication channel is found for the incoming identifier, the Security Manager **113** is consulted to see if a new channel is allowed. The Security Manager **113** reads configuration information that the user stored via, for example, a user interface into the Communication Configuration **112**. If it is not permitted, the Connection Manager **114** continues to wait for another packet or timer (step **207**).

In steps **212–214**, if the Security Manager **113** allows a new channel to be created, the Connection Manager **114** creates and stores a new Communication Channel **115**. A new Communication Channel **115** is created in the same manner as the Connection Manager **114** (depending on the programming language used). The Connection Manager **114** may have a variable that lists all the communication channels. The new channel is given the incoming identifier and has its Connection State set to LIVE (step **213**). Once this is done, a connection packet is constructed (discussed previously) and sent to the Packet Router **116** with instructions to send the packet to the processing system via wireless broadcast **121** that sent the incoming identification packet (step **214**). This connection packet notifies the other processing system that the present processing system has accepted its request for a connection and can be used to initiate application-specific data synchronization (corresponding to the particular application of the present invention). The Connection Manager **114** then continues to wait for packets or timers (step **207**).

In steps **215–216**, if a data packet comes to the Connection Manager **114** from the Packet Router **116**, the Connection Manager **114** finds the Communication Channel over which the packet arrived and sets the Connection State of the that channel to LIVE. This keeps channels alive as long as they are being used to carry application data. After the Connection State is updated, the Connection Manager **114** continues to wait for other packets or timers (step **208**).

In steps **217–226**, once the connection degradation timer triggers (step **208**), the Connection Manager **114** loops through all the stored Communication Channels and lowers their Connection State. For example, in Microsoft Windows, the operating system calls a function in a software object of the present invention when the timer triggers. If the channel's Connection State is LIVE, it is demoted to STALE. If it is STALE, it is demoted to DEAD. If it is DEAD, it is set to DISCONNECTED, a disconnect packet is constructed and sent through the Packet Router **116** to the other processing system, the channel is closed, and the channel is removed from the list stored in the Connection Manager **114** (steps **223–226**). Note that when the Connection State reaches DISCONNECTED, it is unlikely that the disconnect packet can be sent through the channel since the other processing system is most likely out of communication range. Any application data given to a Communication Channel whose state is other than LIVE should be stored and not sent to the Packet Router **116**. When the channel's state is upgraded to LIVE, this stored data can then be sent through the Packet Router **116**.

The previous set of steps are the key to the dynamic network. It allows a processing system to temporarily disappear from and reappear on the network (by moving behind a steel wall or slightly out of range, for example) without closing the communication channel. If processing systems are sharing data in a shared peer-to-peer configuration or if some processing systems are acting as servers for others, establishing a connection between processing systems could involve a high volume of data exchange. This algorithm minimizes the overhead associated in establishing, terminating, and re-establishing Communication Channels.

The present invention has been described with respect to one exemplary embodiment. Those having ordinary skill in the art will recognize that the present invention may be implemented in a variety of ways, while falling within the scope of the accompanying patent claims. For example, the present invention may be used with a variety of wireless communication systems and protocols, and could even be used with non-wireless communication networks. Moreover, while a specific process and data format have been disclosed, it will be readily apparent that other equivalent processes and formats may also be utilized.

What is claimed is:

1. In a dynamic data network for linking a first processing system to a second processing system over a communication medium, the first processing system comprising:

(a) means for transmitting over the communication medium a first identification signal associated with the first processing system;

(b) means for detecting a second identification signal received over the communication medium from the second processing system, whereby the second identification signal is transmitted by the second processing system in response to the receipt of the first identification signal by the second processing system;

(c) means for transmitting over the communication medium to the second processing system a first connection signal, responsive to the detection of the second identification signal by the detecting means;

(d) means for establishing a communication channel between the first processing system and the second processing system, responsive to the detection of a second connection signal transmitted from the second processing system, indicative of the second processing system's receipt of the first connection signal from the first processing system;

(e) means for storing an indication of a state of said communication channel, said state including an established state, a temporarily de-established state, and a permanently de-established state; and

(f) means for storing information relating each communication channel in an established state to a detected identification signal.

2. The system of claim 1, wherein the communication medium is a wireless radio frequency connection.

3. The system of claim 1, wherein the system further comprises: means for temporarily de-establishing the communication channel between the first processing system and the second processing system, responsive to a disconnection of the first processing system and the second processing system over the communication medium.

4. The system of claim 3, wherein the system further comprises: means for re-establishing the communication channel between the first processing system and the second processing system if the first processing system and the second processing system remain disconnected for less than a pre-selected amount of time.

5. The system of claim 3, wherein the system further comprises: means for permanently de-establishing the communication channel between the first processing system and the second processing system if the first processing system and the second processing system remain disconnected for greater than a preselected amount of time.

6. A process for linking a first processing system to a second processing system over a communication medium in a dynamic data network, the process comprising the steps of:

(a) transmitting over the communication medium a first identification signal associated with the first processing system;

(b) detecting a second identification signal received over the communication medium from the second processing system, whereby the second identification signal is transmitted by the second processing system in response to the receipt of the first identification signal by the second processing system;

(c) transmitting over the communication medium to the second processing system a first connection signal, responsive to the detection of the second identification signal by the detecting means;

(d) establishing a data communication channel between the first processing system and the second processing system, responsive to the detection of a second connection signal transmitted from the second processing system, indicative of the second processing system's receipt of the first connection signal from the first processing system;

(e) storing an indication of a state of said data communication channel, said state including an established state, a temporarily de-established state, and a permanently de-established state, said indication set to indicate said established state in response to said establishing step (d); and

(f) storing information relating each data communication channel in an established state to a detected identification signal.

7. The process of claim 6, further comprising the step of: temporarily de-establishing the communication channel between the first processing system and the second processing system, responsive to a disconnection of the first processing system and the second processing system over the communication medium.

8. The process of claim 7, further comprising the step of: re-establishing the communication channel between the first processing system and the second processing system if the first processing system and the second processing system remain disconnected for less than a pre-selected amount of time.

9. The process of claim 7, further comprising the step of: permanently de-establishing the communication channel between the first processing system and the second processing system if the first processing system and the second processing system remain disconnected for greater than a pre-selected amount of time.

10. A computer network comprising:

(a) a communication medium;

(b) a first processing system comprising:

(i) means for transmitting over the communication medium a first identification signal associated with the first processing system;

(ii) means for detecting a second identification signal received over the communication medium;

(iii) means for transmitting over the communication medium a first connection signal, responsive to the detection of the second identification signal by the detecting means;

(iv) means for establishing a data communication channel over the communication medium, responsive to the detection of a second connection signal over the communication medium;

(v) means for storing an indication of a state of said data communication channel, said state including an established state, a temporarily de-established state, and a permanently de-established state; and

(vi) means for storing information relating each data communication channel in an established state to a detected identification signal; and

(c) a second processing system comprising:

(i) means for transmitting over the communication medium a second identification signal associated with the second processing system;

(ii) means for detecting over the communication medium the first connection signal; and

(iii) means for transmitting over the communication medium a second connection signal, responsive to the detection of the first connection signal by the detecting means.

11. The system of claim 10, wherein the communication medium is a wireless radio frequency connection.

12. The system of claim 10, wherein the system further comprises: means for temporarily de-establishing the communication channel between the first processing system and the second processing system, responsive to a disconnection of the first processing system and the second processing system over the communication medium.

13. The system of claim 12, wherein the system further comprises: means for re-establishing the communication channel between the first processing system and the second processing system if the first processing system and the second processing system remain disconnected for less than a pre-selected amount of time.

14. The system of claim 12, wherein the system further comprises: means for permanently de-establishing the communication channel between the first processing system and the second processing system if the first processing system and the second processing system remain disconnected for greater than a preselected amount of time.

* * * * *

Attorney Docket No. 030048003US

#7/A
LDS
12-19-03
entered

Express Mail No. EV335519837US

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

IN RE APPLICATION OF:	FRED B. HOLT *ET AL.*	EXAMINER:	DAVID R. LAZARO
APPLICATION NO.:	09/629,577	ART UNIT:	2155
FILED:	JULY 31, 2000	CONF. NO:	4317
FOR: **LEAVING A BROADCAST CHANNEL**			

Amendment Under 37 C.F.R. § 1.111

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

The present communication responds to the Office Action dated November 5, 2003 in the above-identified application. Please amend the application as follows:

Amendments to the Specification begin on page 2.

Amendments to the Abstract begin on page 3.

Amendments to the Claims are reflected in the listing of claims beginning on page 4.

Arguments/Remarks begin on page 8.

Amendments to the Specification:

//

Please replace the paragraph beginning at page 1, line 3, with the following rewritten paragraph:

A1

This application is related to U.S. Patent Application No. 09/629,576, entitled "BROADCASTING NETWORK," filed on July 31, 2000 (Attorney Docket No. 030048001 US); U.S. Patent Application No. 09/629,570, entitled "JOINING A BROADCAST CHANNEL," filed on July 31, 2000 (Attorney Docket No. 030048002 US); U.S. Patent Application No. 09/629,577, "LEAVING A BROADCAST CHANNEL," filed on July 31, 2000 (Attorney Docket No. 030048003 US); U.S. Patent Application No. 09/629,575, entitled "BROADCASTING ON A BROADCAST CHANNEL," filed on July 31, 2000 (Attorney Docket No. 030048004 US); U.S. Patent Application No. 09/629,572, entitled "CONTACTING A BROADCAST CHANNEL," filed on July 31, 2000 (Attorney Docket No. 030048005 US); U.S. Patent Application No. 09/629,023, entitled "DISTRIBUTED AUCTION SYSTEM," filed on July 31, 2000 (Attorney Docket No. 030048006 US); U.S. Patent Application No. 09/629,043, entitled "AN INFORMATION DELIVERY SERVICE," filed on July 31, 2000 (Attorney Docket No. 030048007 US); U.S. Patent Application No. 09/629,024, entitled "DISTRIBUTED CONFERENCING SYSTEM," filed on July 31, 2000 (Attorney Docket No. 030048008 US); and U.S. Patent Application No. 09/629,042, entitled "DISTRIBUTED GAME ENVIRONMENT," filed on July 31, 2000 (Attorney Docket No. 030048009 US), the disclosures of which are incorporated herein by reference.

Amendments to the Abstract:

Please add the following **new** paragraph as an **Abstract**.

A2

A method for leaving a multicast computer network is disclosed. The method allows for the disconnection of a first computer from a second computer. When the first computer decides to disconnect from the second computer, the first computer sends a disconnect message to the second computer. Then, when the second computer receives the disconnect message from the first computer, the second computer broadcasts a connection port search message to find a third computer to which it can connect.

Amendments to the Claims:

Following is a complete listing of the claims pending in the application, as amended:

1-8. (Withdrawn)

1 ~~9.~~ (Currently amended) A method of disconnecting a first computer from a second computer, the first computer and the second computer being connected to a broadcast channel, said broadcast channel forming an m-regular graph where m is at least 3, the method comprising:

when the first computer decides to disconnect from the second computer, the first computer sends a disconnect message to the second computer, said disconnect message including a list of neighbors of the first computer; and

3 X when the second computer receives the disconnect message from the first computer, the second computer broadcasts a connection port search message on the broadcast channel to find a third computer to which it can connect in order to maintain an m-regular graph, said third computer being one of the neighbors on said list of neighbors. c

2 ~~10.~~ (Original) The method of claim 1 ~~9~~ wherein the second computer receives a port connection message indicating that the third computer is proposing that the third computer and the second computer connect.

3 ~~11.~~ (Original) The method of claim 1 ~~9~~ wherein the first computer disconnects from the second computer after sending the disconnect message.

4 ~~12.~~ (Original) The method of claim 1 ~~9~~ wherein the broadcast channel is implemented using the Internet.

13. (Cancelled)

14. (Cancelled)

5 ~~15~~. (Original) The method of claim 1~~0~~ wherein the first computer and second computer are connected via a TCP/IP connection.

6 ~~16~~. (Currently amended) A method for healing a disconnection of ~~disconnecting~~ a first computer from a second computer, the computers being connected to a broadcast channel, said broadcast channel being an m-regular graph where m is at least 3, the method comprising:

~~connecting the first computer to a second computer;~~

attempting to send a message from the first computer to the second computer; ~~and~~

when the attempt to send the message is unsuccessful, broadcasting from the first computer a connection port search message indicating that the first computer needs a connection; and

having a third computer not already connected to said first computer respond to said connection port search message in a manner as to maintain an m-regular graph.

A3

7 ~~17~~. (Original) The method of claim 6 ~~16~~ including:

when a third computer receives the connection port search message and the third computer also needs a connection, sending a message from the third computer to the first computer proposing that the first computer and third computer connect.

8 ~~18~~. (Original) The method of claim 7 ~~17~~ including:

when the first computer receives the message proposing that the first computer and third computer connect, sending from the first computer to the third computer a message indicating that the first computer accepts the proposal to connect the first computer to the third computer.

9 ~~19~~. (Original) The method of claim 6 ~~16~~ wherein each computer connected to the broadcast channel is connected to at least three other computers.

20. (Cancelled)

21. (Cancelled)

10 ~~22~~. (Original) The method of claim 6 ~~16~~ wherein the broadcasting includes sending the message to each computer to which the first computer is connected.

11 ~~23~~. (Currently amended) A computer-readable medium containing instructions for controlling disconnecting of a computer from another computer, the computer and the other computer being connected to a broadcast channel, said broadcast channel being an m-regular graph where m is at least 3, comprising:

a component that, when the computer decides to disconnect from the other computer, the computer sends a disconnect message to the other computer, said disconnect message including a list of neighbors of the computer; and

R3

a component that, when the computer receives a disconnect message from another computer, the computer broadcasts a connection port search message on the broadcast channel to find a computer to which it can connect in order to maintain an m-regular graph, said computer to which it can connect being one of the neighbors on said list of neighbors.

c

c

12 ~~24~~. (Original) The computer-readable medium of claim 11 ~~23~~ including:

a component that, when the computer receives a connection port search message and the computer needs to connect to another computer, sends to the computer that sent the connection port search message a port connection message indicating that the computer is proposing that the computer that sent the connection port search message connect to the computer.

13 ~~25~~. (Original) The computer-readable medium of claim 12 ~~24~~ including:

a component that, when the computer receives a port connection message, connecting to the computer that sent the port connection message.

26. (Cancelled)

27. (Cancelled)

14 ~~28~~. (Original) The computer-readable medium of claim 11 ~~23~~ wherein the computers are connected via a TCP/IP connection.

A3

15 ~~29~~. (Original) The computer-readable medium of claim 11 ~~23~~ wherein the computers that are connected to the broadcast channel are peers.

16 ~~30~~. (Original) The computer-readable medium of claim 11 ~~23~~ wherein the broadcast channel is implemented using the Internet.

REMARKS

This communication is in response to the first Office Action dated November 5, 2003. Claims 9-30 are currently pending. Claims 1-8 have been withdrawn due to election of Claims 9-30 without traverse in response to a Restriction Requirement. In the Office Action, the Examiner noted that the Abstract is missing. An abstract has been provided herein on a separate sheet as requested by the Examiner. The Examiner also rejected Claims 9-30 as being obvious in view of U.S. Patent No. 6,618,752 to Moore et al. (Moore), U.S. Patent No. 6,353,599 to Bi et al. (Bi), and "Graph Theory with Applications" by Bondy et al. (Bondy).

The Present Claimed Invention

The claims of the present application are directed primarily towards the disconnection of a computer from a broadcast network (channel). While the present specification comprehensively covered all aspects of a broadcast network, the present claimed invention is directed towards only those specific aspects related to disconnection (voluntary or involuntary) of a computer from that network.

A connected computer disconnects from the broadcast channel either in a planned or unplanned manner. When a computer disconnects in a planned manner, it sends a disconnect message to each of its four neighbors. The disconnect message includes a list that identifies the four neighbors of the disconnecting computer. When a neighbor receives the disconnect message, it tries to connect to one of the computers on the list. In one embodiment, the first computer in the list will try to connect to the second computer in the list, and the third computer in the list will try to connect to the fourth computer in the list. If a computer cannot connect (*e.g.*, the first and second computers are already connected), then the computers may try connecting in various other combinations. If connections cannot be established, each computer broadcasts a message that it needs to establish a connection with another computer. When a computer with an available internal port receives the message, it can then establish a connection with the computer that broadcast the message.

When a computer disconnects in an unplanned manner, such as resulting from a power failure, the neighbors connected to the disconnected computer recognize the disconnection when each attempts to send its next message to the now disconnected computer. Each former neighbor of the disconnected computer recognizes that it is short one connection (*i.e.*, it has a hole or empty port). When a connected computer detects that one of its neighbors is now disconnected, it broadcasts a port connection request on the broadcast channel, which indicates that it has one internal port that needs a connection. The port connection request identifies the call-in port of the requesting computer. When a connected computer that is also short a connection receives the connection request, it communicates with the requesting computer through its external port to establish a connection between the two computers.

It is possible that a planned or unplanned disconnection may result in two neighbors each having an empty internal port. In such a case, since they are neighbors, they are already connected and cannot fill their empty ports by connecting to each other. Such a condition is referred to as the "neighbors with empty ports" condition. Each neighbor broadcasts a port connection request when it detects that it has an empty port as described above. When a neighbor receives the port connection request from the other neighbor, it will recognize the condition that its neighbor also has an empty port.

To detect this condition, which would be a problem if not repaired, the first neighbor to receive the port connection request recognizes the condition and sends a condition check message to the other neighbor. The condition check message includes a list of the neighbors of the sending computer. When the receiving computer receives the list, it compares the list to its own list of neighbors. If the lists are different, then this condition has occurred in the large regime and repair is needed. To repair this condition, the receiving computer will send a condition repair request to one of the neighbors of the sending computer which is not already a neighbor of the receiving computer. When the computer receives the condition repair request, it disconnects from one of its neighbors (other than the neighbor that is involved with the condition) and connects to the computer that sent the condition repair request. Thus, one of the original neighbors involved in the condition will have had a port filled.

However, two computers are still in need of a connection, the other original neighbor and the computer that is now disconnected from the computer that received the condition repair request. Those two computers send out port connection requests. If those two computers are not neighbors, then they will connect to each other when they receive the requests. If, however, the two computers are neighbors, then they repeat the condition repair process until two non-neighbors are in need of connections.

Distinctions Between the Prior Art and the Claimed Invention

The primary reference upon which the Examiner relies upon is the Moore patent. The Moore patent discloses a software method for multicasting information over large networks. The example given in Moore is the distribution of, for example, music to various client users over the Internet. Moore correctly identified that the client server architecture commonly used where a single server serves multiple streams of data to each of the clients can be limiting. In particular, the number of clients served is limited by the capacity of the server and the bandwidth of the server's connection to the network (such as the Internet).

Instead, Moore proposes what is characterized as a daisy chain arrangement where clients act as "mini-servers" to forward the data stream onto other clients. Perhaps this can be best seen in Figure 5B where the server 206 serves a data stream to a first child host 506. When a second child host 504 wishes to access the data stream, the child host 504 is connected to the child host

506, rather than to the original server 206. Figure 5C shows another network architecture which is similar to the daisy chaining of Figure 5B, but includes multiple branching into a tree structure. Figure 5D shows a two-level daisy chain tree structure. Importantly, in all of the network architectures shown in the Moore patent, in no instance can it be considered that the architecture of Moore describes **a regular graph**.

Furthermore, as noted by the Examiner, column 10 of the Moore patent does disclose a method for disconnecting one of the child hosts from the network. The method described in the Moore patent is a simplistic method which connects the upstream host to the downstream host of the disconnected computer.

The Bi patent is cited for the proposition of teaching the use of sending a connection port search message to find a computer that is available for connection.

The Bondy reference is cited for the general proposition of teaching graph theory as applied to computer systems. Bondy mentions that the use of graph theory can be applied to computer networks to insure greater reliability. However, there is no teaching in Bondy as to how to disconnect a computer from a network and have the remaining computers in the network form new interconnections.

In response to the Examiner's arguments, applicants have amended the independent claims 9, 16, and 23 to include limitations that are not fairly shown in the cited references and that are not rendered obvious by the cited references. Specifically, each of the independent claims now require that the broadcast channel forms an M-regular graph with its constituent computers. The corresponding dependent claims 14, 21, and 20 have been cancelled. Further, each of the independent claims have been amended to indicate the importance that the graph has an "M" value of at least 3. Therefore, the corresponding dependent claims related to that limitation have been deleted as well.

After review of the cited references, applicants believe that the amendments to the claims place this case in condition for allowance. In particular, the network architecture described by Moore clearly is not a graph structure, let alone an M-regular graph structure with an M at least equal to 3. Instead, the Moore patent discloses a computer architecture that is at best a tree structure where information and data only flow in one direction. In contrast, in a multicasting graph structure of the present invention, data flows from each computer to all of the other computers in its multicast list. The Examiner attempts to remedy the differences between the Moore patent and the claimed invention by citing Bondy. Still, it is difficult to see how it would have been obvious to combine the disconnection techniques of Moore with the graph theory teachings of Bondy.

As set forth in column 10 of Moore, the only discourse as to how a computer can leave the network while the network reconfigures itself is where in a daisy chain system, the client upstream and the client downstream of the disconnected computer form a connection. This protocol for disconnection is simplistic because the network architecture itself is simplistic. There is simply no other way to reconfigure the network upon having a computer leave. In contrast, because of the complexities of an architecture that incorporates graph theory ideas, the present invention provides important methods and techniques for reconfiguring the M-regular graph that is the computer network upon disconnection of a computer.

Therefore, claim 9 has been amended to indicate that when a voluntary disconnection takes place, the disconnecting computer **sends a list of its neighbors to all of its neighbors**. The neighbors of the first computer can then receive that list and **can attempt to connect to other computers on that list**. This type of complex disconnection and healing process of a regular graph computer network is not fairly shown in the Moore nor the Bondy references. For this reason, claims 9-12 and 15 are in condition for allowance.

Claims 16-19 and 22 relate to the situation where a computer is involuntarily disconnected from the M-regular graph computer network. Claim 16 has been amended to indicate that the healing process of the computer network is performed in a way such as to maintain the M-regular graph nature of the computer network. Once again, as noted above, because Moore teaches a simple non-graph architecture where disconnections are easily handled, there would be no incentive to combine the graph theory of Bondy with the Moore teachings. Therefore, claims 16-19 and 22 are in condition for allowance.

Claims 23-25 and 28-30 mirror claims 9-12 and 15. Thus, these claims are in condition for allowance for the same reasons as those claims.

As seen from the remarks set forth above, at the heart of this case is whether or not it is obvious to combine the deficient teachings of Moore with Bondy. Applicants respectfully submit that the Examiner has failed to carry the burden. The Examiner's conclusory remarks as to obvious cannot satisfy his burden under prevailing case law. According to controlling caselaw, the motivation to combine references cannot be based on mere common knowledge and common sense as to benefits that would result from such a combination, and instead must be based on specific teachings in the prior art, such as a specific suggestion in a prior art reference.

For example, last year the Federal Circuit rejected an argument by the PTO's Board of Patent Appeals and Interferences that the ability to combine the teachings of two prior art references to produce beneficial results was sufficient motivation to combine them, and overturned the Board's finding of obviousness because of the failure to provide a specific motivation in the prior art to combine the two prior art references.[1] The Manual of Patent Examining Procedure ("MPEP") provides similar instructions.[2]

[1] In *In re Sang-Su Lee*, the Federal Circuit last year indicated the following:

Conversely, and in a similar manner to the arguments rejected by the Federal Circuit, the Examiner's motivation to combine these three prior art references is based solely on the alleged beneficial results that would result from combining them, with no motivation from the prior art cited to support the combination. Therefore, given the record, applicant respectfully submits that the Examiner's rejections are improper.

The Nortrup reference describes a television set having a menu display by which the user can adjust various picture and audio functions; however, the Nortrup display does not include a demonstration of how to adjust the functions. The Thunderchopper Handbook describes the Thunderchopper game's video display as having a "demonstration mode" showing how to play the game . . . Lee appealed to the Board, arguing that . . . the prior art provided no teaching or motivation or suggestion to combine this reference [Thunderchopper] with Nortrup . . . On the matter of motivation to combine the Nortrup and Thunderchopper references, . . . review of the Examiner's Answer reveals that the examiner merely stated that both the Nortrup function menu and the Thunderchopper demonstration mode are program features and that the Thunderchopper mode "is user-friendly" and it functions as a tutorial, and that it would have been obvious to combine them.

When patentability turns on the question of obviousness, the search for and analysis of the prior art includes evidence relevant to the finding of whether there is a teaching, motivation, or suggestion to select and combine the references relied on as evidence of obviousness. See, e.g., . . . In re Dembiczak, 175 F.3d 994, 999, 50 USPQ2d 1614, 1617 (Fed. Cir. 1999) ("Our case law makes clear that the best defense against the subtle but powerful attraction of a hindsight-based obviousness analysis is rigorous application of the requirement for a showing of the teaching or motivation to combine prior art references."); In re Dance, 160 F.3d 1339, 1343, 48 USPQ2d 1635, 1637 (Fed. Cir. 1998) (there must be some motivation, suggestion, or teaching of the desirability of making the specific combination that was made by the applicant); In re Fine, 837 F.2d 1071, 1075, 5 USPQ2d 1596, 1600 (Fed. Cir. 1988) ("'teachings of references can be combined only if there is some suggestion or incentive to do so.'") (emphasis in original) (quoting ACS Hosp. Sys., Inc. v. Montefiore Hosp., 732 F.2d 1572, 1577, 221 USPQ 929, 933 (Fed. Cir. 1984)). . . .

With respect to Lee's application, neither the examiner nor the Board adequately supported the selection and combination of the Nortrup and Thunderchopper references to render obvious that which Lee described. The examiner's conclusory statements . . . do not adequately address the issue of motivation to combine.
In re Sang-Su Lee, 277 F.3d 1338, at 1341-1343, (Fed. Cir. 2002).

[2] To establish a prima facie case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations. *The teaching or suggestion to make the claimed combination and the reasonable expectation of success must both be found in the prior art,* not in applicant's disclosure. In re Vaeck, 947 F.2d 488, 20 USPQ2d 1438 (Fed. Cir. 1991). See Manual of Patent Examining Procedure, § 2143 (emphasis added).

In view of the foregoing, the claims pending in the application comply with the requirements of 35 U.S.C. § 112 and patentably define over the applied art. A Notice of Allowance is, therefore, respectfully requested. If the Examiner has any questions or believes a telephone conference would expedite prosecution of this application, the Examiner is encouraged to call the undersigned at (206) 359-6488.

Respectfully submitted,
Perkins Coie LLP

Date: 12/11/03

Chun M. Ng
Registration No. 36,878

Correspondence Address:
Customer No. 25096
Perkins Coie LLP
P.O. Box 1247
Seattle, Washington 98111-1247
(206) 359-8000

12-12-03 2155

Attorney Docket No. 030048003US

Express Mail No. EV335519837US

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

IN RE APPLICATION OF:	FRED B. HOLT *ET AL.*	EXAMINER:	DAVID R. LAZARO
APPLICATION NO.:	09/629,577	ART UNIT:	2155
FILED:	JULY 31, 2000	CONF. NO:	4317
FOR: **LEAVING A BROADCAST CHANNEL**			

Transmittal of Amendment Under 37 C.F.R. § 1.111

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

1. Transmitted herewith are the following:

☒ Amendment Under 37 C.F.R. § 1.111
□ Petition for -Month Extension of Time
□ Terminal Disclaimer
□ Sequence Listing printout, floppy diskette, matching declaration
□ Information Disclosure Statement, Form PTO-1449 (modified), References
□ Check in the amount of $.

2. Entity Status

□ Small Entity Status (37 C.F.R. § 1.9 and § 1.27) has been established by a previously submitted Small Entity Statement.

3. Conditional Petition for Extension of Time:

Applicant petitions for an Extension of Time, if necessary, for timely submission of this transmittal and enclosures.

4. Fee Calculation and Payment

For:	(Col. 1) No. Filed	(Col. 2) No. Extra	Small Entity			Other Than a Small Entity	
			Rate	Fee		Rate	Fee
Total Claims	24	0	x $ 9 =	$	or	x $ 18 =	$0
Independent Claims	4	0	x $43 =	$	or	x $ 86 =	$0
□ Multiple Dependent Claim Presented			+ $145 =	$	or	+ $290 =	$
□ Extension of Time Fee				$			$
*If the difference in Col. 1 is less than zero, enter "0" in Col. 2.			TOTAL	$	or	TOTAL	$0

5. Provisional Fee Authorization

Please charge any underpayment in fees for timely filing of this transmittal and enclosures to Deposit Account No. 50-0665.

Respectfully submitted,
Perkins Coie LLP

Date: 12/11/03

Chun M. Ng
Registration No. 36,878

Correspondence Address:
Customer No. 25096
Perkins Coie LLP
P.O. Box 1247
Seattle, Washington 98111-1247
(206) 359-8000

P124

## *Notice of Allowability*	**Application No.** 09/629,577	**Applicant(s)** HOLT ET AL.	
	Examiner David Lazaro	**Art Unit** 2155	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address--

All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance (PTOL-85) or other appropriate communication will be mailed in due course. **THIS NOTICE OF ALLOWABILITY IS NOT A GRANT OF PATENT RIGHTS.** This application is subject to withdrawal from issue at the initiative of the Office or upon petition by the applicant. See 37 CFR 1.313 and MPEP 1308.

1. ☒ This communication is responsive to *the amendment filed in Paper 7 on 12/11//03*.
2. ☒ The allowed claim(s) is/are *9-12,15-19,22-25 and 28-30*.
3. ☒ The drawings filed on *07/31/00* are accepted by the Examiner.
4. ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
   a) ☐ All b) ☐ Some* c) ☐ None of the:
   1. ☐ Certified copies of the priority documents have been received.
   2. ☐ Certified copies of the priority documents have been received in Application No. _____ .
   3. ☐ Copies of the certified copies of the priority documents have been received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

   * Certified copies not received: _____.
5. ☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application) since a specific reference was included in the first sentence of the specification or in an Application Data Sheet. 37 CFR 1.78.
   (a) ☐ The translation of the foreign language provisional application has been received.
6. ☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121 since a specific reference was included in the first sentence of the specification or in an Application Data Sheet. 37 CFR 1.78.

Applicant has THREE MONTHS FROM THE "MAILING DATE" of this communication to file a reply complying with the requirements noted below. Failure to timely comply will result in ABANDONMENT of this application. **THIS THREE-MONTH PERIOD IS NOT EXTENDABLE.**

7. ☐ A SUBSTITUTE OATH OR DECLARATION must be submitted. Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL PATENT APPLICATION (PTO-152) which gives reason(s) why the oath or declaration is deficient.
8. ☐ CORRECTED DRAWINGS (as "replacement sheets") must be submitted.
   (a) ☐ including changes required by the Notice of Draftsperson's Patent Drawing Review (PTO-948) attached
   1) ☐ hereto or 2) ☐ to Paper No. ____.
   (b) ☐ including changes required by the proposed drawing correction filed _____, which has been approved by the Examiner.
   (c) ☐ including changes required by the attached Examiner's Amendment / Comment or in the Office action of Paper No. _____.

Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the drawings in the front (not the back) of each sheet. Replacement sheet(s) should be labeled as such in the margin according to 37 CFR 1.121(d).

9. ☐ DEPOSIT OF and/or INFORMATION about the deposit of BIOLOGICAL MATERIAL must be submitted. Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Attachment(s)

1☐ Notice of References Cited (PTO-892)
2☐ Notice of Draftperson's Patent Drawing Review (PTO-948)
3☐ Information Disclosure Statements (PTO-1449 or PTO/SB/08), Paper No. _____
4☐ Examiner's Comment Regarding Requirement for Deposit of Biological Material
5☐ Notice of Informal Patent Application (PTO-152)
6☒ Interview Summary (PTO-413), Paper No.*8* .
7☒ Examiner's Amendment/Comment
8☒ Examiner's Statement of Reasons for Allowance
9☐ Other

Patrice Winder
PATRICE WINDER
PRIMARY EXAMINER

DETAILED ACTION

RESTRICTIONS

1. This application is in condition for allowance except for the presence of Claims 1-8 to an invention non-elected without traverse. Accordingly, claims have been cancelled.

EXAMINER'S AMENDMENT

2. An examiner's amendment to the record appears below. Should the changes and/or additions be unacceptable to applicant, an amendment may be filed as provided by 37 CFR 1.312. To ensure consideration of such an amendment, it MUST be submitted no later than the payment of the issue fee.

Authorization for this examiner's amendment was given in a telephone interview with Chun Ng (Reg. No. 36,878) on 01/23/04.

The application has been amended as follows:

In the claims:

Claims 1-8 are cancelled.

In Claim 9, line 8, after "message", please insert --on the broadcast channel--.

In Claim 9, line 9, after "connect", please insert --in order to maintain an m-regular graph--.

In Claim 23, line 9, after "message", please insert --on the broadcast channel--.

In Claim 23, line 10, after "connect", please insert --in order to maintain an m-regular graph--.

3. The following is an examiner's statement of reasons for allowance: Claims 9-12, 15-19, 22-25 and 28-30 are allowable over the prior art of record based on the argument set forth by the applicant in Paper 3 starting on page 10 as directed to the presently amended claims.

Any comments considered necessary by applicant must be submitted no later than the payment of the issue fee and, to avoid processing delays, should preferably accompany the issue fee. Such submissions should be clearly labeled "Comments on Statement of Reasons for Allowance."

Any inquiry concerning this communication or earlier communications from the examiner should be directed to David Lazaro whose telephone number is 703-305-4868. The examiner can normally be reached on 8:30-5:00 M-F.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Hosain Alam can be reached on 703-308-6662. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is 703-305-3900.

David Lazaro
January 23, 2004

Patrice Winder
PATRICE WINDER
PRIMARY EXAMINER

Interview Summary

Application No.	**Applicant(s)**	
09/629,577	HOLT ET AL.	
Examiner	**Art Unit**	
David Lazaro	2155	

All participants (applicant, applicant's representative, PTO personnel):

(1) *David Lazaro*. (3)_____.

(2) *Chun Ng*. (4)_____.

Date of Interview: *23 January 2004*.

Type: a)☒ Telephonic b)□ Video Conference
c)□ Personal [copy given to: 1)□ applicant 2)□ applicant's representative]

Exhibit shown or demonstration conducted: d)□ Yes e)☒ No.
If Yes, brief description: _____.

Claim(s) discussed: *9 and 23*.

Identification of prior art discussed: _____.

Agreement with respect to the claims f)☒ was reached. g)□ was not reached. h)□ N/A.

Substance of Interview including description of the general nature of what was agreed to if an agreement was reached, or any other comments: *Claims 9 and 23 were discussed in order to expedite prosecution of the application. Applicant's representative agreed to an examiners amendment so that the body of both Claims 9 and 23 would reflect an explicit linking to the broadcast channel and its m-regular structure. The examiner's amendment will make the application allowable*.

(A fuller description, if necessary, and a copy of the amendments which the examiner agreed would render the claims allowable, if available, must be attached. Also, where no copy of the amendments that would render the claims allowable is available, a summary thereof must be attached.)

THE FORMAL WRITTEN REPLY TO THE LAST OFFICE ACTION MUST INCLUDE THE SUBSTANCE OF THE INTERVIEW. (See MPEP Section 713.04). If a reply to the last Office action has already been filed, APPLICANT IS GIVEN ONE MONTH FROM THIS INTERVIEW DATE, OR THE MAILING DATE OF THIS INTERVIEW SUMMARY FORM, WICHEVER IS LATER, TO FILE A STATEMENT OF THE SUBSTANCE OF THE INTERVIEW. See Summary of Record of Interview requirements on reverse side or on attached sheet.

Examiner Note: You must sign this form unless it is an Attachment to a signed Office action.

Examiner's signature, if required

Summary of Record of Interview Requirements

Manual of Patent Examining Procedure (MPEP), Section 713.04, Substance of Interview Must be Made of Record
A complete written statement as to the substance of any face-to-face, video conference, or telephone interview with regard to an application must be made of record in the application whether or not an agreement with the examiner was reached at the interview.

Title 37 Code of Federal Regulations (CFR) § 1.133 Interviews
Paragraph (b)
In every instance where reconsideration is requested in view of an interview with an examiner, a complete written statement of the reasons presented at the interview as warranting favorable action must be filed by the applicant. An interview does not remove the necessity for reply to Office action as specified in §§ 1.111, 1.135. (35 U.S.C. 132)

37 CFR §1.2 Business to be transacted in writing.
All business with the Patent or Trademark Office should be transacted in writing. The personal attendance of applicants or their attorneys or agents at the Patent and Trademark Office is unnecessary. The action of the Patent and Trademark Office will be based exclusively on the written record in the Office. No attention will be paid to any alleged oral promise, stipulation, or understanding in relation to which there is disagreement or doubt.

The action of the Patent and Trademark Office cannot be based exclusively on the written record in the Office if that record is itself incomplete through the failure to record the substance of interviews.

It is the responsibility of the applicant or the attorney or agent to make the substance of an interview of record in the application file, unless the examiner indicates he or she will do so. It is the examiner's responsibility to see that such a record is made and to correct material inaccuracies which bear directly on the question of patentability.

Examiners must complete an Interview Summary Form for each interview held where a matter of substance has been discussed during the interview by checking the appropriate boxes and filling in the blanks. Discussions regarding only procedural matters, directed solely to restriction requirements for which interview recordation is otherwise provided for in Section 812.01 of the Manual of Patent Examining Procedure, or pointing out typographical errors or unreadable script in Office actions or the like, are excluded from the interview recordation procedures below. Where the substance of an interview is completely recorded in an Examiners Amendment, no separate Interview Summary Record is required.

The Interview Summary Form shall be given an appropriate Paper No., placed in the right hand portion of the file, and listed on the "Contents" section of the file wrapper. In a personal interview, a duplicate of the Form is given to the applicant (or attorney or agent) at the conclusion of the interview. In the case of a telephone or video-conference interview, the copy is mailed to the applicant's correspondence address either with or prior to the next official communication. If additional correspondence from the examiner is not likely before an allowance or if other circumstances dictate, the Form should be mailed promptly after the interview rather than with the next official communication.

The Form provides for recordation of the following information:

- Application Number (Series Code and Serial Number)
- Name of applicant
- Name of examiner
- Date of interview
- Type of interview (telephonic, video-conference, or personal)
- Name of participant(s) (applicant, attorney or agent, examiner, other PTO personnel, etc.)
- An indication whether or not an exhibit was shown or a demonstration conducted
- An identification of the specific prior art discussed
- An indication whether an agreement was reached and if so, a description of the general nature of the agreement (may be by attachment of a copy of amendments or claims agreed as being allowable). Note: Agreement as to allowability is tentative and does not restrict further action by the examiner to the contrary.
- The signature of the examiner who conducted the interview (if Form is not an attachment to a signed Office action)

It is desirable that the examiner orally remind the applicant of his or her obligation to record the substance of the interview of each case. It should be noted, however, that the Interview Summary Form will not normally be considered a complete and proper recordation of the interview unless it includes, or is supplemented by the applicant or the examiner to include, all of the applicable items required below concerning the substance of the interview.

A complete and proper recordation of the substance of any interview should include at least the following applicable items:

1) A brief description of the nature of any exhibit shown or any demonstration conducted,
2) an identification of the claims discussed,
3) an identification of the specific prior art discussed,
4) an identification of the principal proposed amendments of a substantive nature discussed, unless these are already described on the Interview Summary Form completed by the Examiner,
5) a brief identification of the general thrust of the principal arguments presented to the examiner,
   (The identification of arguments need not be lengthy or elaborate. A verbatim or highly detailed description of the arguments is not required. The identification of the arguments is sufficient if the general nature or thrust of the principal arguments made to the examiner can be understood in the context of the application file. Of course, the applicant may desire to emphasize and fully describe those arguments which he or she feels were or might be persuasive to the examiner.)
6) a general indication of any other pertinent matters discussed, and
7) if appropriate, the general results or outcome of the interview unless already described in the Interview Summary Form completed by the examiner.

Examiners are expected to carefully review the applicant's record of the substance of an interview. If the record is not complete and accurate, the examiner will give the applicant an extendable one month time period to correct the record.

Examiner to Check for Accuracy

If the claims are allowable for other reasons of record, the examiner should send a letter setting forth the examiner's version of the statement attributed to him or her. If the record is complete and accurate, the examiner should place the indication, "Interview Record OK" on the paper recording the substance of the interview along with the date and the examiner's initials.

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

NOTICE OF ALLOWANCE AND FEE(S) DUE

25096 7590 01/27/2004

PERKINS COIE LLP
PATENT-SEA
P.O. BOX 1247
SEATTLE, WA 98111-1247

EXAMINER	
LAZARO, DAVID R	
ART UNIT	PAPER NUMBER
2155	

DATE MAILED: 01/27/2004

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/629,577	07/31/2000	Fred B. Holt	030048003US	4317

TITLE OF INVENTION: LEAVING A BROADCAST CHANNEL

APPLN. TYPE	SMALL ENTITY	ISSUE FEE	PUBLICATION FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	NO	$1330	$0	$1330	04/27/2004

THE APPLICATION IDENTIFIED ABOVE HAS BEEN EXAMINED AND IS ALLOWED FOR ISSUANCE AS A PATENT. PROSECUTION ON THE MERITS IS CLOSED. THIS NOTICE OF ALLOWANCE IS NOT A GRANT OF PATENT RIGHTS. THIS APPLICATION IS SUBJECT TO WITHDRAWAL FROM ISSUE AT THE INITIATIVE OF THE OFFICE OR UPON PETITION BY THE APPLICANT. SEE 37 CFR 1.313 AND MPEP 1308.

THE ISSUE FEE AND PUBLICATION FEE (IF REQUIRED) MUST BE PAID WITHIN THREE MONTHS FROM THE MAILING DATE OF THIS NOTICE OR THIS APPLICATION SHALL BE REGARDED AS ABANDONED. THIS STATUTORY PERIOD CANNOT BE EXTENDED. SEE 35 U.S.C. 151. THE ISSUE FEE DUE INDICATED ABOVE REFLECTS A CREDIT FOR ANY PREVIOUSLY PAID ISSUE FEE APPLIED IN THIS APPLICATION. THE PTOL-85B (OR AN EQUIVALENT) MUST BE RETURNED WITHIN THIS PERIOD EVEN IF NO FEE IS DUE OR THE APPLICATION WILL BE REGARDED AS ABANDONED.

HOW TO REPLY TO THIS NOTICE:

I. Review the SMALL ENTITY status shown above.

If the SMALL ENTITY is shown as YES, verify your current SMALL ENTITY status:

A. If the status is the same, pay the TOTAL FEE(S) DUE shown above.

B. If the status is changed, pay the PUBLICATION FEE (if required) and twice the amount of the ISSUE FEE shown above and notify the United States Patent and Trademark Office of the change in status, or

If the SMALL ENTITY is shown as NO:

A. Pay TOTAL FEE(S) DUE shown above, or

B. If applicant claimed SMALL ENTITY status before, or is now claiming SMALL ENTITY status, check the box below and enclose the PUBLICATION FEE and 1/2 the ISSUE FEE shown above.

□ Applicant claims SMALL ENTITY status.
See 37 CFR 1.27.

II. PART B - FEE(S) TRANSMITTAL should be completed and returned to the United States Patent and Trademark Office (USPTO) with your ISSUE FEE and PUBLICATION FEE (if required). Even if the fee(s) have already been paid, Part B - Fee(s) Transmittal should be completed and returned. If you are charging the fee(s) to your deposit account, section "4b" of Part B - Fee(s) Transmittal should be completed and an extra copy of the form should be submitted.

III. All communications regarding this application must give the application number. Please direct all communications prior to issuance to Mail Stop ISSUE FEE unless advised to the contrary.

IMPORTANT REMINDER: Utility patents issuing on applications filed on or after Dec. 12, 1980 may require payment of maintenance fees. It is patentee's responsibility to ensure timely payment of maintenance fees when due.

PART B - FEE(S) TRANSMITT

Complete and send this form, together with applicable fee(s), to: Mail **Mail Stop ISSUE FEE**
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450

or Fax (703) 746-4000

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 4 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Legibly mark-up with any corrections or use Block 1)

25096 7590 01/27/2004

PERKINS COIE LLP
PATENT-SEA
P.O. BOX 1247
SEATTLE, WA 98111-1247

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission

I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO, on the date indicated below.

(Depositor's name)

(Signature)

(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/629,577	07/31/2000	Fred B. Holt	030048003US	4317

TITLE OF INVENTION: LEAVING A BROADCAST CHANNEL

APPLN. TYPE	SMALL ENTITY	ISSUE FEE	PUBLICATION FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	NO	$1330	$0	$1330	04/27/2004

EXAMINER	ART UNIT	CLASS-SUBCLASS
LAZARO, DAVID R	2155	709-204000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).

□ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.

□ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. **Use of a Customer Number is required.**

2. For printing on the patent front page, list (1) the names of up to 3 registered patent attorneys or agents OR, alternatively, (2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.

1____________________

2____________________

3____________________

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. Inclusion of assignee data is only appropriate when an assignment has been previously submitted to the USPTO or is being submitted under separate cover. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE (B) RESIDENCE: (CITY and STATE OR COUNTRY)

Please check the appropriate assignee category or categories (will not be printed on the patent); □ individual □ corporation or other private group entity □ government

4a. The following fee(s) are enclosed:

□ Issue Fee

□ Publication Fee

□ Advance Order - # of Copies ________________

4b. Payment of Fee(s):

□ A check in the amount of the fee(s) is enclosed.

□ Payment by credit card. Form PTO-2038 is attached.

□ The Director is hereby authorized by charge the required fee(s), or credit any overpayment, to Deposit Account Number ________________ (enclose an extra copy of this form).

Director for Patents is requested to apply the Issue Fee and Publication Fee (if any) or to re-apply any previously paid issue fee to the application identified above.

(Authorized Signature) (Date)

NOTE; The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office.

This collection of information is required by 37 CFR 1.311. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, Alexandria, Virginia 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, Alexandria, Virginia 22313-1450.

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

TRANSMIT THIS FORM WITH FEE(S)

PTOL-85 (Rev. 11/03) Approved for use through 04/30/2004. OMB 0651-0033 U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

UNITED STATES PATENT AND TRADEMARK OFFICE

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

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/629,577	07/31/2000	Fred B. Holt	030048003US	4317

25096 7590 01/27/2004

PERKINS COIE LLP
PATENT-SEA
P.O. BOX 1247
SEATTLE, WA 98111-1247

EXAMINER	
LAZARO, DAVID R	
ART UNIT	PAPER NUMBER
2155	8

DATE MAILED: 01/27/2004

Determination of Patent Term Adjustment under 35 U.S.C. 154 (b)
(application filed on or after May 29, 2000)

The Patent Term Adjustment to date is 766 day(s). If the issue fee is paid on the date that is three months after the mailing date of this notice and the patent issues on the Tuesday before the date that is 28 weeks (six and a half months) after the mailing date of this notice, the Patent Term Adjustment will be 766 day(s).

If a Continued Prosecution Application (CPA) was filed in the above-identified application, the filing date that determines Patent Term Adjustment is the filing date of the most recent CPA.

Applicant will be able to obtain more detailed information by accessing the Patent Application Information Retrieval (PAIR) system (http://pair.uspto.gov).

Any questions regarding the Patent Term Extension or Adjustment determination should be directed to the Office of Patent Legal Administration at (703) 305-1383. Questions relating to issue and publication fee payments should be directed to the Customer Service Center of the Office of Patent Publication at (703) 305-8283.

PTOL-85 (Rev. 11/03) Approved for use through 04/30/2004.

BS #9
4-2-04
8

Express Mail No. EV343595835US

Attorney Docket No. 030048003US

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

IN RE APPLICATION OF:	FRED B. HOLT	EXAMINER:	DAVID R. LAZARO
APPLICATION NO.:	09/629,577	ART UNIT:	2155
FILED:	July 31, 2000	CONFIRMATION NO:	4317
FOR:	LEAVING A BROADCAST CHANNEL		

Comments on Statement of Reasons for Allowance

Mail Stop Issue Fee
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

In the Notice of Allowability mailed January 27, 2004, the Examiner allowed claims 9-12, 15-19, 22-25 and 28-30. Although the applicant's attorney agrees with the Examiner's conclusion that these claims are allowable, the applicant's attorney notes that the claims may be allowable for reasons other than those identified by the Examiner and does not concede that the Examiner's characterizations of the terms of the claims and the prior art are correct.

Respectfully submitted,
Perkins Coie LLP

Date: 3/18/04

Chun Ng
Registration No. 36,878

Correspondence Address:
Customer No. 25096
Perkins Coie LLP
P.O. Box 1247
Seattle, Washington 98111-1247
(206) 359-8000

[Q:\Clients\Boeing (03004)\8003 (Leaving)\Us01\Comments on Statement of Reasons for Allowance.doc]

BS
#10
LL

Attorney Docket No. 030048003US

Express Mail Label EV343595835US

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

IN RE APPLICATION OF:	FRED B. HOLT *ET AL.*	EXAMINER:	DAVID R. LAZARO
APPLICATION NO.:	09/629,577	ART UNIT:	2155
FILED:	JULY 31, 2000	CONF. NO:	4317
FOR: **LEAVING A BROADCAST CHANNEL**			

Transmittal of Formal Drawings

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

Further to the Notice of Allowance dated January 27, 2004, enclosed are the required formal drawings, Figs 1-24 and 26-34.

No fees are believed due in connection with this response. However, if fees are due, the Commissioner is requested to charge them to Deposit Account No. 50-0665.

Respectfully submitted,
Perkins Coie LLP

Date: 3/19/04

Chun Ng
Registration No. 36,878

Correspondence Address:
Customer No. 25096
Perkins Coie LLP
P.O. Box 1247
Seattle, Washington 98111-1247
(206) 359-8000

6732147

09624571

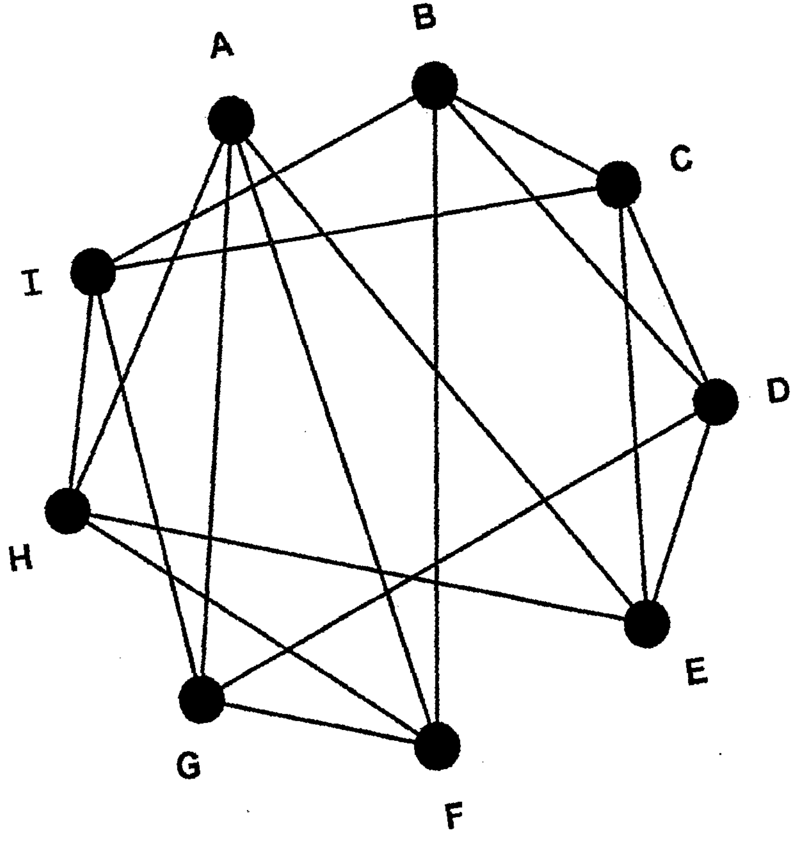


Fig. 1

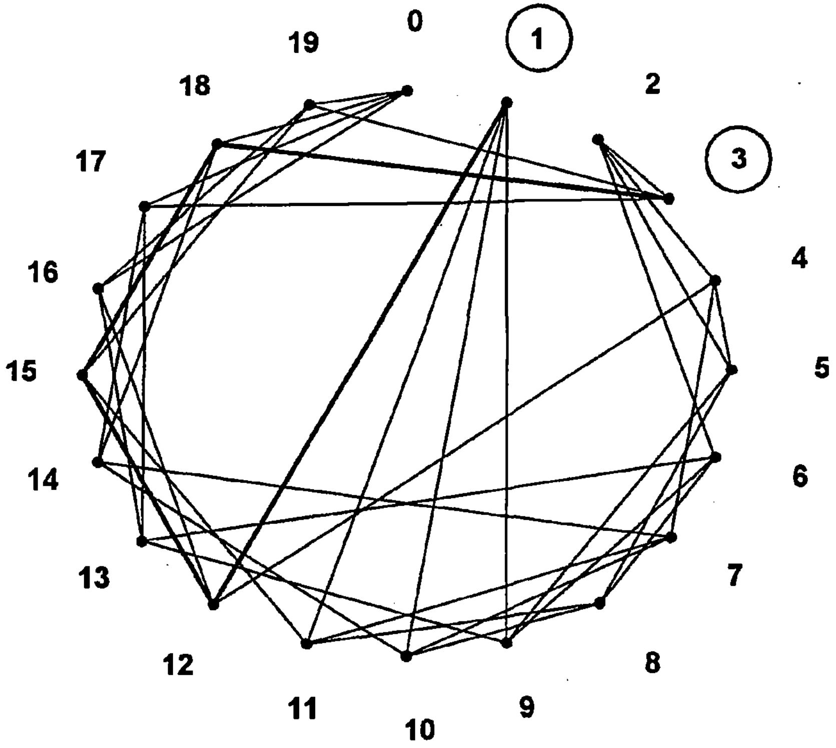


Fig. 2

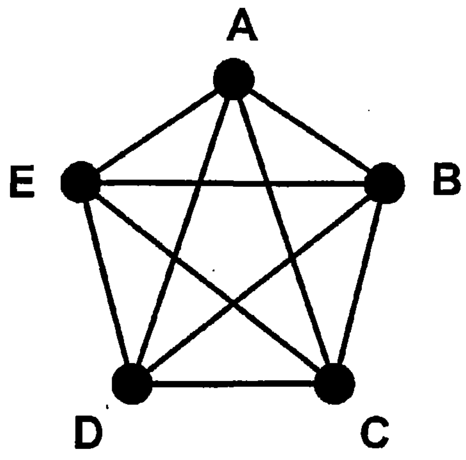


Fig. 3A

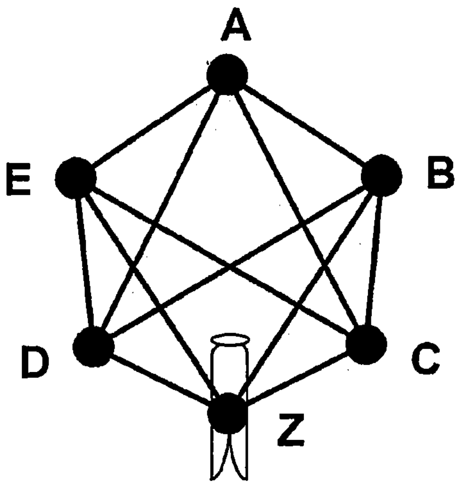


Fig. 3B

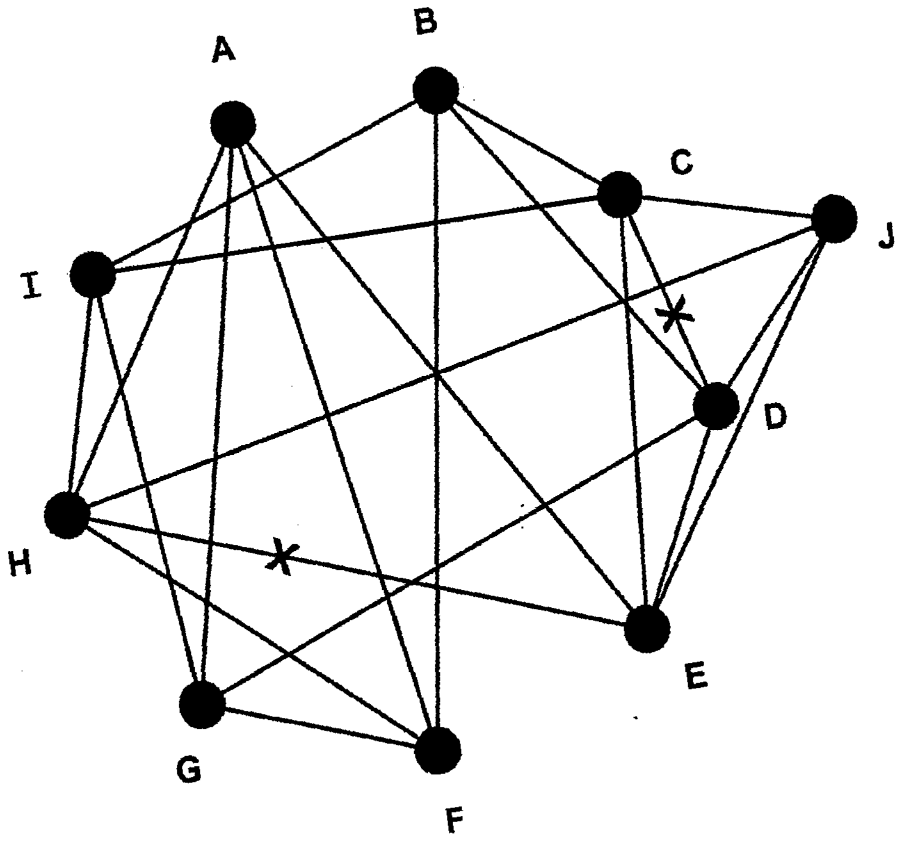


Fig. 4A

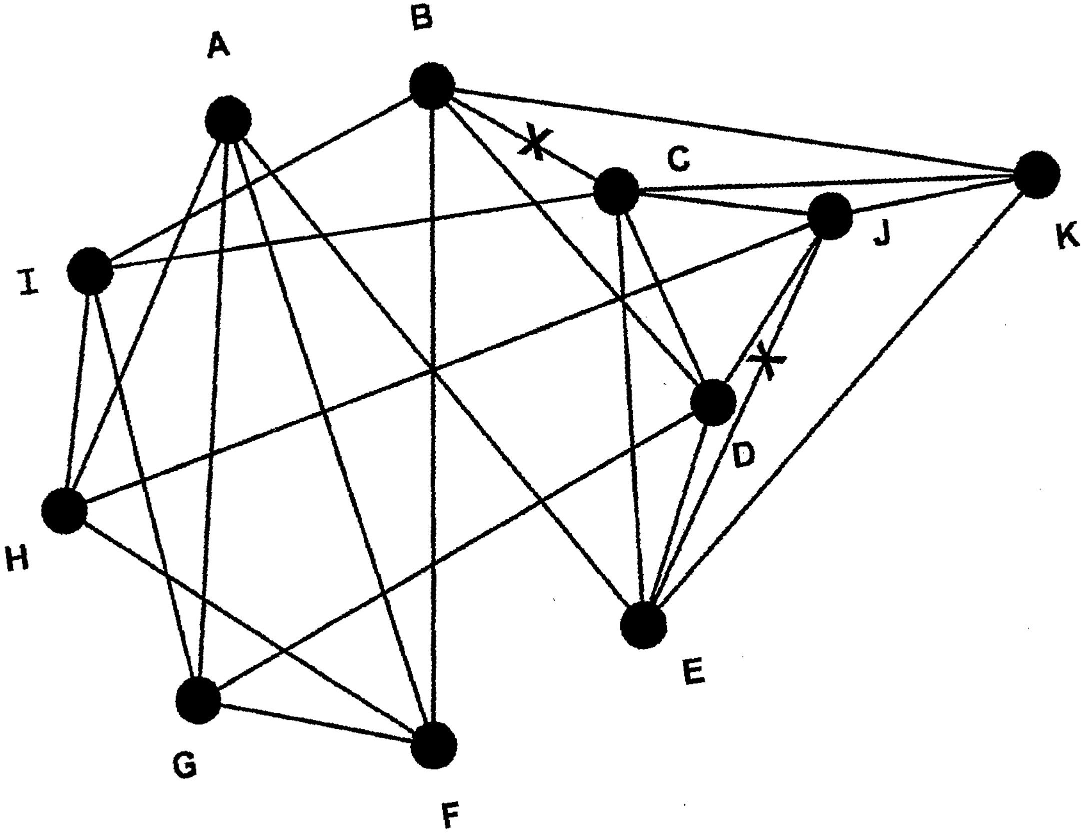


Fig. 4B

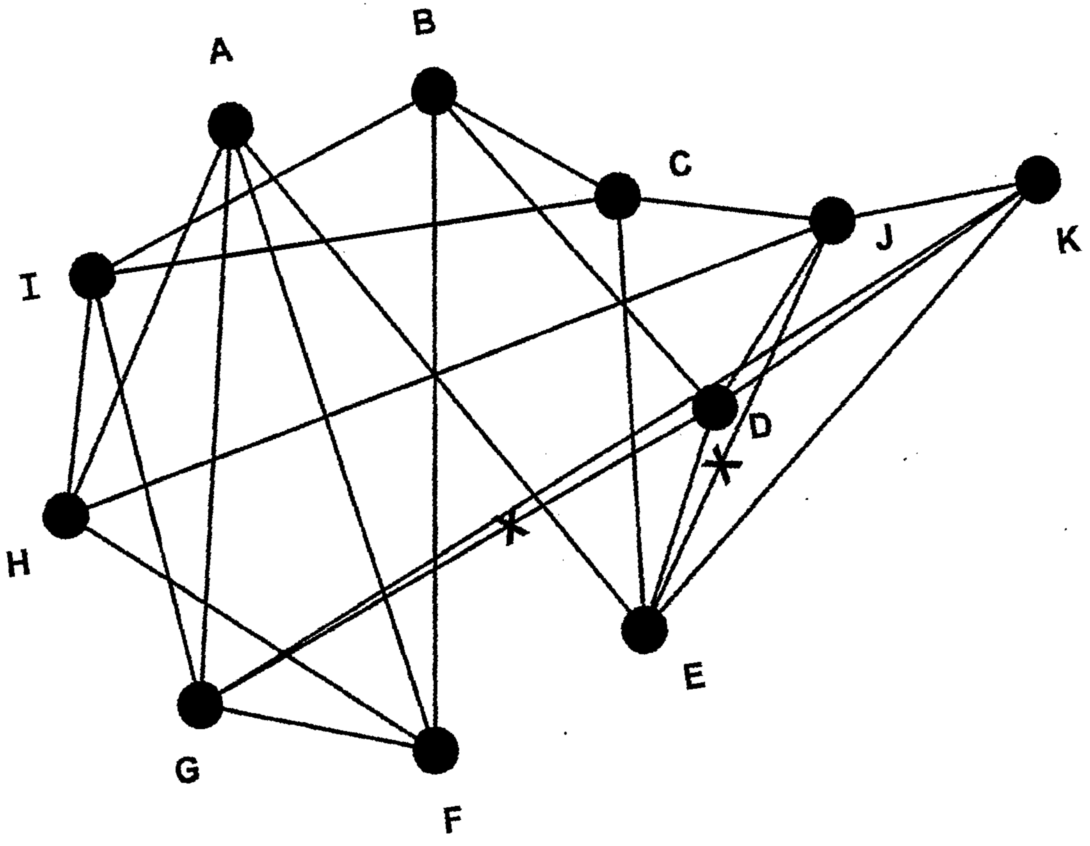


Fig. 4C

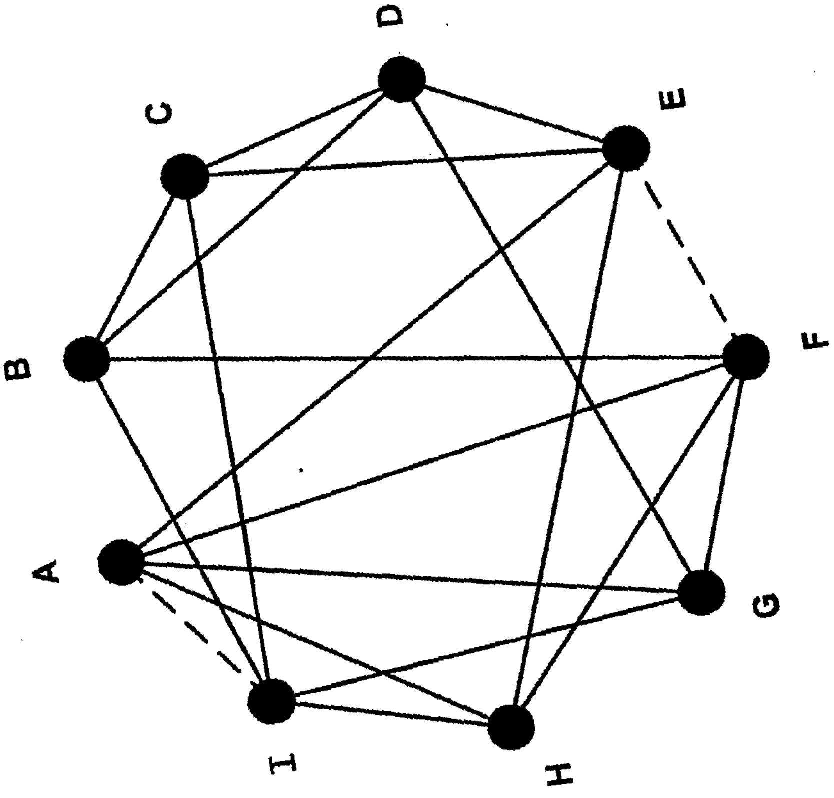


Fig. 5A

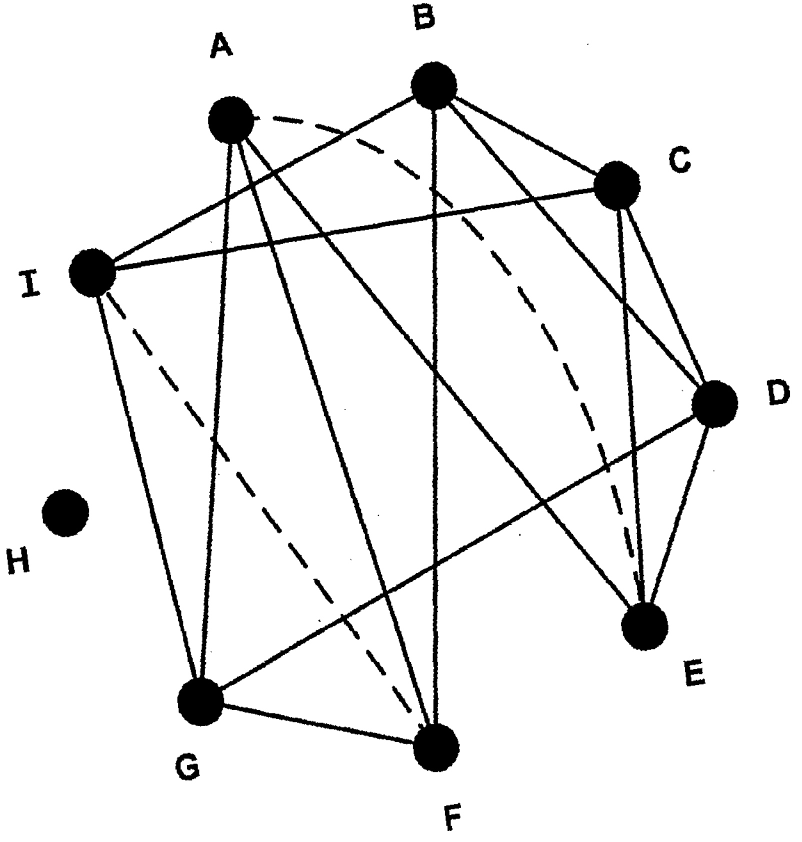


Fig. 5B

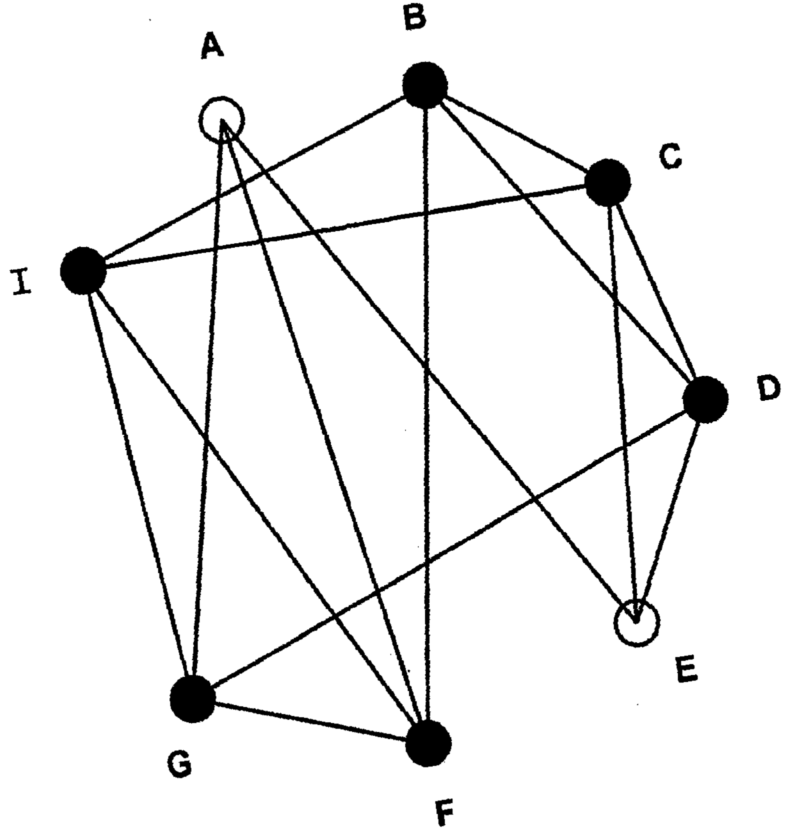


Fig. 5C

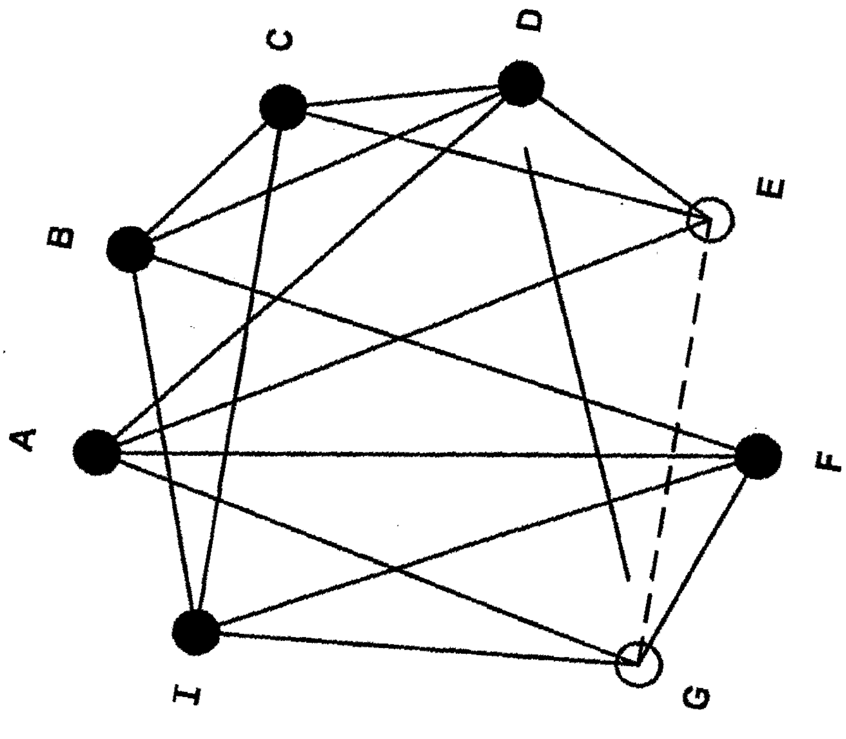


Fig. 5D

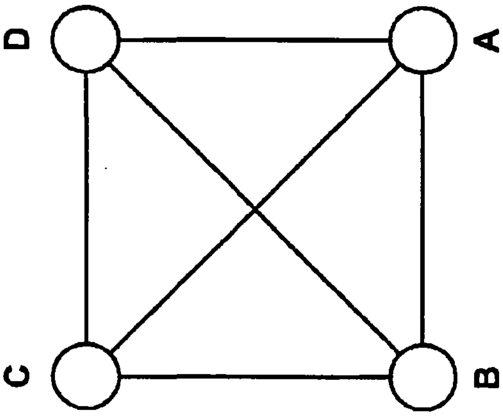


Fig. 5E

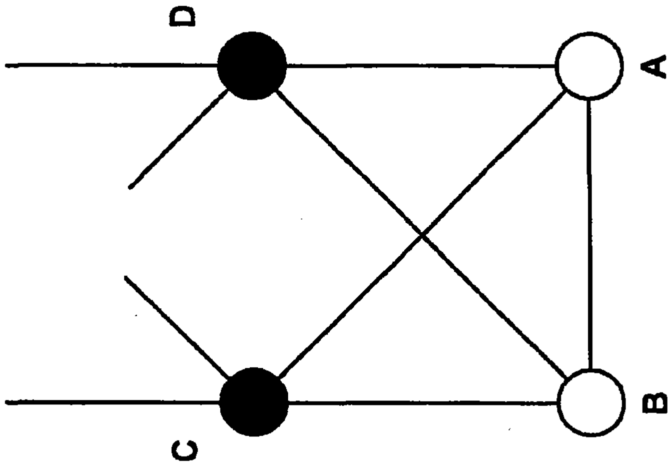


Fig. 5F

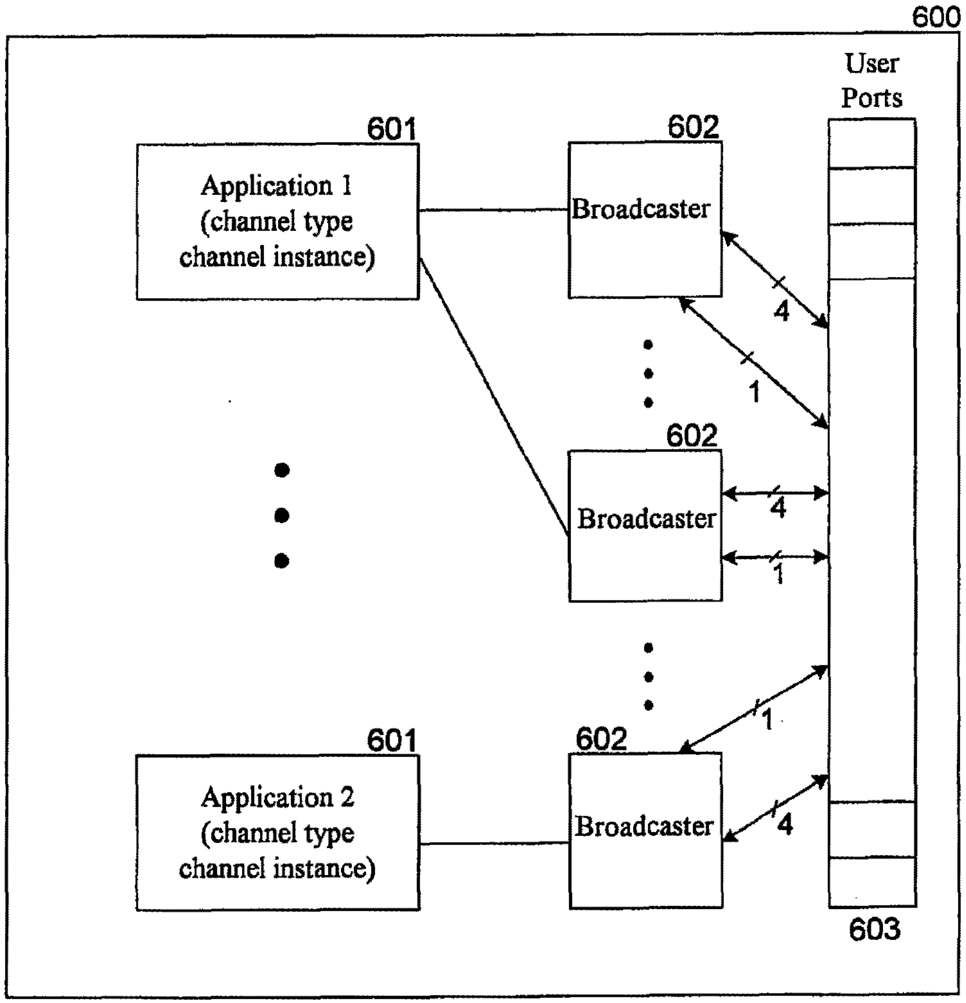

600
User Ports
601
Application 1 (channel type channel instance)
602
Broadcaster
4
1
602
Broadcaster
4
1
601
Application 2 (channel type channel instance)
602
Broadcaster
1
4
603

Fig. 6

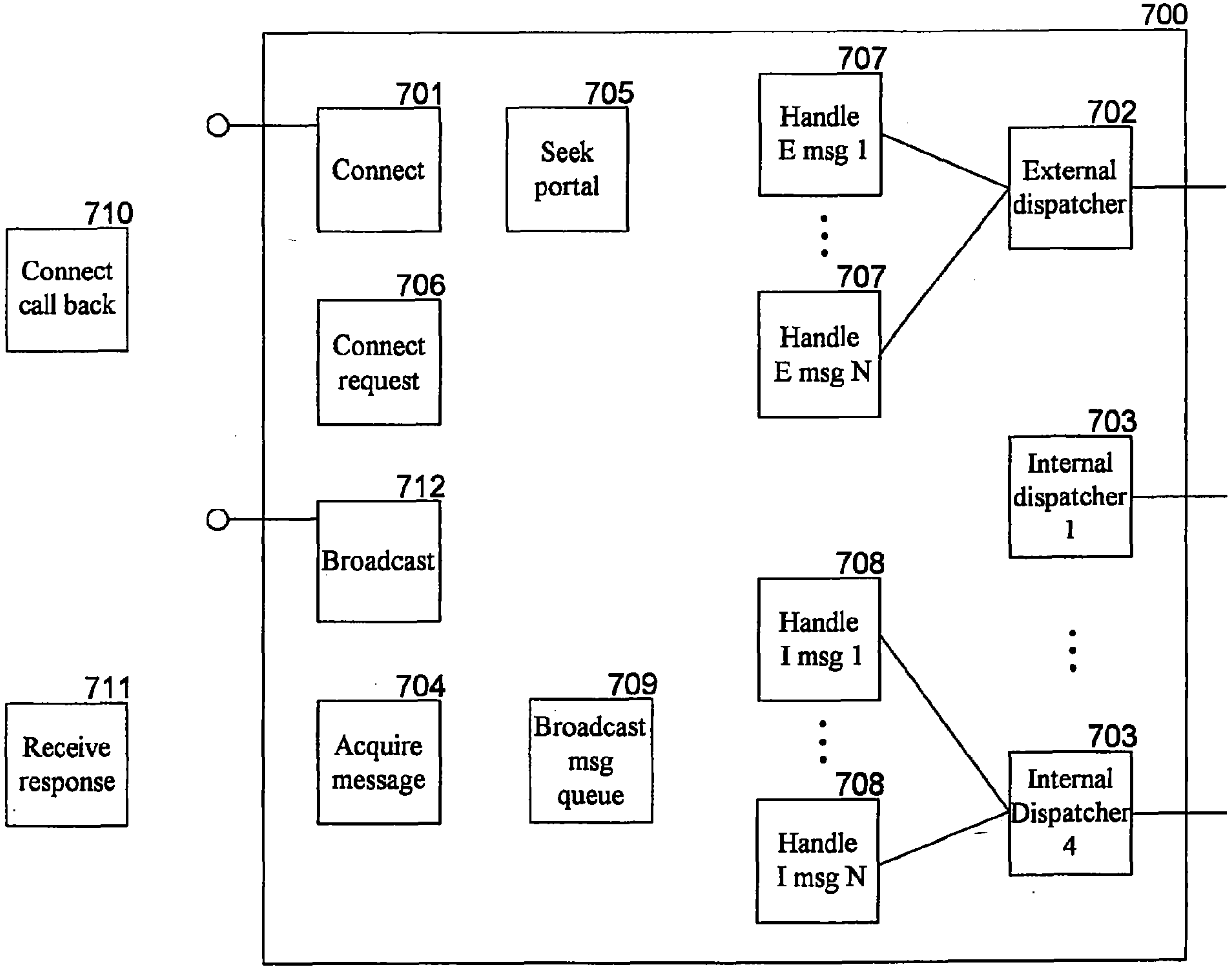

700
701
Connect
705
Seek portal
707
Handle E msg 1
707
Handle E msg N
702
External dispatcher
710
Connect call back
706
Connect request
703
Internal dispatcher 1
712
Broadcast
708
Handle I msg 1
708
Handle I msg N
711
Receive response
704
Acquire message
709
Broadcast msg queue
703
Internal Dispatcher 4

Fig. 7

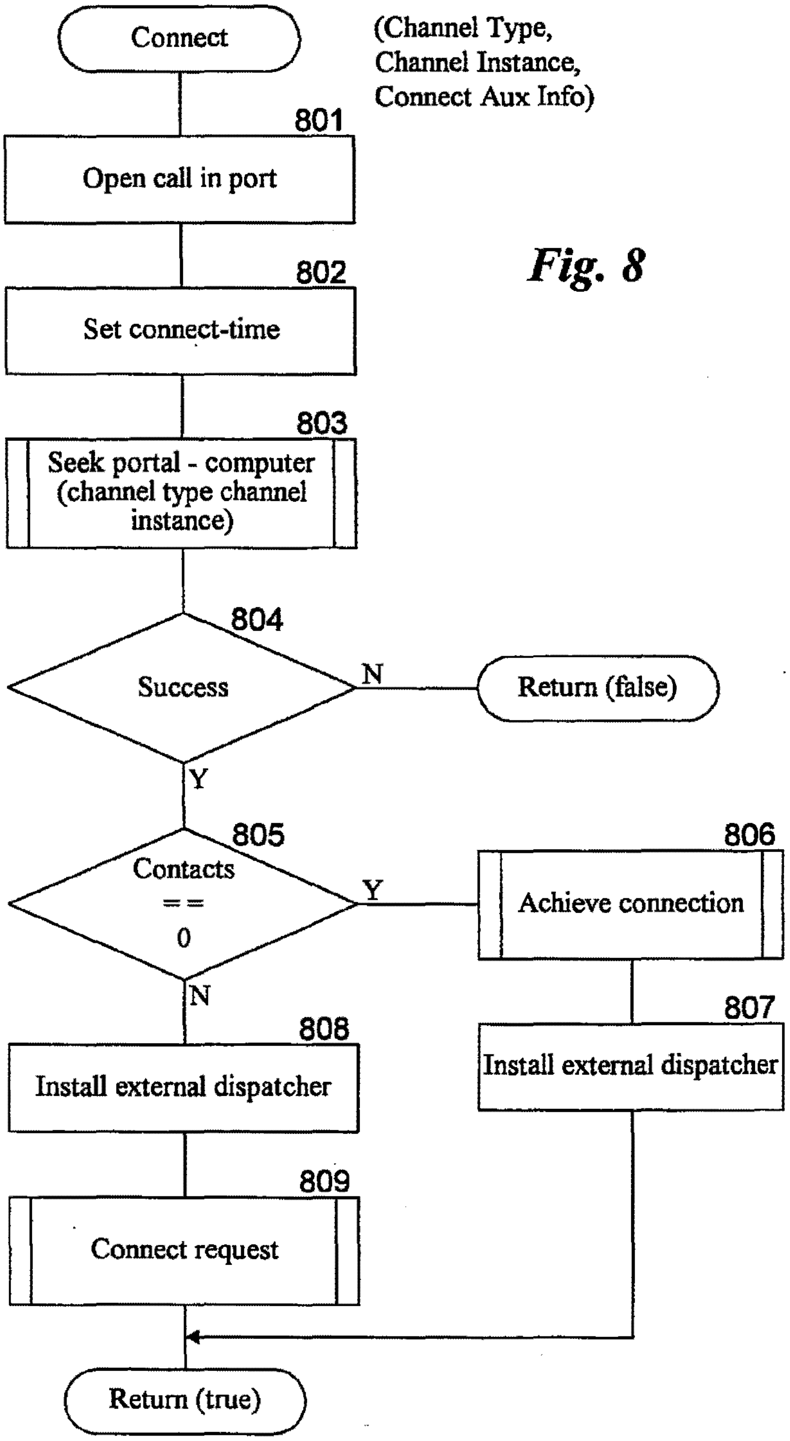

Connect
(Channel Type,
Channel Instance,
Connect Aux Info)
801
Open call in port
802
Set connect-time
803
Seek portal - computer (channel type channel instance)
804
Success
N
Return (false)
Y
805
Contacts == 0
Y
806
Achieve connection
807
Install external dispatcher
N
808
Install external dispatcher
809
Connect request
Return (true)

Fig. 8

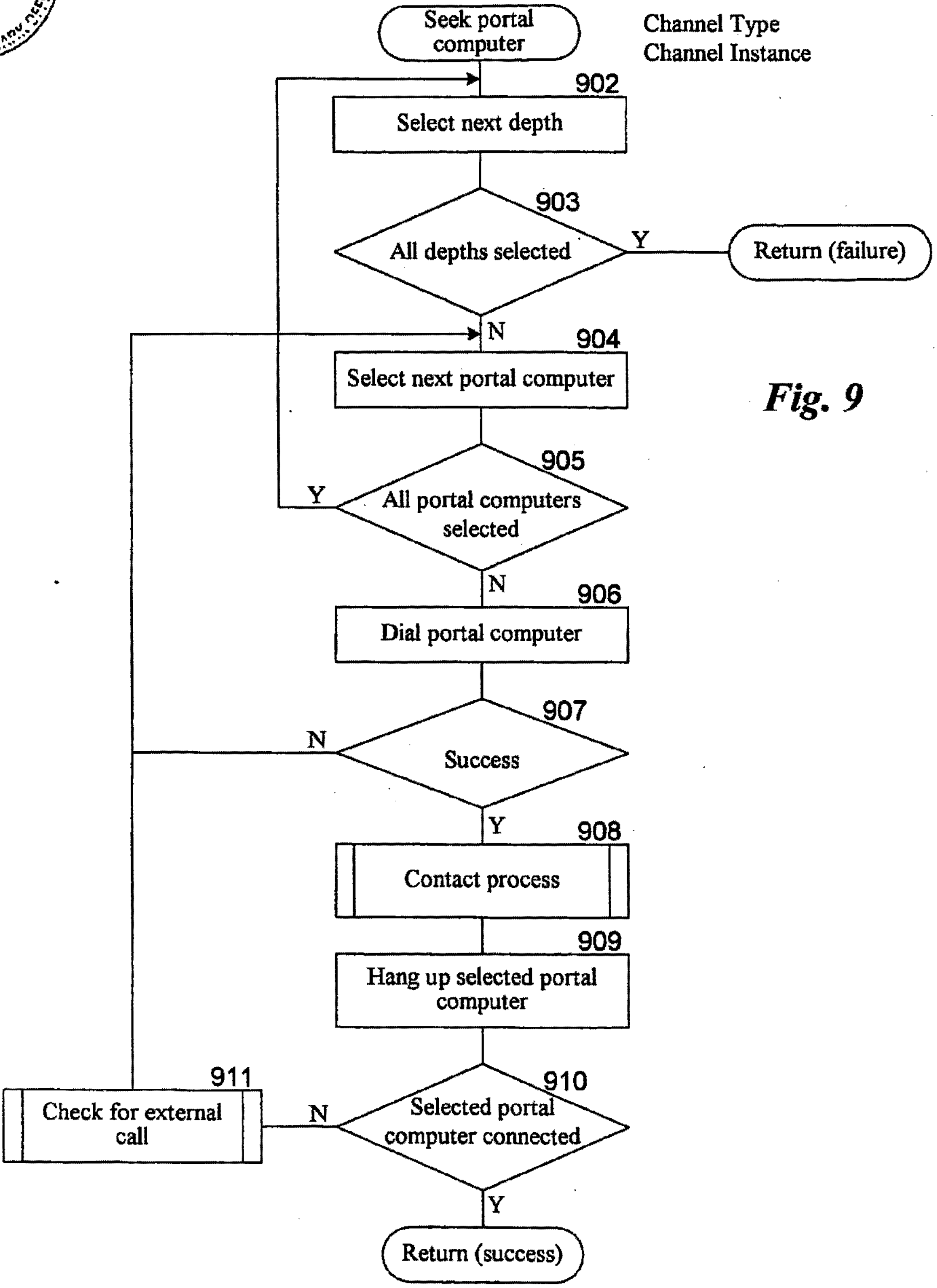

Seek portal computer
Channel Type
Channel Instance
902
Select next depth
903
All depths selected
Y
Return (failure)
N
904
Select next portal computer
905
Y
All portal computers selected
N
906
Dial portal computer
907
N
Success
Y
908
Contact process
909
Hang up selected portal computer
911
Check for external call
N
910
Selected portal computer connected
Y
Return (success)

Fig. 9

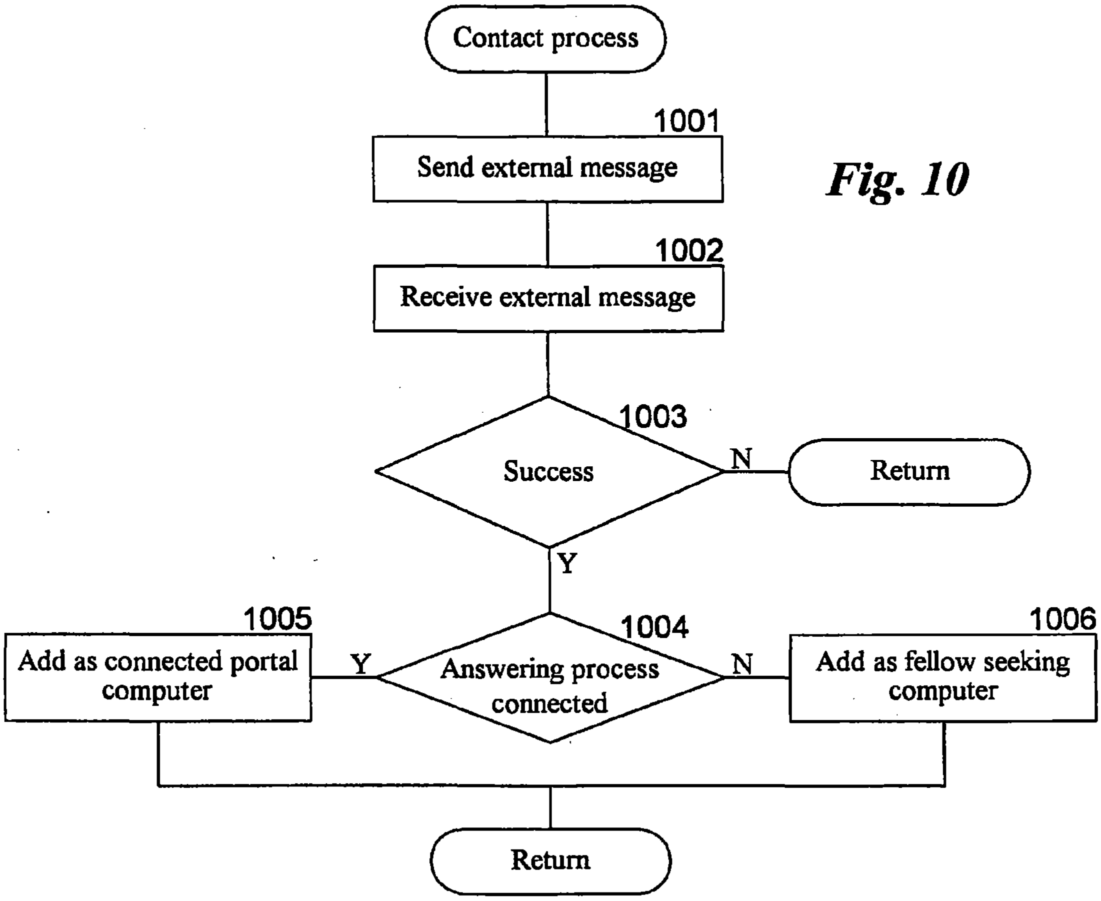
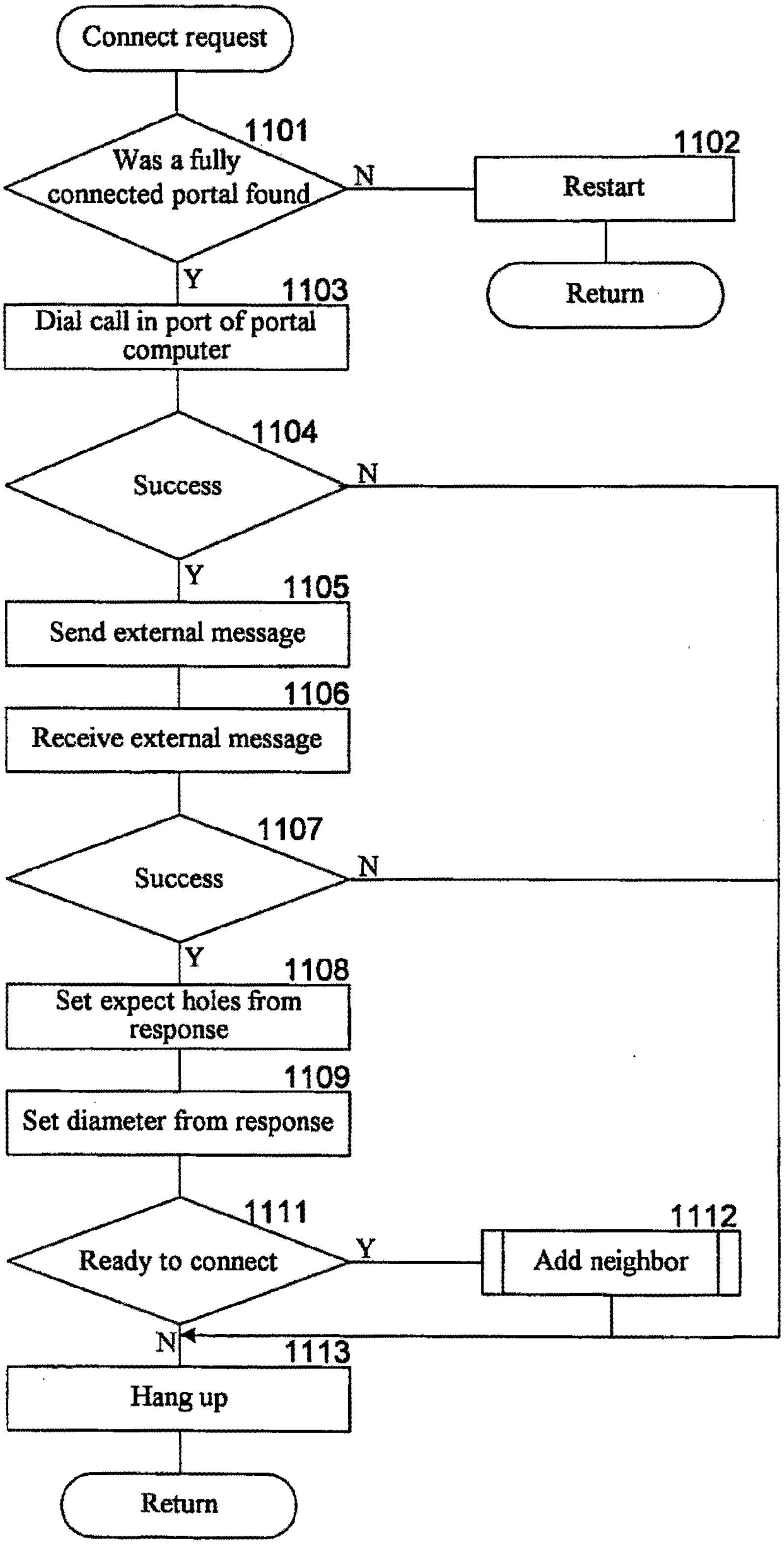

Contact process
1001
Send external message
1002
Receive external message
1003
Success
N
Return
Y
1005
Add as connected portal computer
Y
1004
Answering process connected
N
1006
Add as fellow seeking computer
Return

Fig. 10

Fig. 11


Connect request
1101
Was a fully connected portal found
N
1102
Restart
Return
Y
1103
Dial call in port of portal computer
1104
Success
N
Y
1105
Send external message
1106
Receive external message
1107
Success
N
Y
1108
Set expect holes from response
1109
Set diameter from response
1111
Ready to connect
Y
1112
Add neighbor
N
1113
Hang up
Return

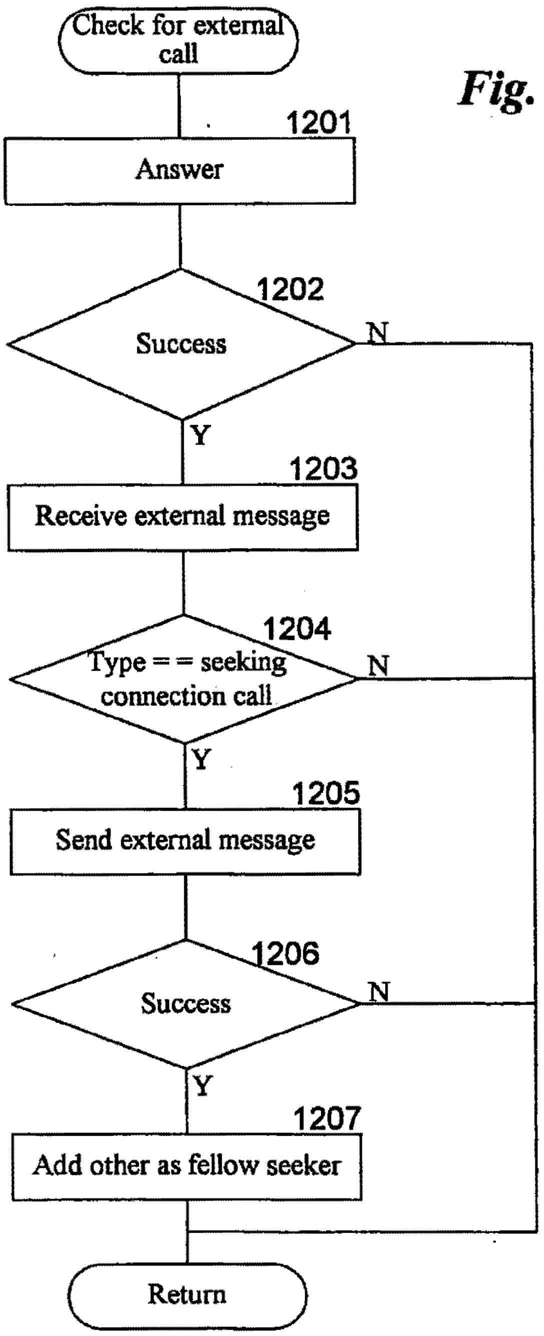

Check for external call
1201
Answer
1202
Success
N
Y
1203
Receive external message
1204
Type = = seeking connection call
N
Y
1205
Send external message
1206
Success
N
Y
1207
Add other as fellow seeker
Return

Fig. 12

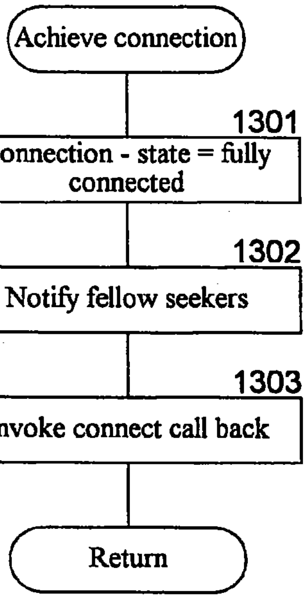

Achieve connection
1301
Connection - state = fully connected
1302
Notify fellow seekers
1303
Invoke connect call back
Return

Fig. 13

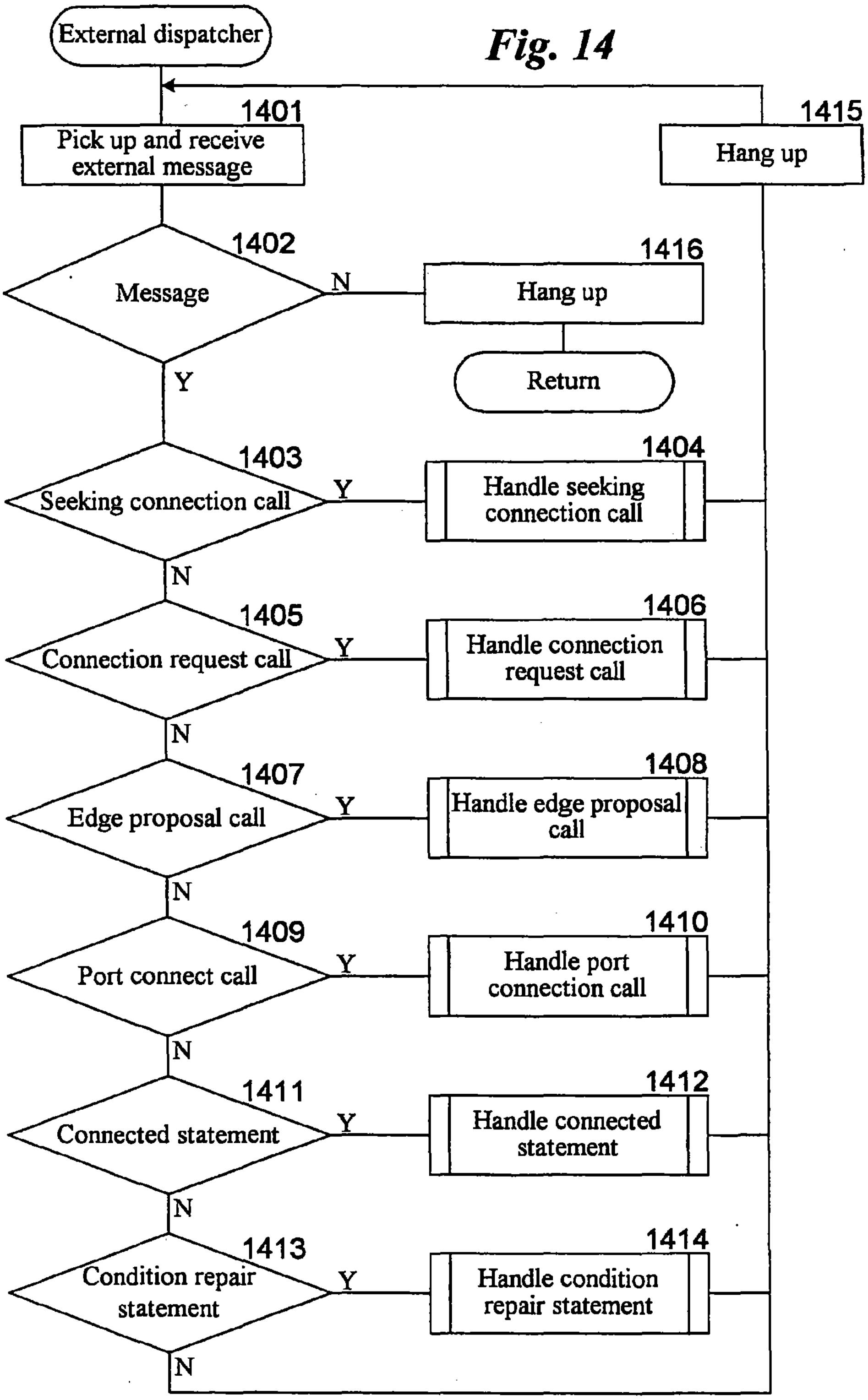

External dispatcher
1401
Pick up and receive external message
1415
Hang up
1402
Message
N
1416
Hang up
Return
Y
1403
Seeking connection call
Y
1404
Handle seeking connection call
N
1405
Connection request call
Y
1406
Handle connection request call
N
1407
Edge proposal call
Y
1408
Handle edge proposal call
N
1409
Port connect call
Y
1410
Handle port connection call
N
1411
Connected statement
Y
1412
Handle connected statement
N
1413
Condition repair statement
Y
1414
Handle condition repair statement
N

Fig. 14

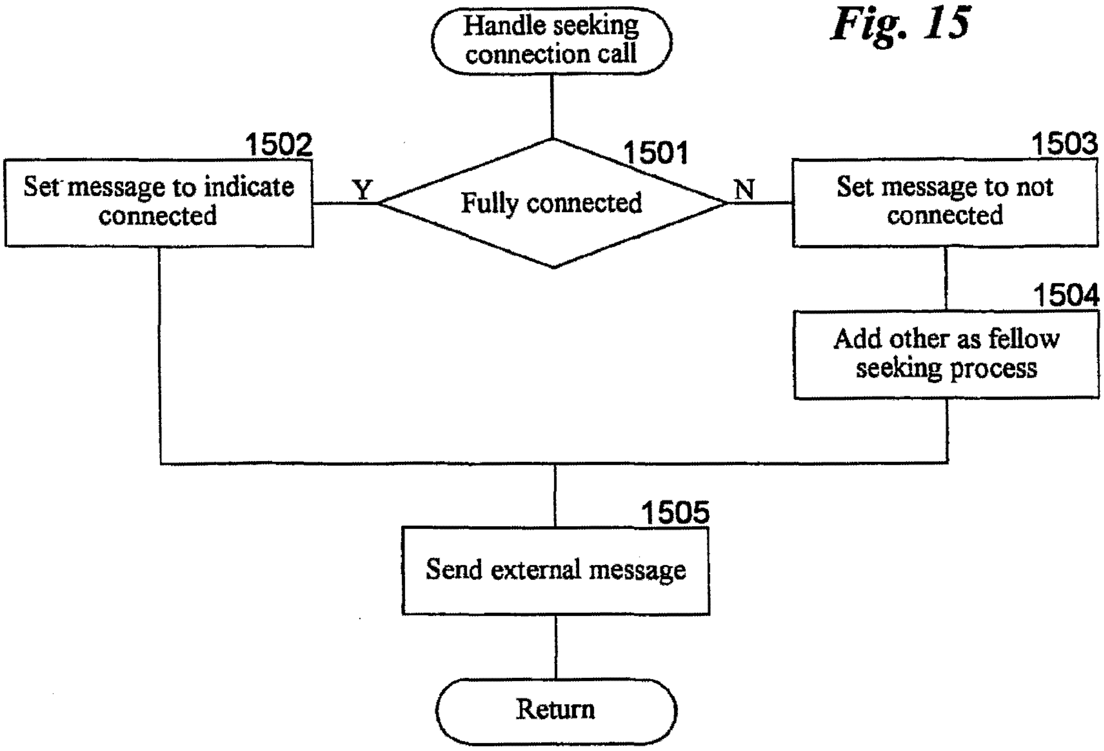

Handle seeking connection call
1501
Fully connected
Y
N
1502
Set message to indicate connected
1503
Set message to not connected
1504
Add other as fellow seeking process
1505
Send external message
Return

Fig. 15

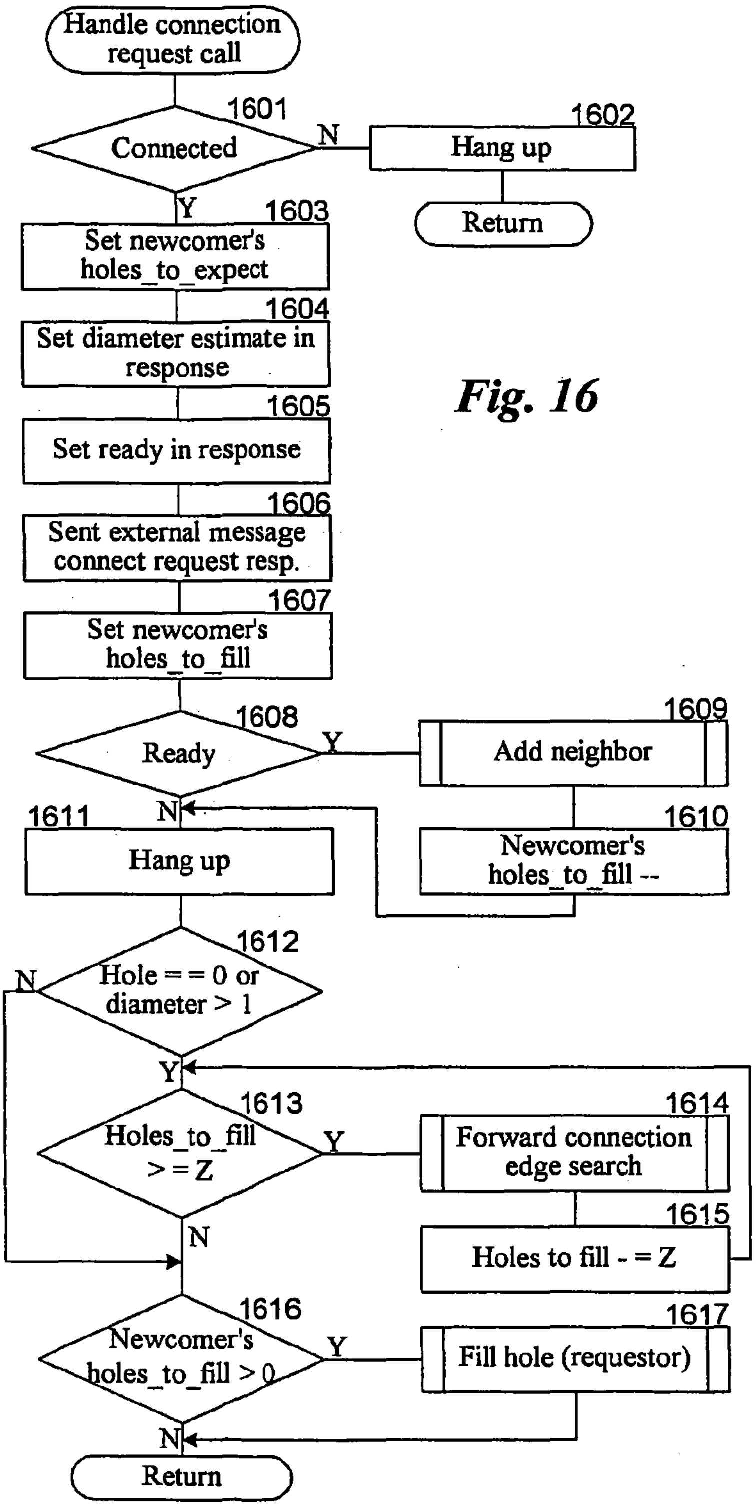

Handle connection request call
1601
Connected
N
1602
Hang up
Return
Y
1603
Set newcomer's holes_to_expect
1604
Set diameter estimate in response
1605
Set ready in response
1606
Sent external message connect request resp.
1607
Set newcomer's holes_to_fill
1608
Ready
Y
1609
Add neighbor
N
1611
Hang up
1610
Newcomer's holes_to_fill --
1612
Hole = = 0 or diameter > 1
N
Y
1613
Holes_to_fill > = Z
Y
1614
Forward connection edge search
N
1615
Holes to fill - = Z
1616
Newcomer's holes_to_fill > 0
Y
1617
Fill hole (requestor)
N
Return

Fig. 16

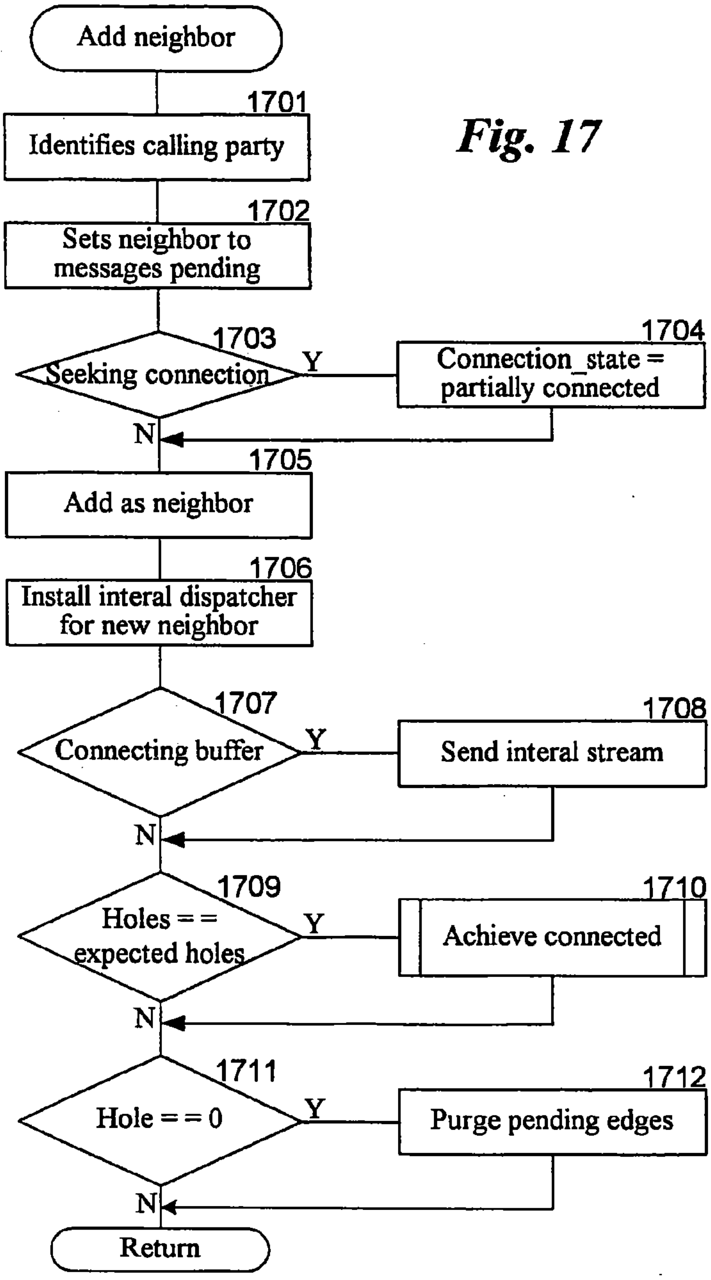
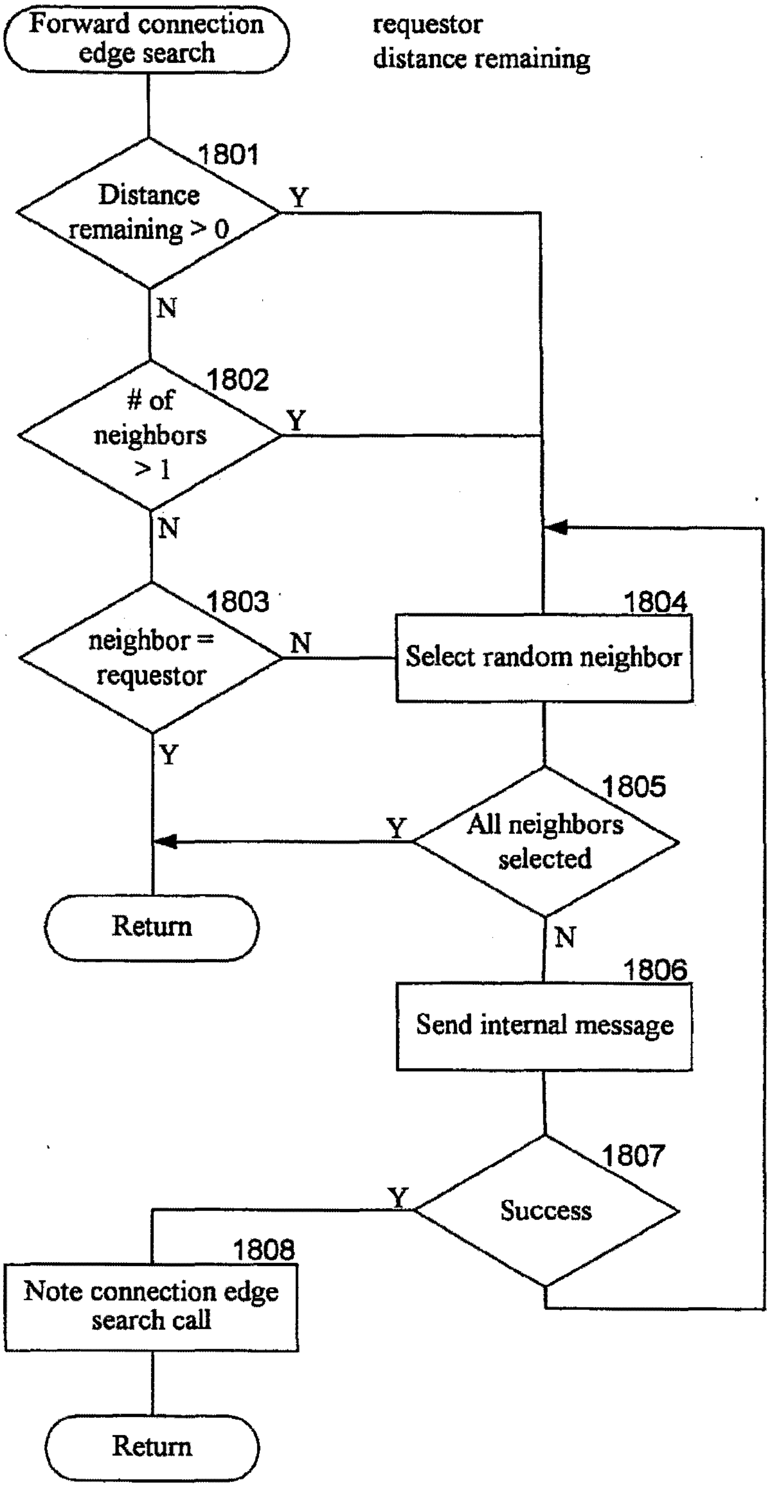

Add neighbor
1701
Identifies calling party
1702
Sets neighbor to messages pending
1703
Seeking connection
Y
1704
Connection_state = partially connected
N
1705
Add as neighbor
1706
Install interal dispatcher for new neighbor
1707
Connecting buffer
Y
1708
Send interal stream
N
1709
Holes = = expected holes
Y
1710
Achieve connected
N
1711
Hole = = 0
Y
1712
Purge pending edges
N
Return

Fig. 17

Fig. 18


Forward connection edge search
requestor
distance remaining
1801
Distance remaining > 0
Y
N
1802
# of neighbors > 1
Y
N
1803
neighbor = requestor
N
Y
1804
Select random neighbor
1805
All neighbors selected
Y
N
Return
1806
Send internal message
1807
Success
Y
1808
Note connection edge search call
Return

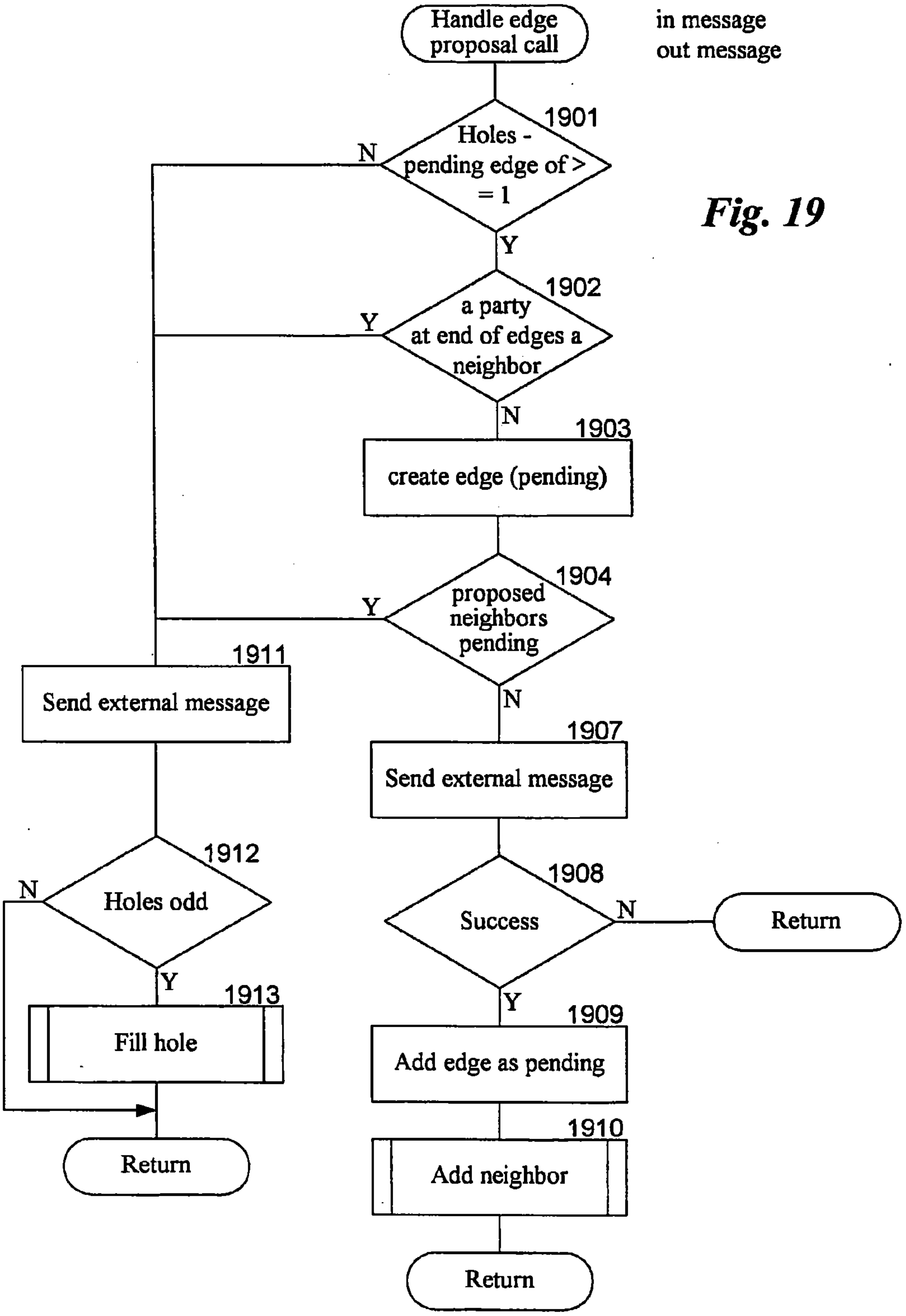

Handle edge proposal call
in message
out message
1901
Holes - pending edge of > = 1
N
Y
1902
a party at end of edges a neighbor
Y
N
1903
create edge (pending)
1904
proposed neighbors pending
Y
N
1911
Send external message
1912
Holes odd
N
Y
1913
Fill hole
Return
1907
Send external message
1908
Success
N
Return
Y
1909
Add edge as pending
1910
Add neighbor
Return

Fig. 19

Fig. 20

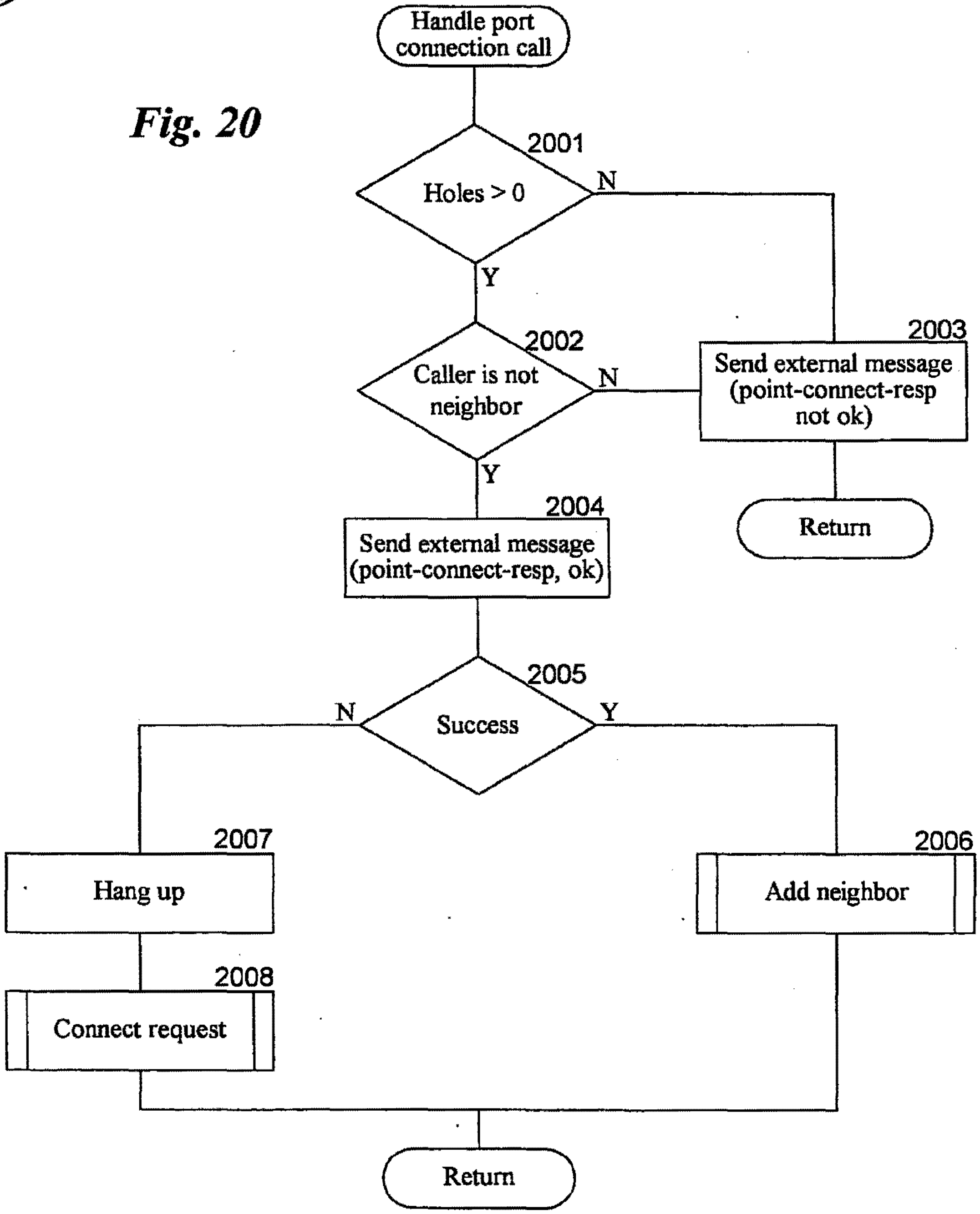

Handle port connection call
2001
Holes > 0
N
Y
2002
Caller is not neighbor
N
Y
2003
Send external message (point-connect-resp not ok)
Return
2004
Send external message (point-connect-resp, ok)
2005
Success
N
Y
2007
Hang up
2008
Connect request
2006
Add neighbor
Return

Fig. 21

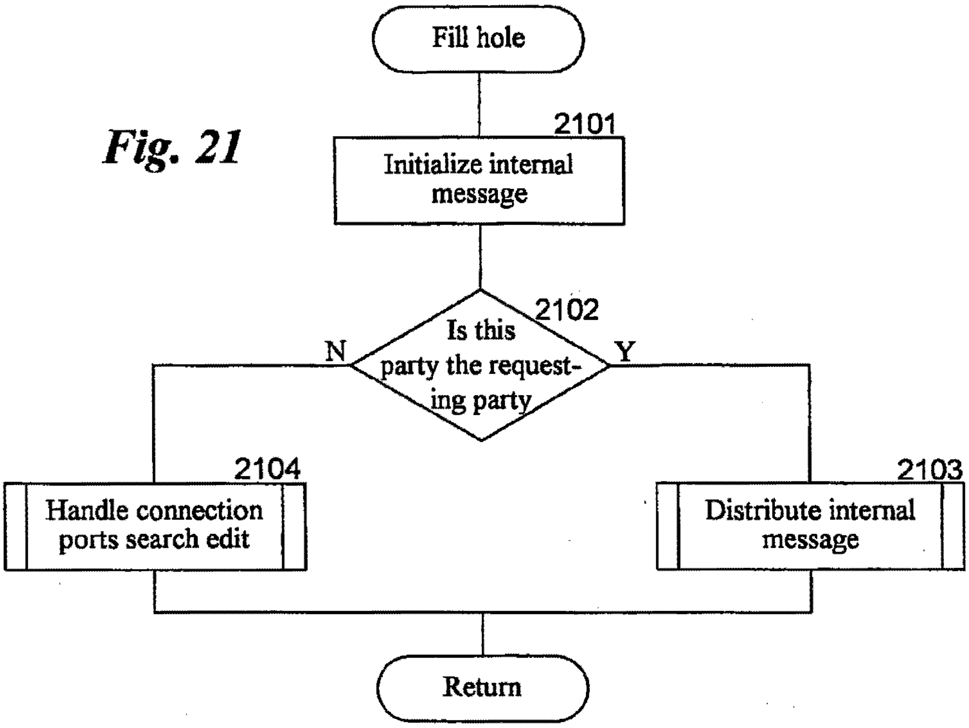

Fill hole
2101
Initialize internal message
2102
Is this party the requesting party
N
Y
2104
Handle connection ports search edit
2103
Distribute internal message
Return

Fig. 22

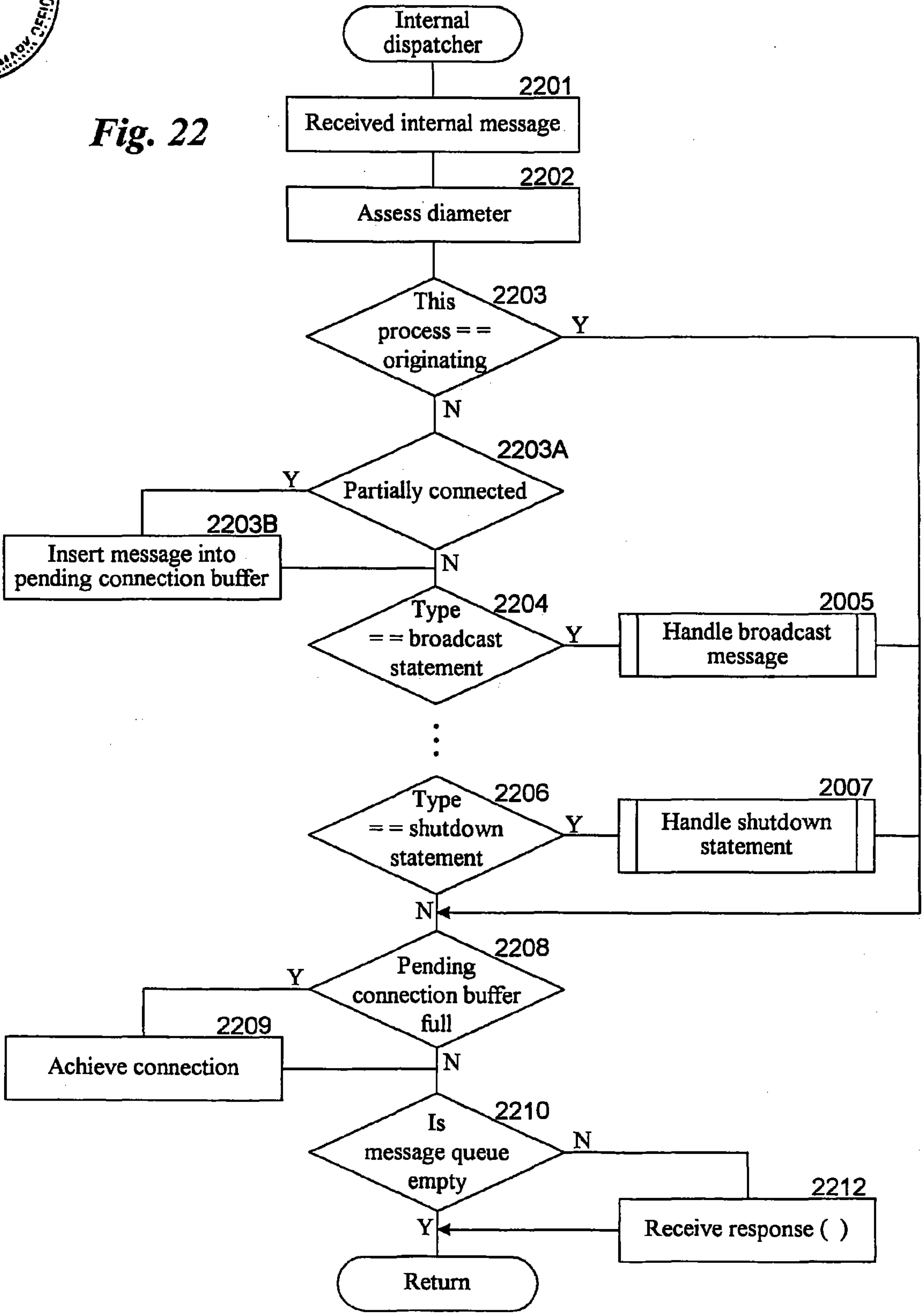

Internal dispatcher
2201
Received internal message
2202
Assess diameter
2203
This process = = originating
Y
N
2203A
Partially connected
Y
2203B
Insert message into pending connection buffer
N
2204
Type = = broadcast statement
Y
2005
Handle broadcast message
2206
Type = = shutdown statement
Y
2007
Handle shutdown statement
N
2208
Pending connection buffer full
Y
2209
Achieve connection
N
2210
Is message queue empty
N
2212
Receive response ( )
Y
Return

Fig. 23

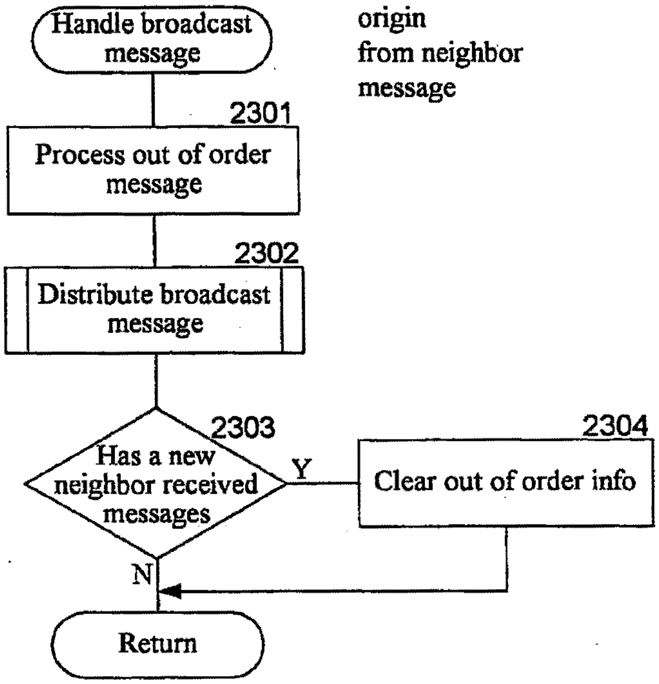
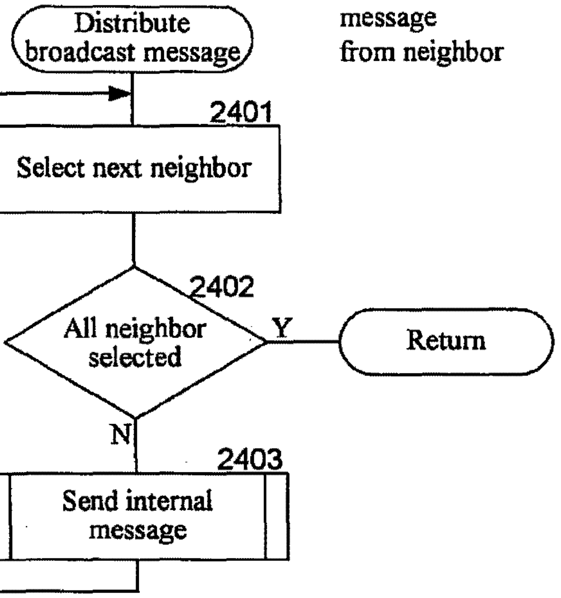

Handle broadcast message
origin
from neighbor
message
2301
Process out of order message
2302
Distribute broadcast message
2303
Has a new neighbor received messages
Y
N
2304
Clear out of order info
Return

Fig. 24


Distribute broadcast message
message from neighbor
2401
Select next neighbor
2402
All neighbor selected
Y
Return
N
2403
Send internal message

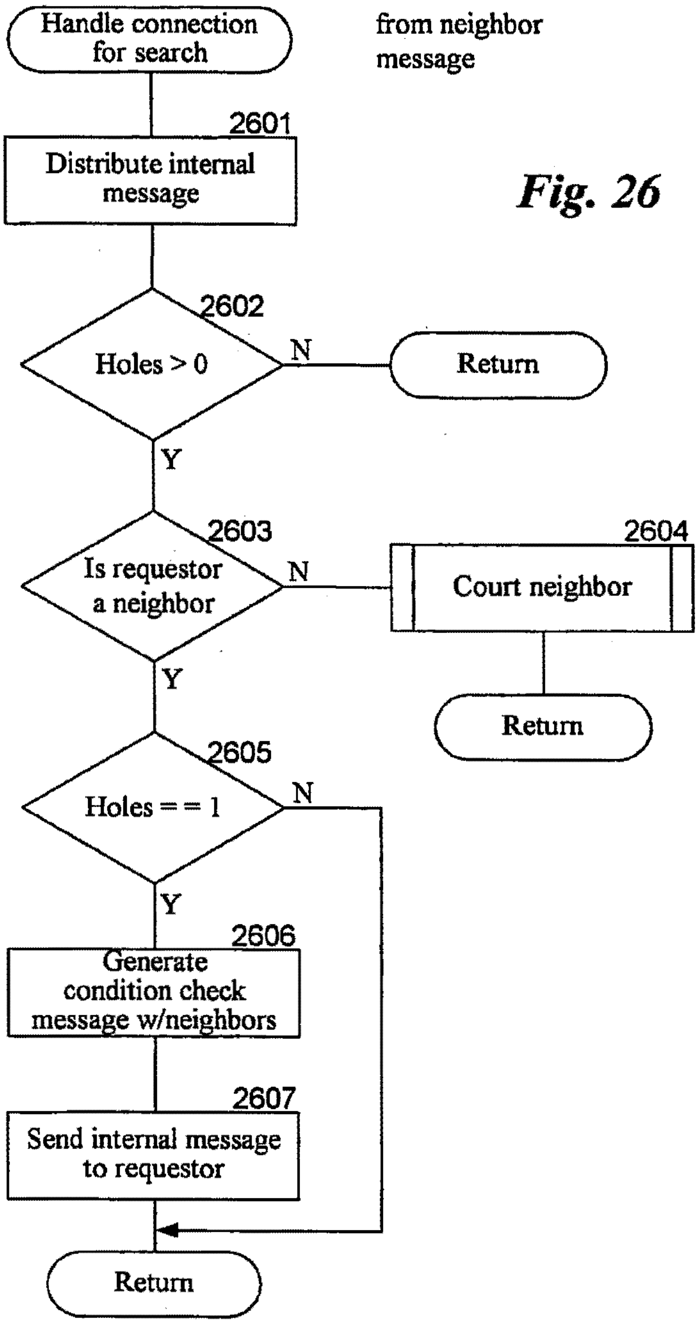

Handle connection for search
from neighbor message
2601
Distribute internal message
2602
Holes > 0
N
Return
Y
2603
Is requestor a neighbor
N
2604
Court neighbor
Return
Y
2605
Holes = = 1
N
Y
2606
Generate condition check message w/neighbors
2607
Send internal message to requestor
Return

Fig. 26

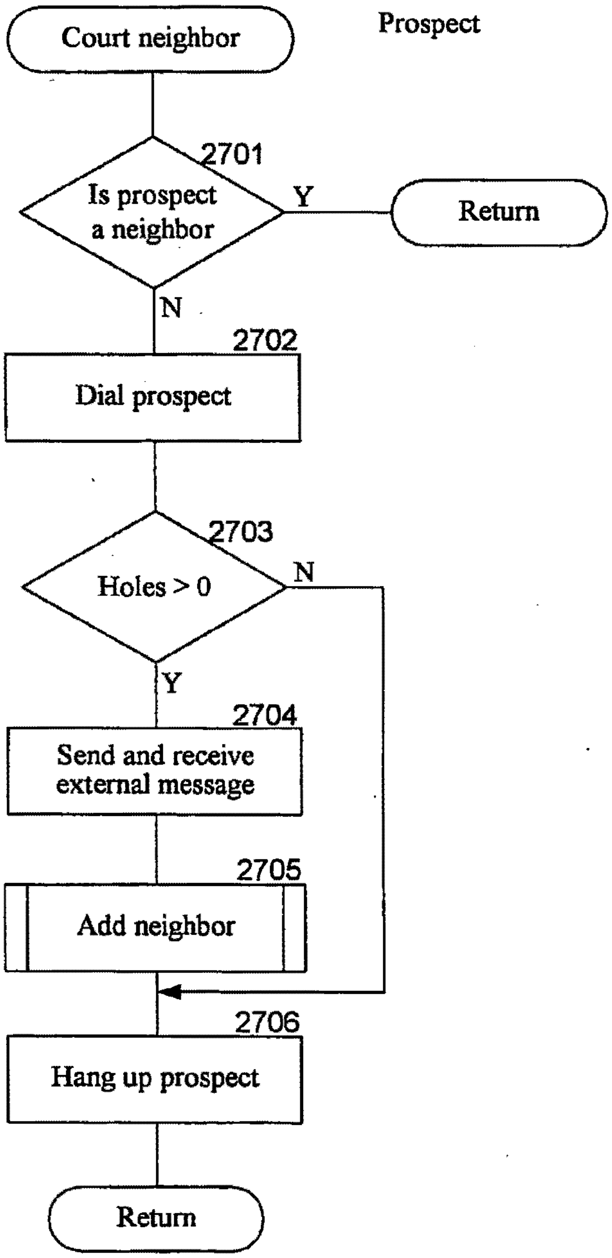

Court neighbor
Prospect
2701
Is prospect a neighbor
Y
Return
N
2702
Dial prospect
2703
Holes > 0
N
Y
2704
Send and receive external message
2705
Add neighbor
2706
Hang up prospect
Return

Fig. 27

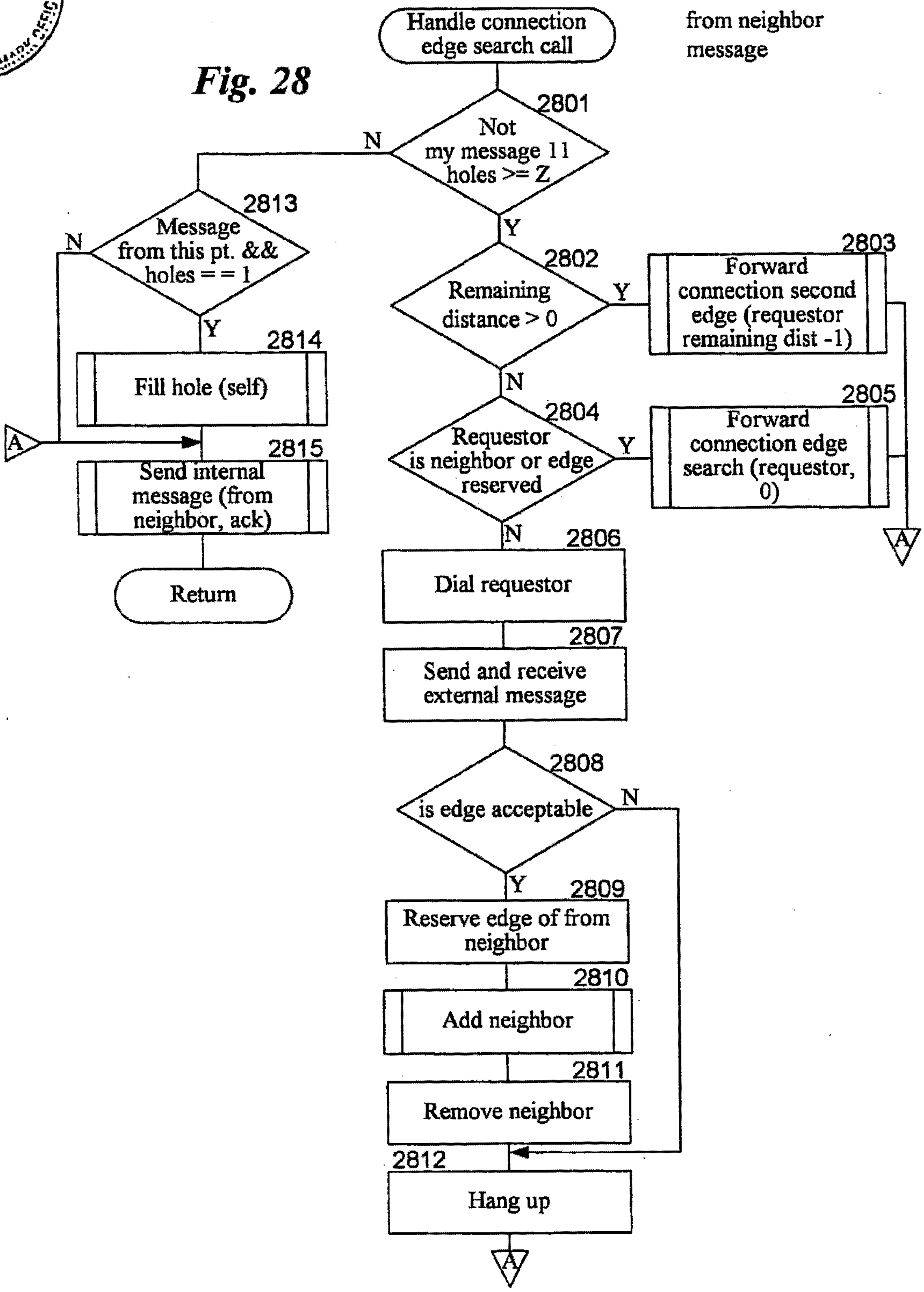

Handle connection edge search call
from neighbor message
2801
Not my message 11 holes >= Z
N
Y
2802
Remaining distance > 0
Y
2803
Forward connection second edge (requestor remaining dist -1)
N
2804
Requestor is neighbor or edge reserved
Y
2805
Forward connection edge search (requestor, 0)
A
N
2806
Dial requestor
2807
Send and receive external message
2808
is edge acceptable
N
Y
2809
Reserve edge of from neighbor
2810
Add neighbor
2811
Remove neighbor
2812
Hang up
A
2813
Message from this pt. && holes = = 1
N
Y
2814
Fill hole (self)
A
2815
Send internal message (from neighbor, ack)
Return

Fig. 28

Fig. 29

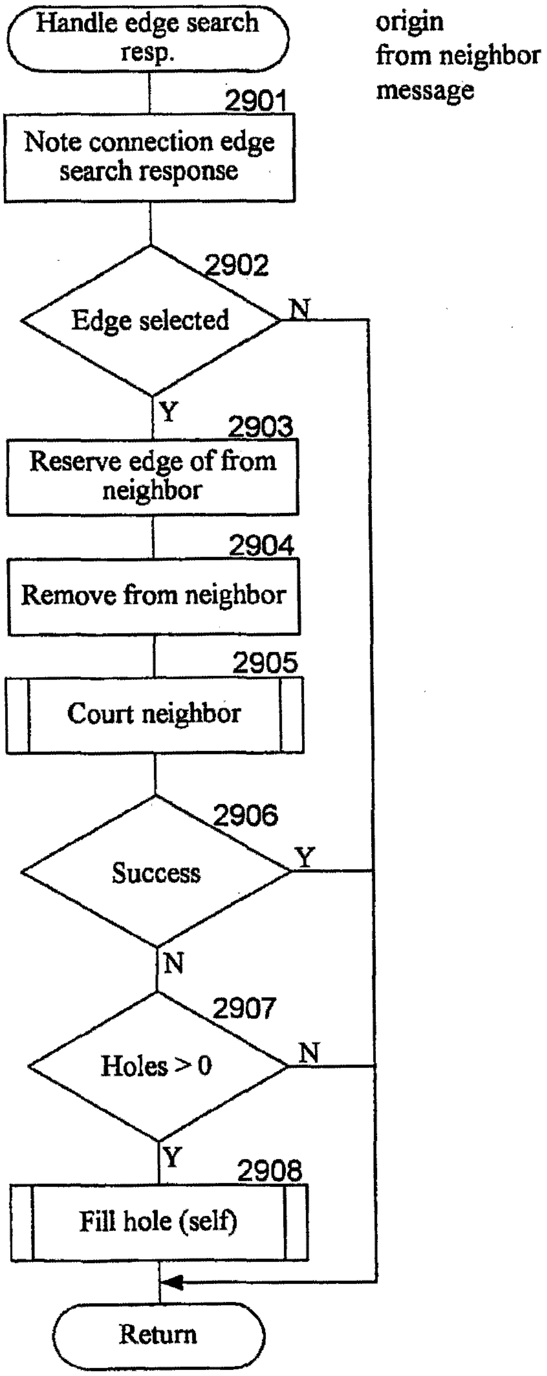

Handle edge search resp.
origin
from neighbor
message
2901
Note connection edge search response
2902
Edge selected
N
Y
2903
Reserve edge of from neighbor
2904
Remove from neighbor
2905
Court neighbor
2906
Success
Y
N
2907
Holes > 0
N
Y
2908
Fill hole (self)
Return

Fig. 30

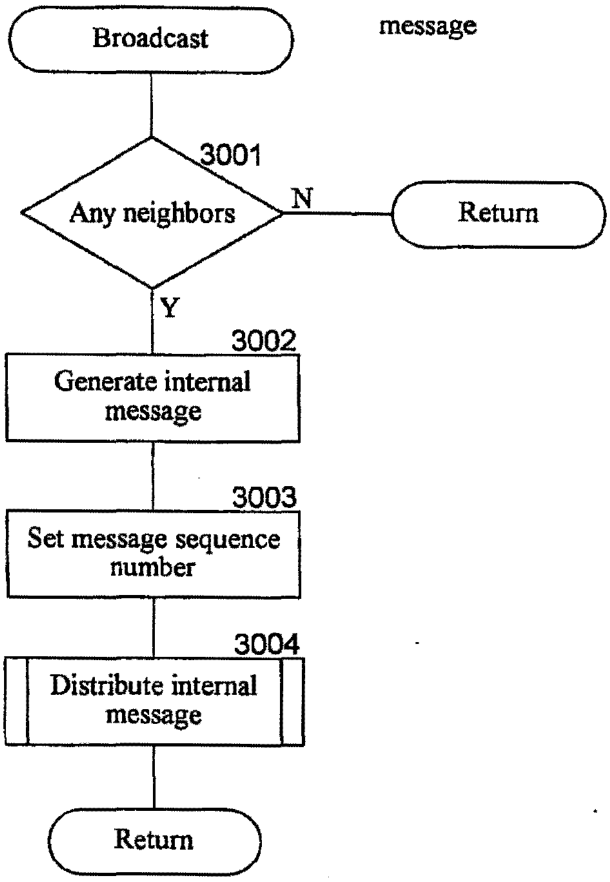

Broadcast
message
3001
Any neighbors
N
Return
Y
3002
Generate internal message
3003
Set message sequence number
3004
Distribute internal message
Return

Fig. 31

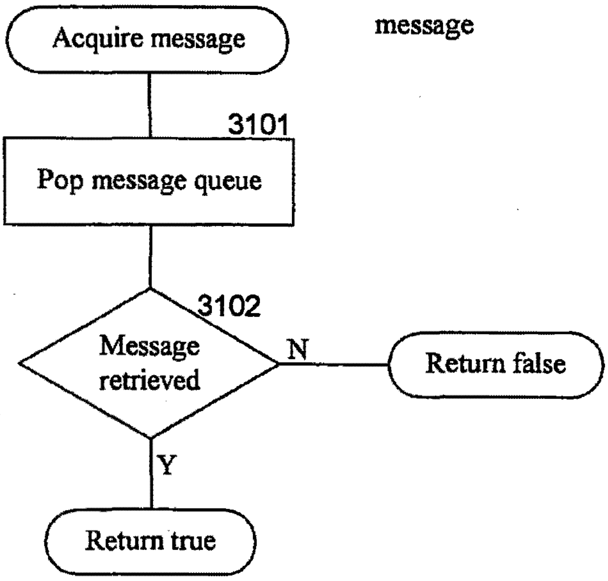

Acquire message
message
3101
Pop message queue
3102
Message retrieved
N
Return false
Y
Return true

Fig. 32

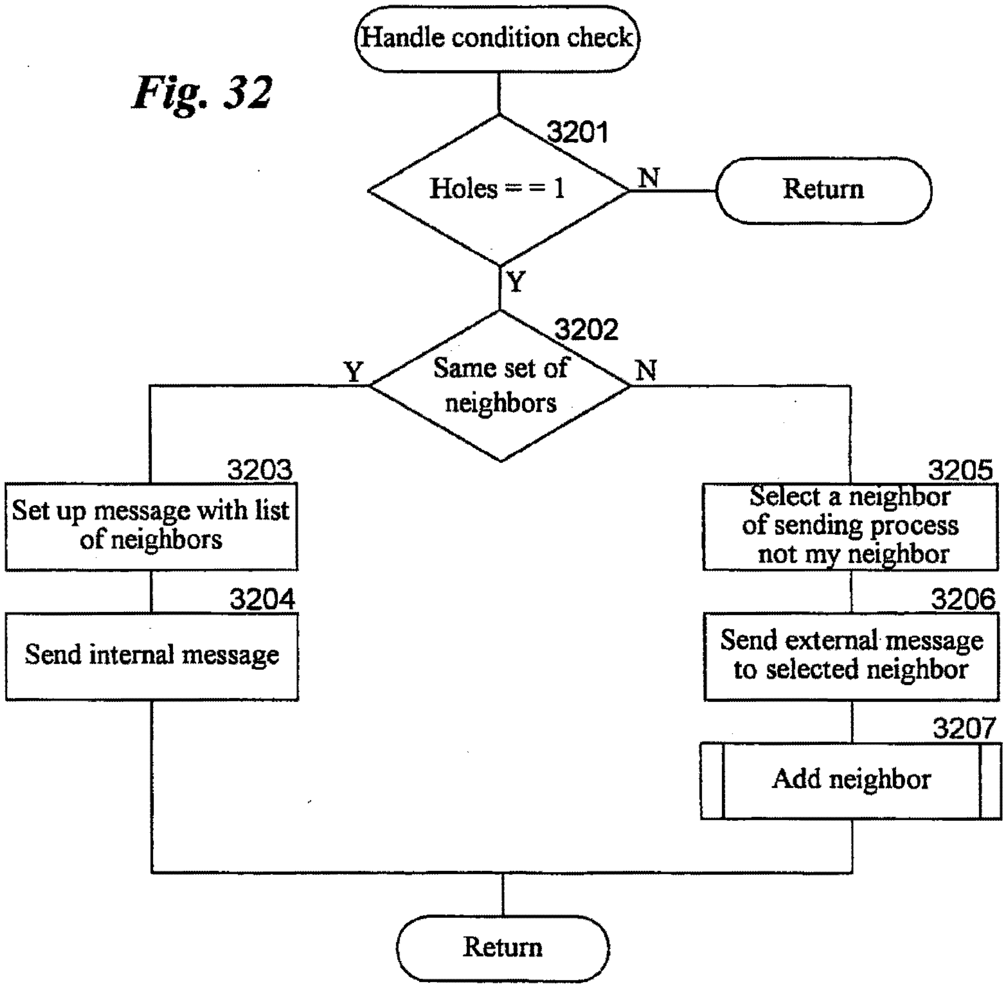
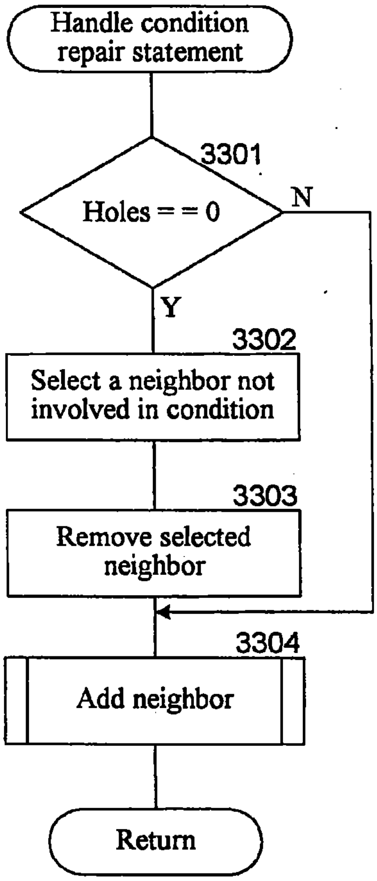

Handle condition check
3201
Holes = = 1
N
Return
Y
3202
Same set of neighbors
Y
N
3203
Set up message with list of neighbors
3204
Send internal message
3205
Select a neighbor of sending process not my neighbor
3206
Send external message to selected neighbor
3207
Add neighbor
Return

Fig. 33


Handle condition repair statement
3301
Holes = = 0
N
Y
3302
Select a neighbor not involved in condition
3303
Remove selected neighbor
3304
Add neighbor
Return

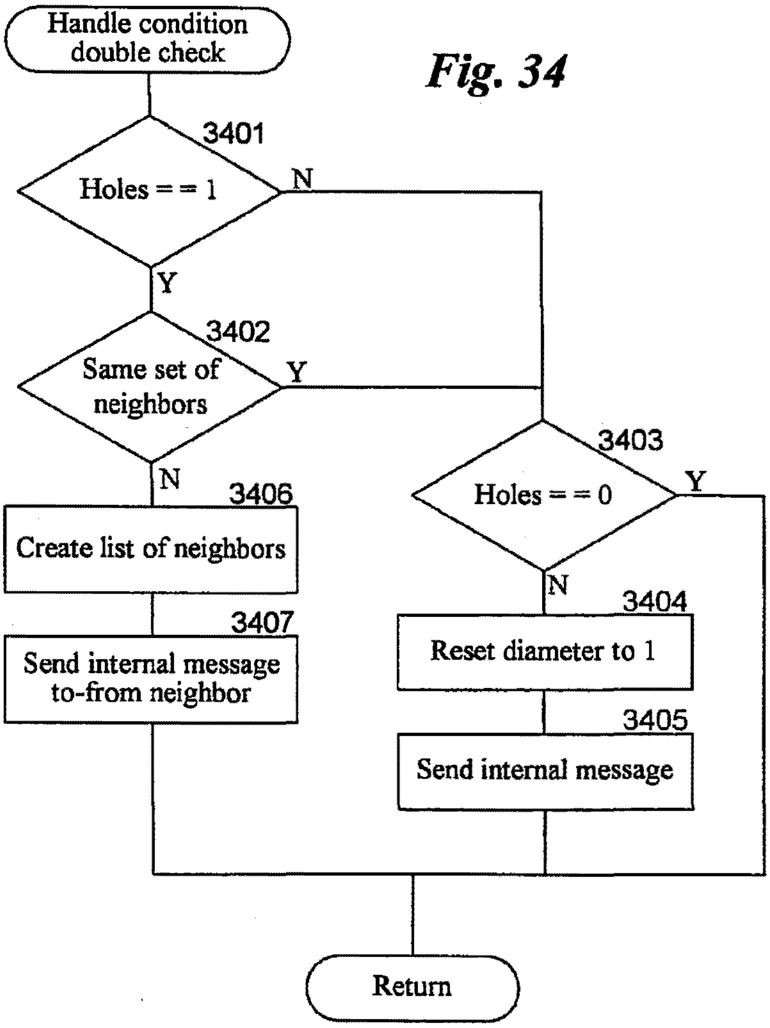

Handle condition double check
3401
Holes = = 1
N
Y
3402
Same set of neighbors
Y
N
3403
Holes = = 0
Y
N
3406
Create list of neighbors
3407
Send internal message to-from neighbor
3404
Reset diameter to 1
3405
Send internal message
Return

Fig. 34

35

Express Mail No. EV343595835US

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: Mail

Mail Stop ISSUE FEE
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450

3-22-04

or Fax (703) 746-4000

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 4 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Legibly mark-up with any corrections or use Block 1)

25096 7590 01/27/2004

PERKINS COIE LLP
PATENT-SEA
P.O. BOX 1247
SEATTLE, WA 98111-1247

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission

I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO, on the date indicated below.

Melody J. Almberg	(Depositor's name)
Melody J. Almberg	(Signature)
March 19, 2004	(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/629,577	07/31/2000	Fred B. Holt	030048003US	4317

TITLE OF INVENTION: LEAVING A BROADCAST CHANNEL

APPLN. TYPE	SMALL ENTITY	ISSUE FEE	PUBLICATION FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	NO	$1330	$0	$1330	04/27/2004

EXAMINER	ART UNIT	CLASS-SUBCLASS
LAZARO, DAVID R	2155	709-204000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).

□ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.

☒ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. **Use of a Customer Number is required.**

2. For printing on the patent front page, list (1) the names of up to 3 registered patent attorneys or agents OR, alternatively, (2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.

1 Perkins Coie LLP
2 ____________
3 ____________

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. Inclusion of assignee data is only appropriate when an assignment has been previously submitted to the USPTO or is being submitted under separate cover. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE	(B) RESIDENCE: (CITY and STATE OR COUNTRY)
The Boeing Company	Seattle, Washington

Please check the appropriate assignee category or categories (will not be printed on the patent); □ individual ☒ corporation or other private group entity □ government

4a. The following fee(s) are enclosed:

☒ Issue Fee
□ Publication Fee
☒ Advance Order - # of Copies 1

4b. Payment of Fee(s):

☒ A check in the amount of the fee(s) is enclosed.
□ Payment by credit card. Form PTO-2038 is attached.
☒ The Director is hereby authorized by charge ~~the required~~ any additional fee(s), or credit any overpayment, to Deposit Account Number 50-0665 (enclose an extra copy of this form).

Director for Patents is requested to apply the Issue Fee and Publication Fee (if any) or to re-apply any previously paid issue fee to the application identified above.

(Authorized Signature) (Date) 3/19/04

NOTE; The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office.

This collection of information is required by 37 CFR 1.311. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, Alexandria, Virginia 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, Alexandria, Virginia 22313-1450.

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

03/23/2004 HBERHE1 00000090 09629577
01 FC:1501 1330.00 OP
02 FC:8001 3.00 OP

TRANSMIT THIS FORM WITH FEE(S)

C of C

Attorney Docket No.: 030048003US

I hereby certify that this correspondence is being deposited with the U.S. Postal Service with sufficient postage as First Class Mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on:

Date: May 26, 2004 By: ______________________
Melody J. Almberg

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

IN RE APPLICATION OF: FRED B. HOLT *et al.*
PATENT NO.: 6,732,147
ISSUED: MAY 4, 2004
FOR: LEAVING A BROADCAST CHANNEL

EXAMINER: DAVID LAZARO
ART UNIT: 2155
CONF. NO.: 4317

Request for Certificate of Correction under 37 C.F.R. §1.322 or §1.323

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

1. The applicant(s) requests a Certificate of Correction to correct errors in the above-identified patent, which are listed on the enclosed Form PTO/SB/44.

2. Any errors on the part of the applicant are of a clerical or typographical nature or are otherwise minor in character. None of the requested corrections would constitute new matter or require reexamination of the patent.

3. Source of Error(s) and Payment of Fee:

☒ All of the errors listed on Form PTO/SB/44 are believed to be due to mistake on the part of the USPTO (37 C.F.R. §1.322). Accordingly, no fees are believed to be due.

☐ At least one of the errors occurred due to applicant's mistake made in good faith (37 C.F.R. §1.323).

☐ A check covering the fee under 37 C.F.R. §1.20(a) ($100.00) is enclosed herewith.

☐ Please charge the fee under 37 C.F.R. §1.20(a) to Deposit Account No. 50-0665. This paper is provided in triplicate.

☒ Please charge any underpayment necessary for consideration of this paper to Deposit Account No. 50-0665.

4. Please send the Certificate of Correction to the undersigned at the address shown below.

Respectfully submitted,
Perkins Coie LLP

Date: 5/26/04

Chun Ng
Registration No. 36,878

Correspondence Address:
Customer No. 25096
Perkins Coie LLP
P.O. Box 1247
Seattle, Washington 98111-1247
(206) 359-8000

PTO/SB/ 44 (02-01)
Approved for use through 01/31/2004. OMB 0651-0033
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.
(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO : 6,732,147 B1

DATED : May 4, 2004

INVENTOR(S) : Fred B. Holt

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 5
Line 9, "a-broadcast" should be --a broadcast--;

Column 6
Line 30, "on-that" should be --on that--;

Column 8
Line 26, delete comma between "newly";

Column 11
Line 60, "port-number" should be --port number--;
Line 63, "port-order" should be --port order--;

Column 13
Line 50, "computer-cannot" should be --computer cannot--;

Column 14
Line 51, delete period after "Regular";

Column 22
Line 41, delete "is" between "receives" and "through";

Column 23
Line 23, delete "is" between "In" and "block";

Column 25
Line 45, insert comma between "2605" and "else";
Line 46, delete comma between "2604" and "In";

MAILING ADDRESS OF SENDER:

Perkins Coie LLP
PATENT-SEA
PO Box 1247
Seattle, WA 98111-1247

PATENT NO. 6,732,147

No. of additional copies

⇨ 1

Burden Hour Statement: This form is estimated to take 1.0 hour to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, Washington, DC 20231.

PTO/SB/ 44 (02-01)
Approved for use through 01/31/2004. OMB 0651-0033
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.
(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO : 6,732,147 B1

DATED : May 4, 2004

INVENTOR(S) : Fred B. Holt

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

<u>Column 5</u>
Line 9, "a-broadcast" should be --a broadcast--;

<u>Column 6</u>
Line 30, "on-that" should be --on that--;

<u>Column 8</u>
Line 26, delete comma between "newly";

<u>Column 11</u>
Line 60, "port-number" should be --port number--;
Line 63, "port-order" should be --port order--;

<u>Column 13</u>
Line 50, "computer-cannot" should be --computer cannot--;

<u>Column 14</u>
Line 51, delete period after "Regular";

<u>Column 22</u>
Line 41, delete "is" between "receives" and "through";

<u>Column 23</u>
Line 23, delete "is" between "In" and "block";

<u>Column 25</u>
Line 45, insert comma between "2605" and "else";
Line 46, delete comma between "2604" and "In";

MAILING ADDRESS OF SENDER:

Perkins Coie LLP
PATENT-SEA
PO Box 1247
Seattle, WA 98111-1247

PATENT NO. 6,732,147

No. of additional copies

⇒ 1

Burden Hour Statement: This form is estimated to take 1.0 hour to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, Washington, DC 20231.

PTO/SB/ 44 (02-01)
Approved for use through 01/31/2004. OMB 0651-0033
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.
(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO : 6,732,147 B1

DATED : May 4, 2004

INVENTOR(S) : Fred B. Holt

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 5
Line 9, "a-broadcast" should be --a broadcast--;

Column 6
Line 30, "on-that" should be --on that--;

Column 8
Line 26, delete comma between "newly";

Column 11
Line 60, "port-number" should be --port number--;
Line 63, "port-order" should be --port order--;

Column 13
Line 50, "computer-cannot" should be --computer cannot--;

Column 14
Line 51, delete period after "Regular";

Column 22
Line 41, delete "is" between "receives" and "through";

Column 23
Line 23, delete "is" between "In" and "block";

Column 25
Line 45, insert comma between "2605" and "else";
Line 46, delete comma between "2604" and "In";

MAILING ADDRESS OF SENDER:

Perkins Coie LLP
PATENT-SEA
PO Box 1247
Seattle, WA 98111-1247

PATENT NO. 6,732,147

No. of additional copies

⇨ 1

Burden Hour Statement: This form is estimated to take 1.0 hour to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, Washington, DC 20231.

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO.	: 6,732,147 B1
DATED	: May 4, 2004
INVENTOR(S)	: Fred B. Holt

Page 1 of 2

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 5,
Line 9, "a-broadcast" should be -- a broadcast --;

Column 6,
Line 30, "on-that" should be -- on that --;

Column 8,
Line 26, delete comma between "newly";

Column 11,
Line 60, "port-number" should be -- port number --;
Line 63, "port-order" should be -- port order --;

Column 13,
Line 50, "computer-cannot" should be -- computer cannot --;

Column 14,
Line 51, delete period after "Regular";

Column 22,
Line 41, delete "is" between "receives" and "through";

Column 23,
Line 23, delete "is" between "In" and "block";

Column 25,
Line 45, insert comma between "2605" and "else";

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,732,147 B1
DATED : May 4, 2004
INVENTOR(S) : Fred B. Holt

Page 2 of 2

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 25 (cont'd),
Line 46, delete comma between "2604" and "In";

Signed and Sealed this

Twenty-seventh Day of July, 2004

Jon W. Dudas

JON W. DUDAS
Acting Director of the United States Patent and Trademark Office

US006732147B1

(12) **United States Patent**
Holt et al.

(10) **Patent No.: US 6,732,147 B1**
(45) **Date of Patent: May 4, 2004**

(54) **LEAVING A BROADCAST CHANNEL**

(75) Inventors: **Fred B. Holt**, Seattle, WA (US); **Virgil E. Bourassa**, Bellevue, WA (US)

(73) Assignee: **The Boeing Company**, Seattle, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 719 days.

(21) Appl. No.: **09/629,577**

(22) Filed: **Jul. 31, 2000**

(51) **Int. Cl.**^{7} **G06F 15/16**

(52) **U.S. Cl.** **709/204**; 709/227

(58) **Field of Search** 709/204, 227, 709/217

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,912,656	A	3/1990	Cain et al.
5,056,085	A	10/1991	Vu
5,309,437	A	5/1994	Perlman et al.
5,426,637	A	6/1995	Derby et al.
5,535,199	A	7/1996	Amri et al.
5,568,487	A	10/1996	Sitbon et al.
5,636,371	A	6/1997	Yu
5,673,265	A	9/1997	Gupta et al.
5,696,903	A	12/1997	Mahany
5,732,074	A	3/1998	Spaur et al.
5,732,219	A	3/1998	Blumer et al.
5,734,865	A	3/1998	Yu
5,737,526	A	4/1998	Periasamy et al.
5,754,830	A	5/1998	Butts et al.
5,761,425	A	6/1998	Miller
5,764,756	A	6/1998	Onweller
5,790,548	A	8/1998	Sistanizadeh et al.
5,790,553	A	8/1998	Deaton, Jr. et al.
5,799,016	A	8/1998	Onweller
5,802,285	A	9/1998	Hirviniemi
5,864,711	A	1/1999	Mairs et al.

(List continued on next page.)

OTHER PUBLICATIONS

Bondy et al. "Graph Theory With Applications" American Elsevier Publishing Co. Inc. pp. 47–50 Secion 3.3.*

Yavatkar et al. "A Reliable Dissemination Protocol for Interactive Collaborative Applications" Proc. ACM Multimedia, 1995 p.333–344 http:/citeseer.nj.nec.com/article/yavatkar95reliable.html.*

Alagar, S. and Venkatesan, S., "Reliable Broadcast in Mobile Wireless Networks," Department of Computer Science, University of Texas at Dallas, Military Communications Conference, 1995, MILCOM '95 Conference Record, IEEE San Diego, California, Nov. 5-8, 1995 (pp. 236–240).

International Search Report for The Boeing Company, International Patent Application No. PCT/US01/24240, Jun. 5, 2002 (7 pages).

U.S. patent application Ser. No. 09/629,570, Bourassa et al., filed Jul. 31, 2000.

U.S. patent application Ser. No. 09/629,576, Bourassa et al., filed Jul. 31, 2000.

U.S. patent application Ser. No. 09/629,575, Bourassa et al., filed Jul. 31, 2000.

U.S. patent application Ser. No.09/629,572, Bourassa et al., filed Jul. 31, 2000.

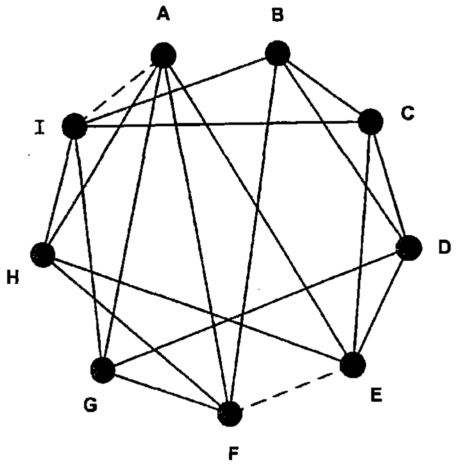
(List continued on next page.)

Primary Examiner—Patrice Winder
Assistant Examiner—David Lazaro
(74) *Attorney, Agent, or Firm*—Perkins Coie LLP

(57) **ABSTRACT**

A method for leaving a multicast computer network is disclosed. The method allows for the disconnection of a first computer from a second computer. When the first computer decides to disconnect from the second computer, the first computer sends a disconnect message to the second computer. Then, when the second computer receives the disconnect message from the first computer, the second computer broadcasts a connection port search message to find a third computer to which it can connect.

16 Claims, 39 Drawing Sheets



U.S. PATENT DOCUMENTS

5,867,660	A		2/1999	Schmidt et al.	
5,867,667	A		2/1999	Butman et al.	
5,870,605	A		2/1999	Bracho et al.	
5,874,960	A		2/1999	Mairs et al.	
5,899,980	A		5/1999	Wilf et al.	
5,907,610	A		5/1999	Onweller	
5,928,335	A		7/1999	Morita	
5,935,215	A		8/1999	Bell et al.	
5,946,316	A	*	8/1999	Chen et al.	370/408
5,948,054	A		9/1999	Nielsen	
5,949,975	A		9/1999	Batty et al.	
5,956,484	A		9/1999	Rosenberg et al.	
5,974,043	A		10/1999	Solomon	
5,987,506	A		11/1999	Carter et al.	
6,003,088	A		12/1999	Houston et al.	
6,013,107	A		1/2000	Blackshear et al.	
6,023,734	A		2/2000	Ratcliff et al.	
6,029,171	A		2/2000	Smiga et al.	
6,032,188	A		2/2000	Mairs et al.	
6,038,602	A		3/2000	Ishikawa	
6,047,289	A		4/2000	Thorne et al.	
6,073,177	A	*	6/2000	Hebel et al.	709/228
6,094,676	A		7/2000	Gray et al.	
6,199,116	B1		3/2001	May et al.	
6,216,177	B1		4/2001	Mairs et al.	
6,223,212	B1		4/2001	Batty et al.	
6,243,691	B1		6/2001	Fisher et al.	
6,252,884	B1	*	6/2001	Hunter	370/443
6,268,855	B1		7/2001	Mairs et al.	
6,271,839	B1		8/2001	Mairs et al.	
6,285,363	B1		9/2001	Mairs et al.	
6,304,928	B1		10/2001	Mairs et al.	
6,353,599	B1	*	3/2002	Bi et al.	370/328
6,618,752	B1	*	9/2003	Moore et al.	709/217

OTHER PUBLICATIONS

U.S. patent application Ser. No. 09/629,023, Bourassa et al., filed Jul. 31, 2000.

U.S. patent application Ser. No. 09/629,043, Bourassa et al., filed Jul. 31, 2000.

U.S. patent application Ser. No. 09/629,024, Bourassa et al., filed Jul. 31, 2000.

U.S. patent application Ser. No. 09/629,042, Bourassa et al., filed Jul. 31, 2000.

Murphy, Patricia, A., "The Next Generation Networking Paradigm: Producer/Consumer Model," Dedicated Systems Magazine—2000 (pp. 26–28).

The Gamer's Guide, "First–Person Shooters," Oct. 20, 1998 (4 pages).

The O'Reilly Network, "Gnutella: Alive, Well, and Changing Fast," Jan. 25, 2001 (5 pages) http://www.open2p.com/lpt/ . . . [Accessed Jan. 29, 2002].

Oram, Andy, "Gnutella and Freenet Represents True Technological Innovation," May 12, 2000 (7 pages) The O'Reilly Network http://www.oreillynet.com/1pt . . . [Accessed Jan. 29, 2002].

Internetworking Technologies Handbook, Chapter 43 (pp. 43–1 –43–16).

Oram, Andy, "Peer–to–Peer Makes the Internet Interesting Again," Sep. 22, 2000 (7 pages) The O'Reilly Network http://linux.oreillynet.com/1pt . . . [Accessed Jan. 29, 2002].

Monte, Richard, "The Random Walk for Dummies, "MIT Undergraduate Journal of Mathematics (pp. 143–148).

Srinivasan, R., "XDR: External Data Representation Standard," Sun Microsystems, Aug. 1995 (20 pages) Internet RFC/STD/FYI/BCP Archives http://www.faqs.org/rfcs/rfc1832.html [Accessed Jan. 29, 2002].

A Databeam Corporate White Paper, "A Primer on the T.120 Series Standards," Copyright 1995 (pp. 1–16).

Kessler, Gary, C., "An Overview of TCP/IP Protocols and the Internet," Apr. 23, 1999 (23 pages) Hill Associates, Inc. http://www.hill.com/library/publications/t . . . [Accessed Jan. 29, 2002].

Bondy, J.A., and Murty, U.S.R., "Graph Theory with Applications," Chapters 1–3 (pp. 1–47), 1976 American Elsevier Publishing Co., Inc., New York, New York.

Cormen, Thomas H. et al., Introduction to Algorithms, Chapter 5.3 (pp. 84–91), Chapter 12 (pp. 218–243), Chapter 13 (p. 245), 1990, The MIT Press, Cambridge, Massachusetts, McGraw–Hill Book Company, New York.

The Common Object Request Broker: Architecture and Specification, Revision 2.6, Dec. 2001, Chapter 12 (pp. 12–1–12–10), Chapter 13 (pp. 13–1–13–56) Chapter 16 (pp. 16–1 –16–26), Chapter 18 (pp. 18–1 –18–52), Chapter 20 (pp. 20–1–20–22).

The University of Warwick, Computer Science Open Days, "Demonstration on the Problems of Distributed Systems," http://www.dcs.warwick.ac.u . . . [Accessed Jan. 29, 2002].

* cited by examiner

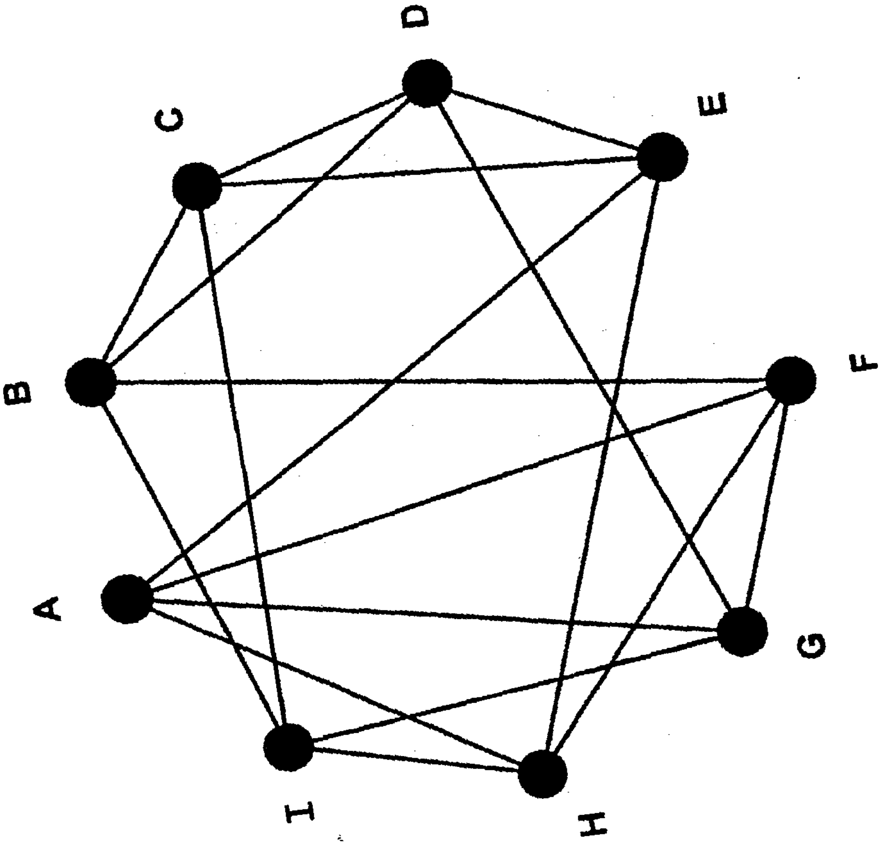


Fig. 1

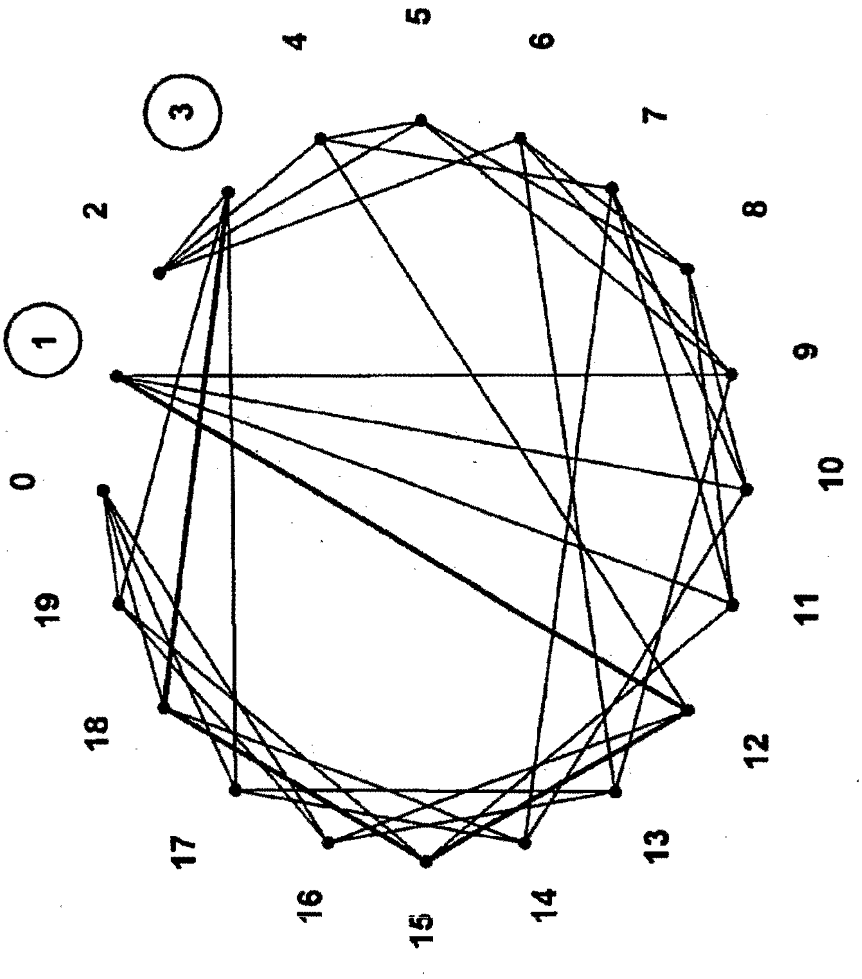


Fig. 2

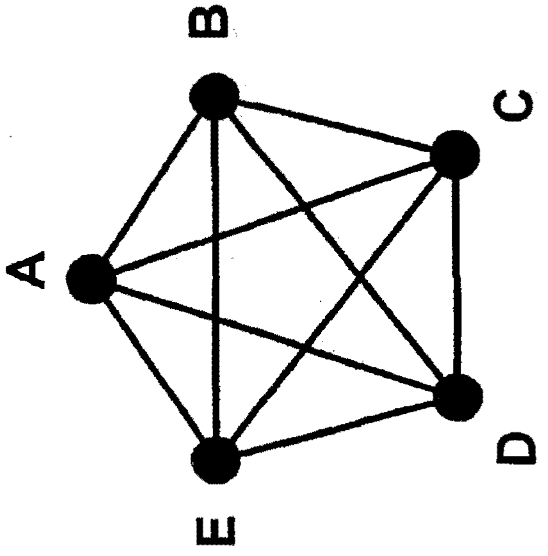


Fig. 3A

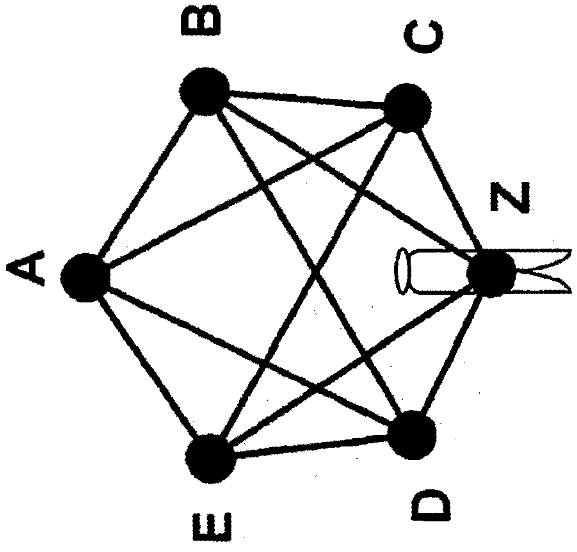


Fig. 3B

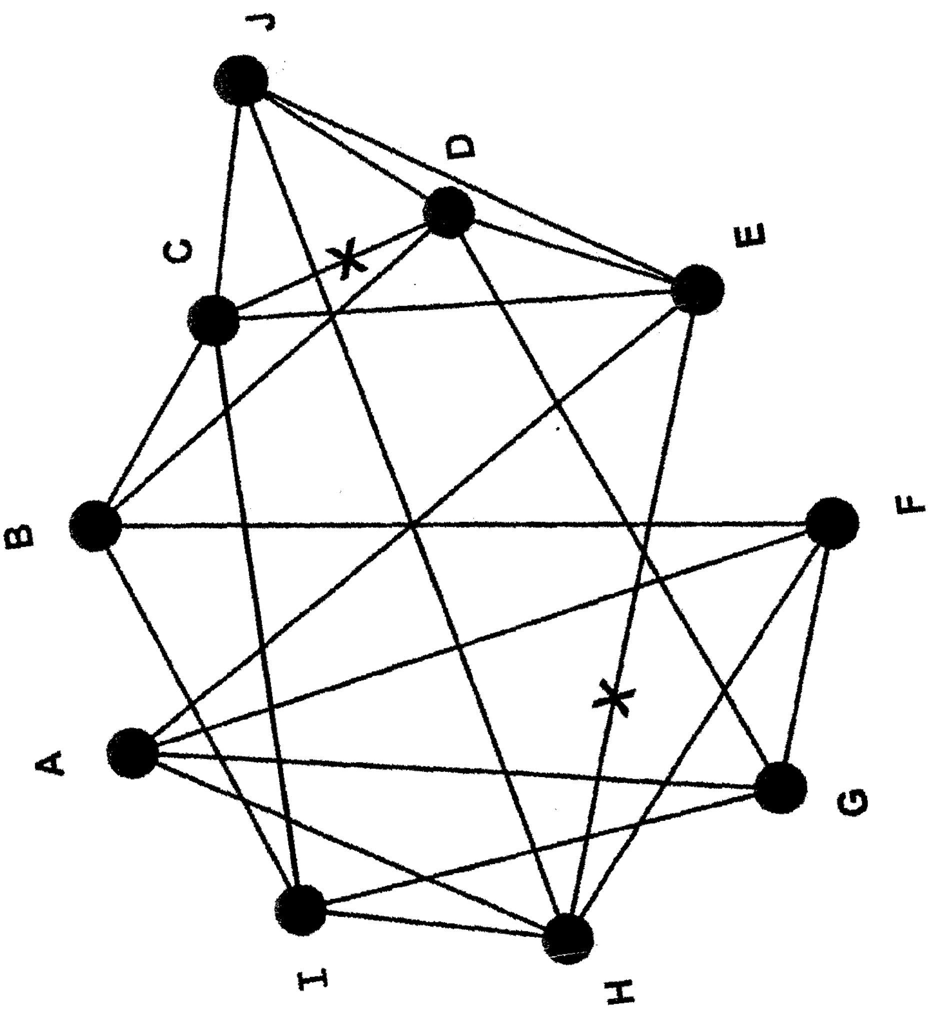


Fig. 4A

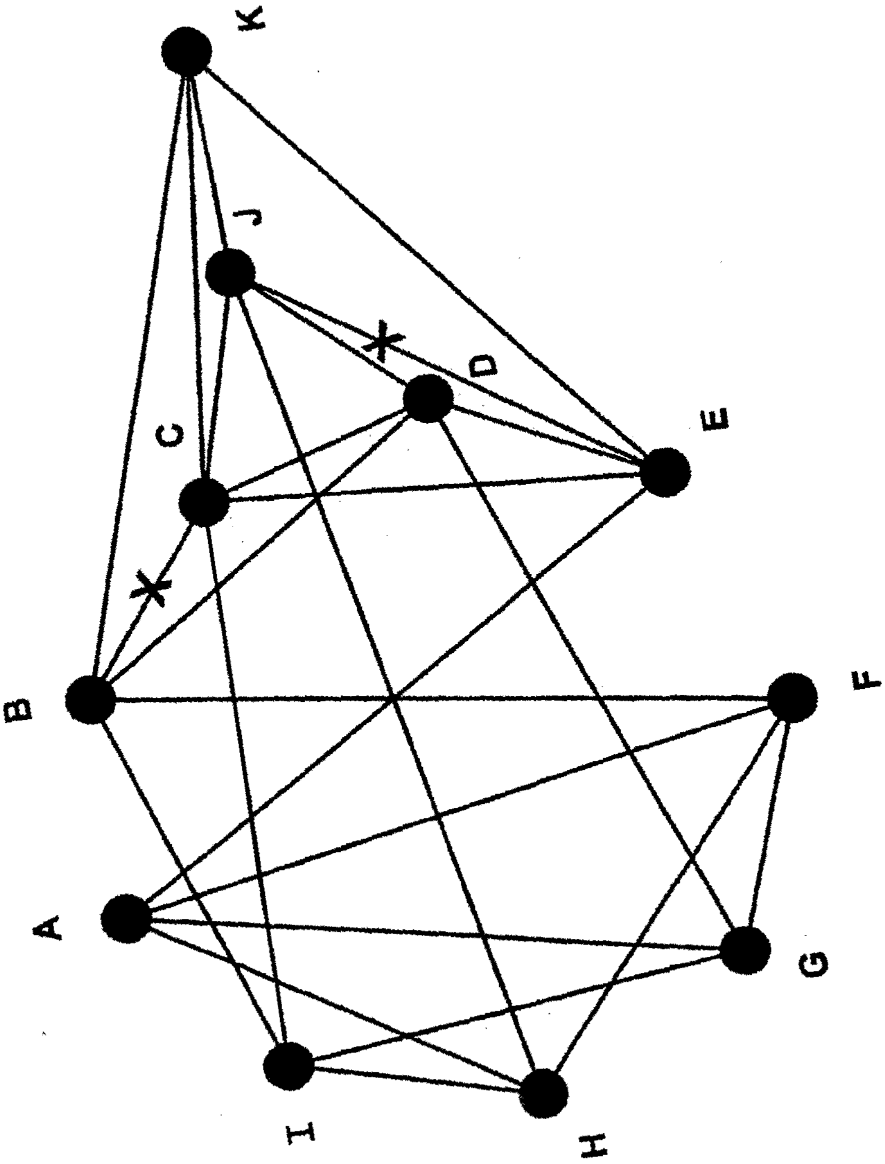


Fig. 4B

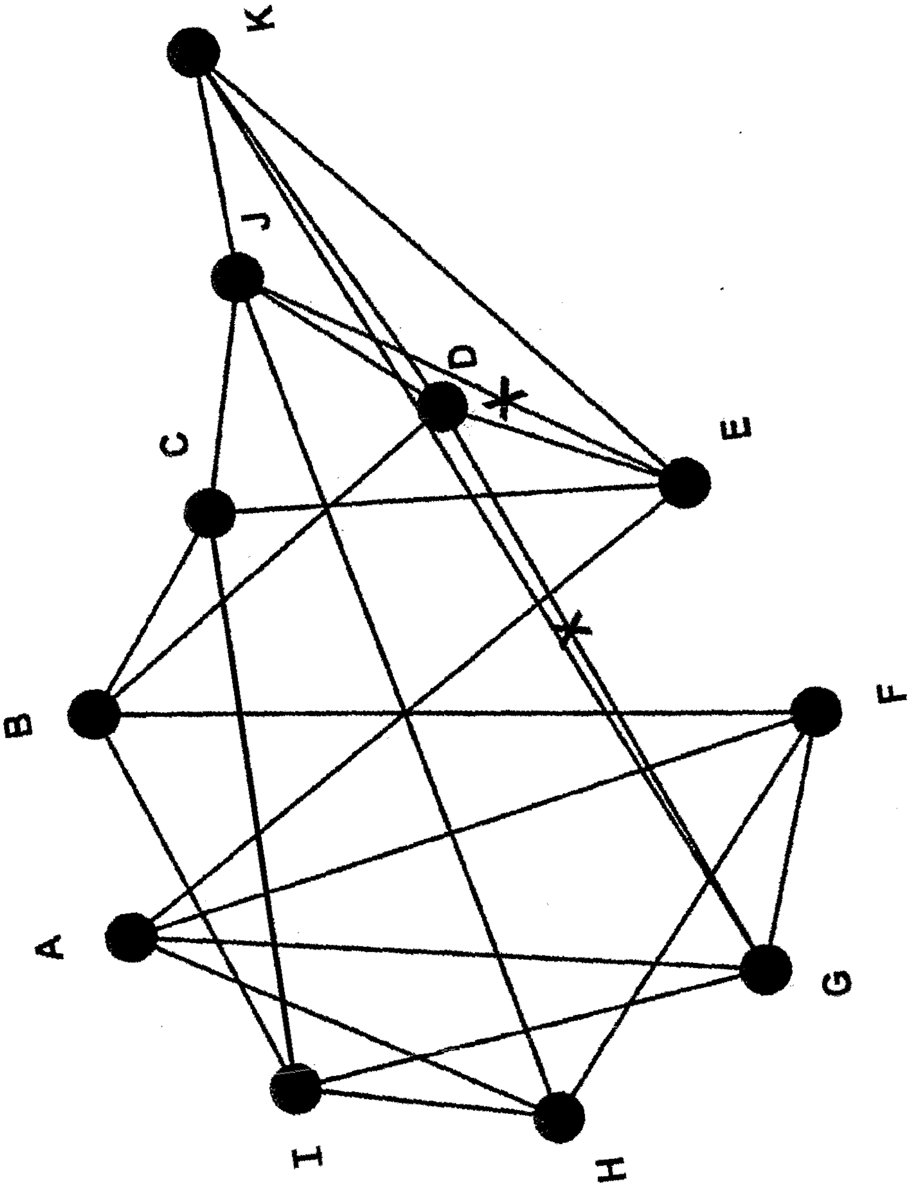


Fig. 4C

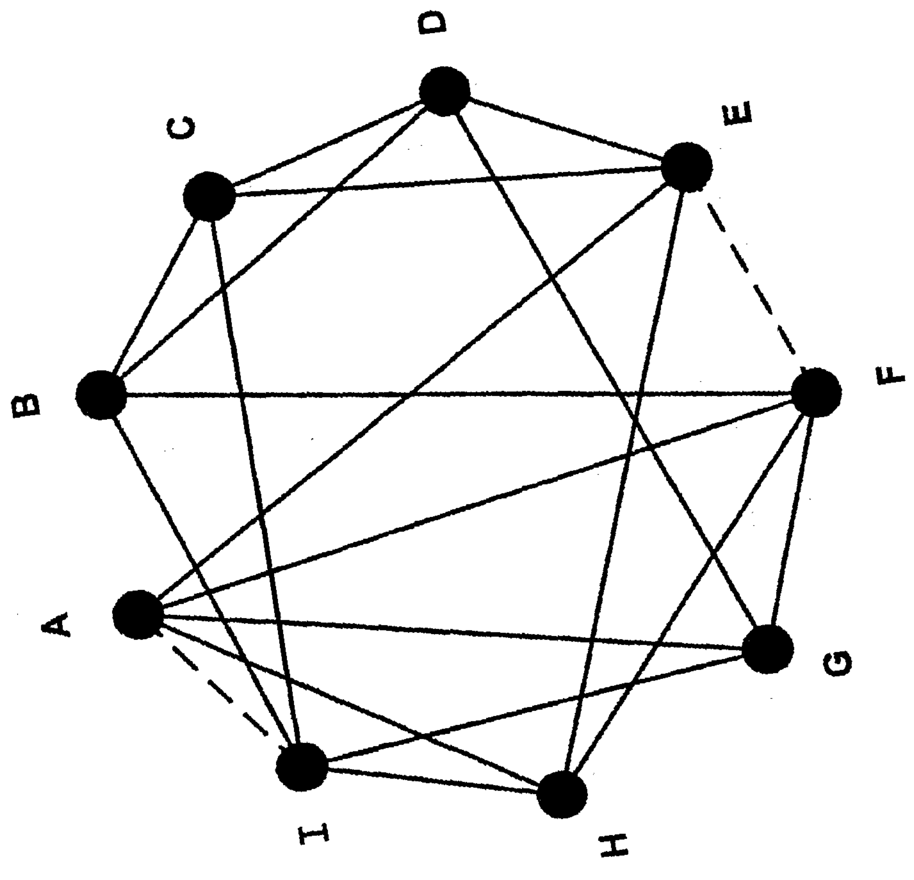


Fig. 5A

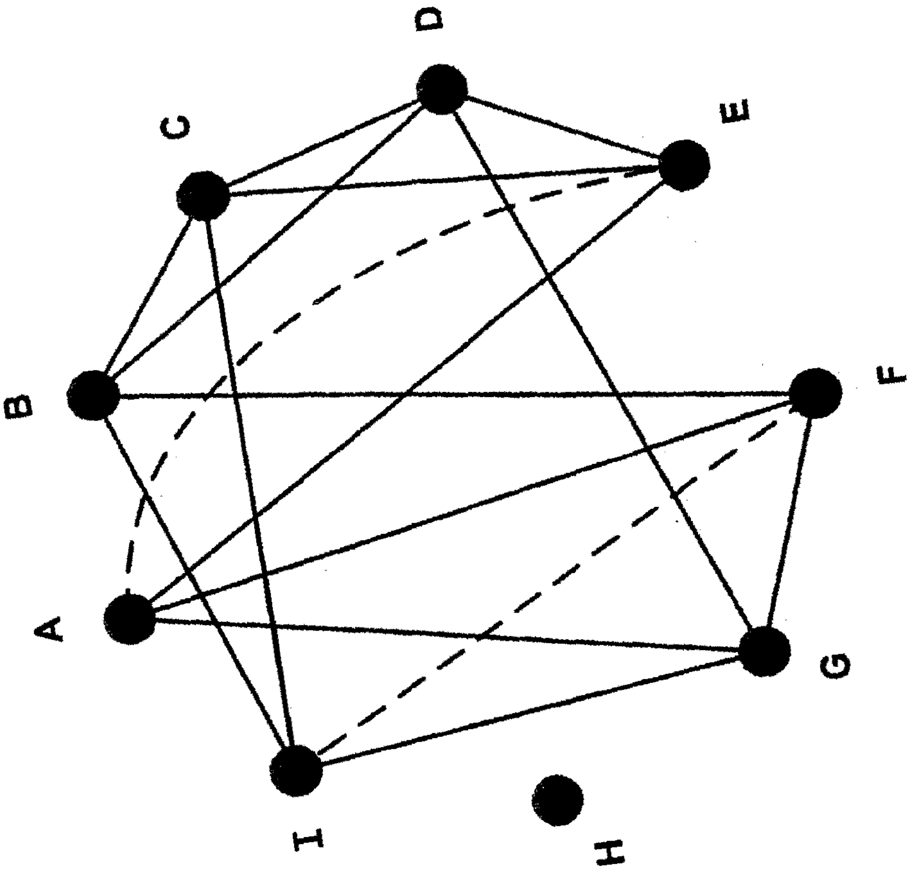


Fig. 5B

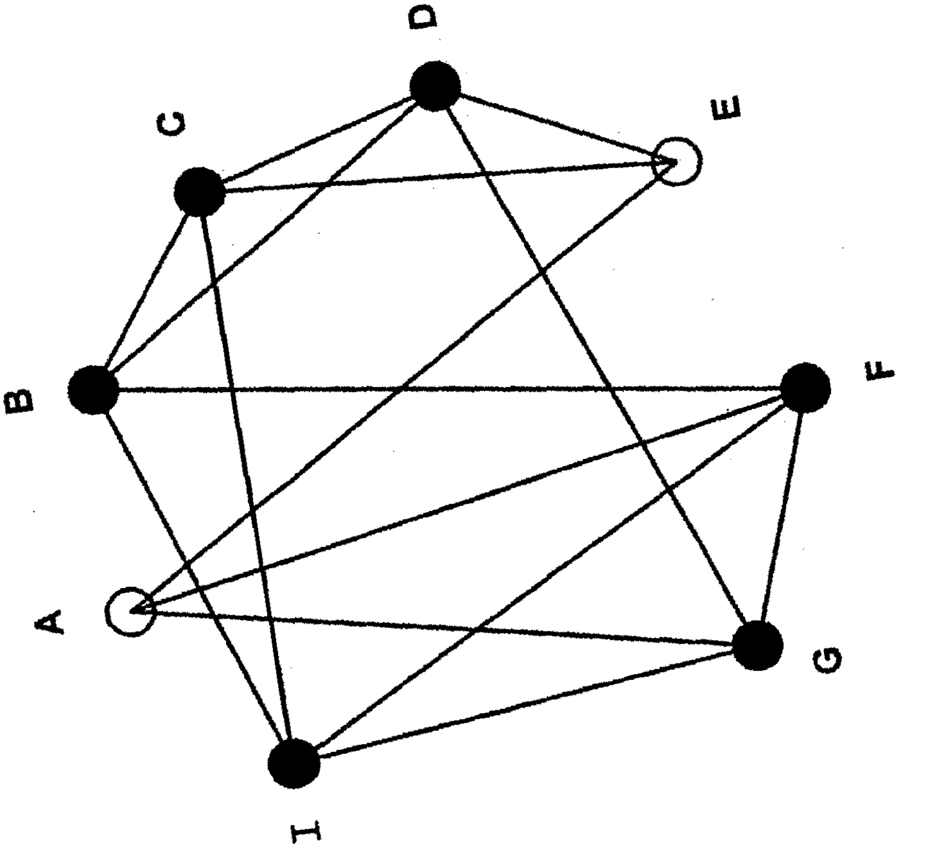


Fig. 5C

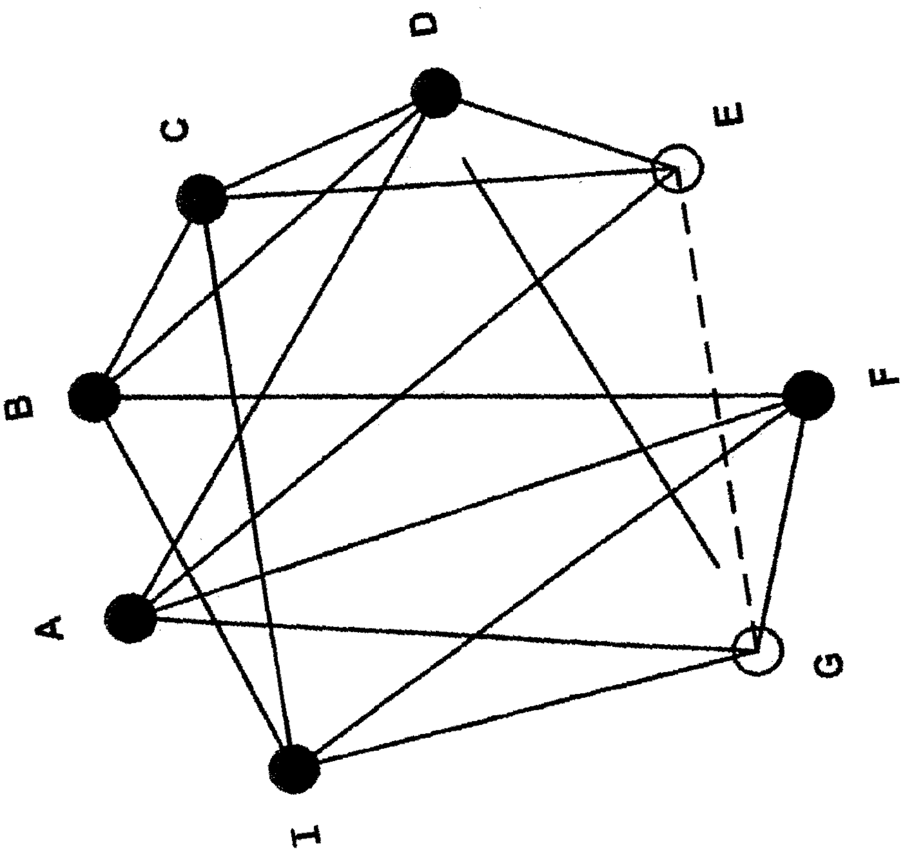


Fig. 5D

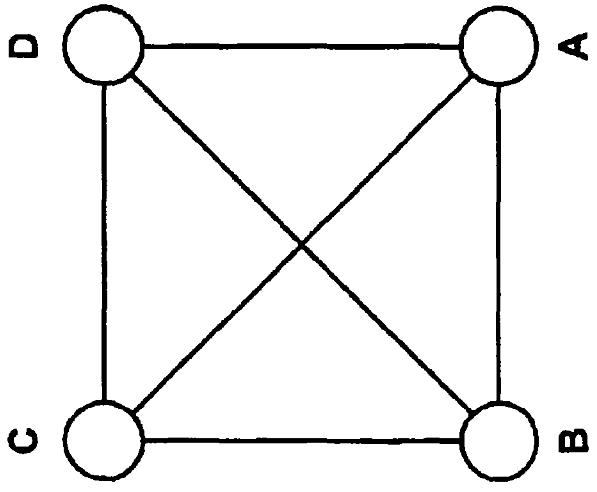


Fig. 5E

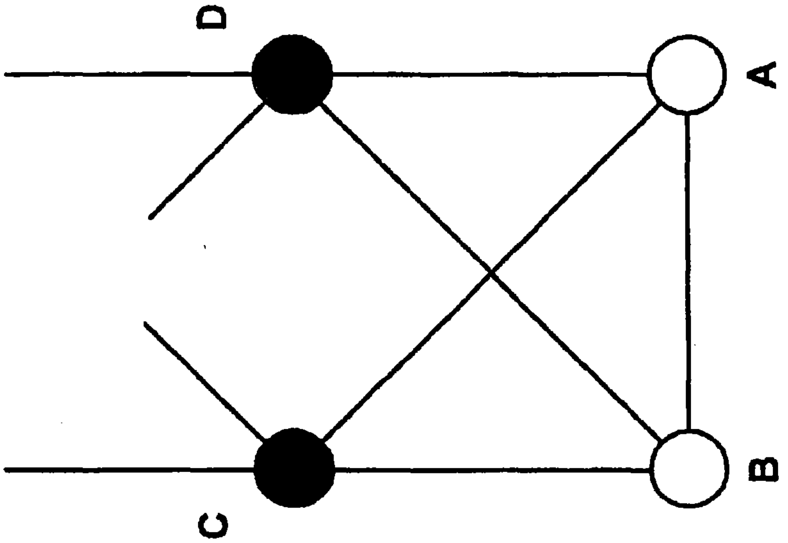


Fig. 5F

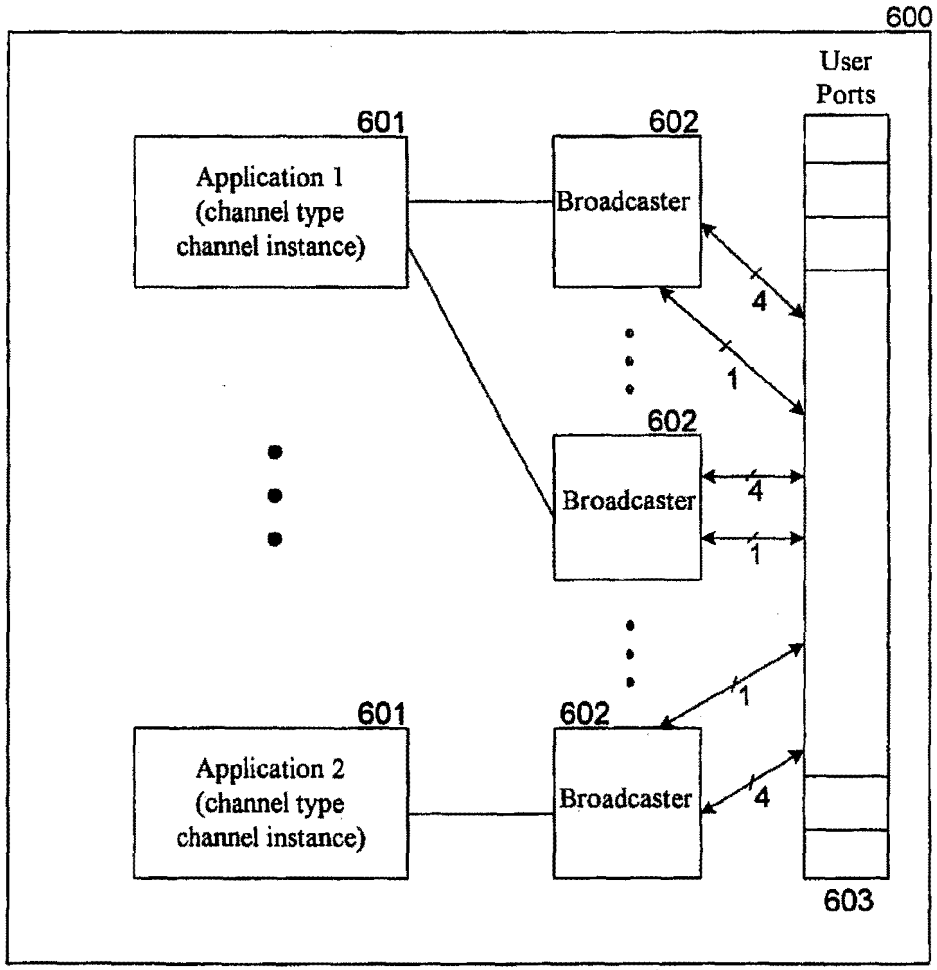

600
User Ports
601
Application 1 (channel type channel instance)
602
Broadcaster
4
1
602
Broadcaster
4
1
601
Application 2 (channel type channel instance)
602
Broadcaster
1
4
603

Fig. 6

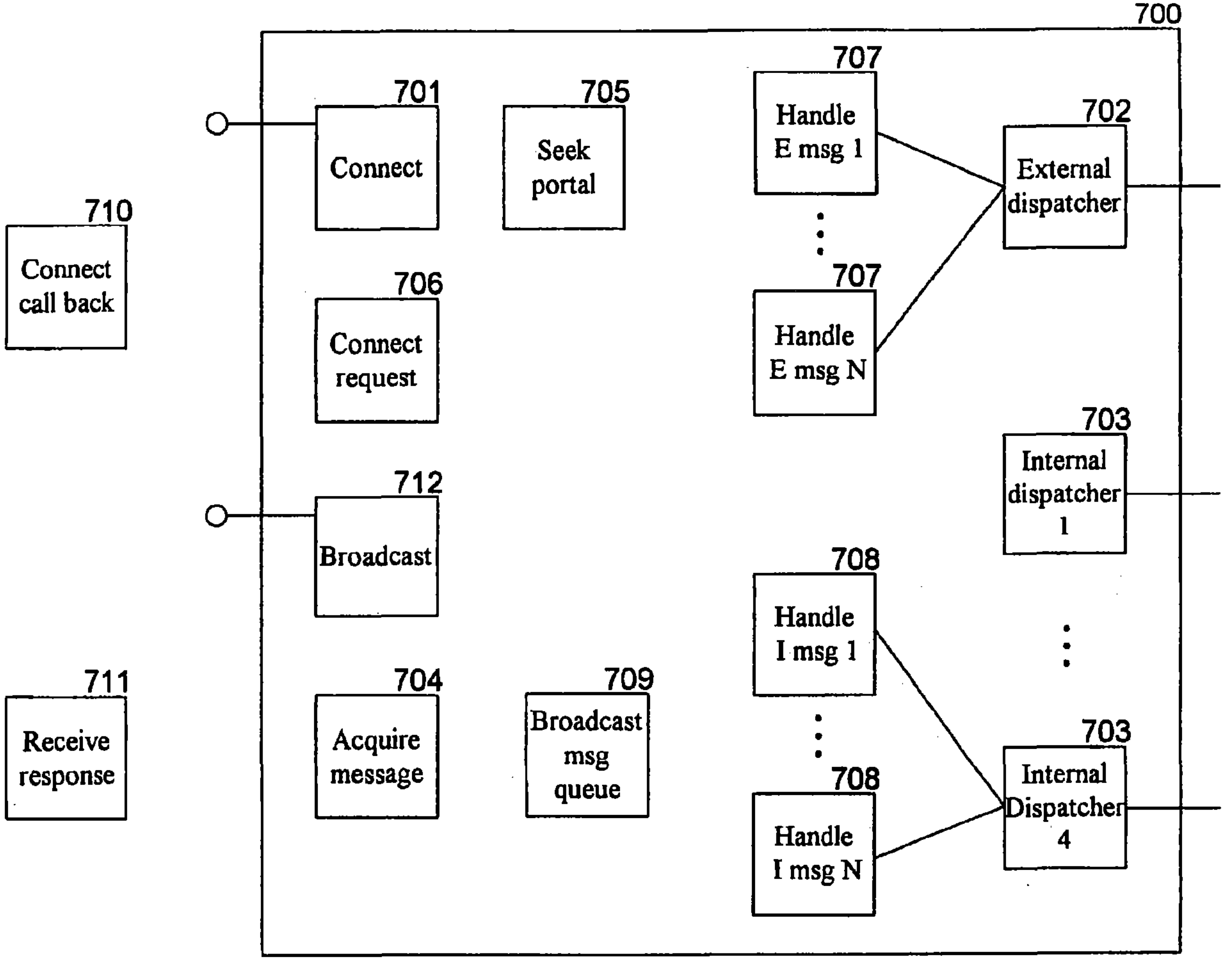

700
701
Connect
705
Seek portal
707
Handle E msg 1
707
Handle E msg N
702
External dispatcher
710
Connect call back
706
Connect request
703
Internal dispatcher 1
712
Broadcast
708
Handle I msg 1
708
Handle I msg N
703
Internal Dispatcher 4
711
Receive response
704
Acquire message
709
Broadcast msg queue

Fig. 7

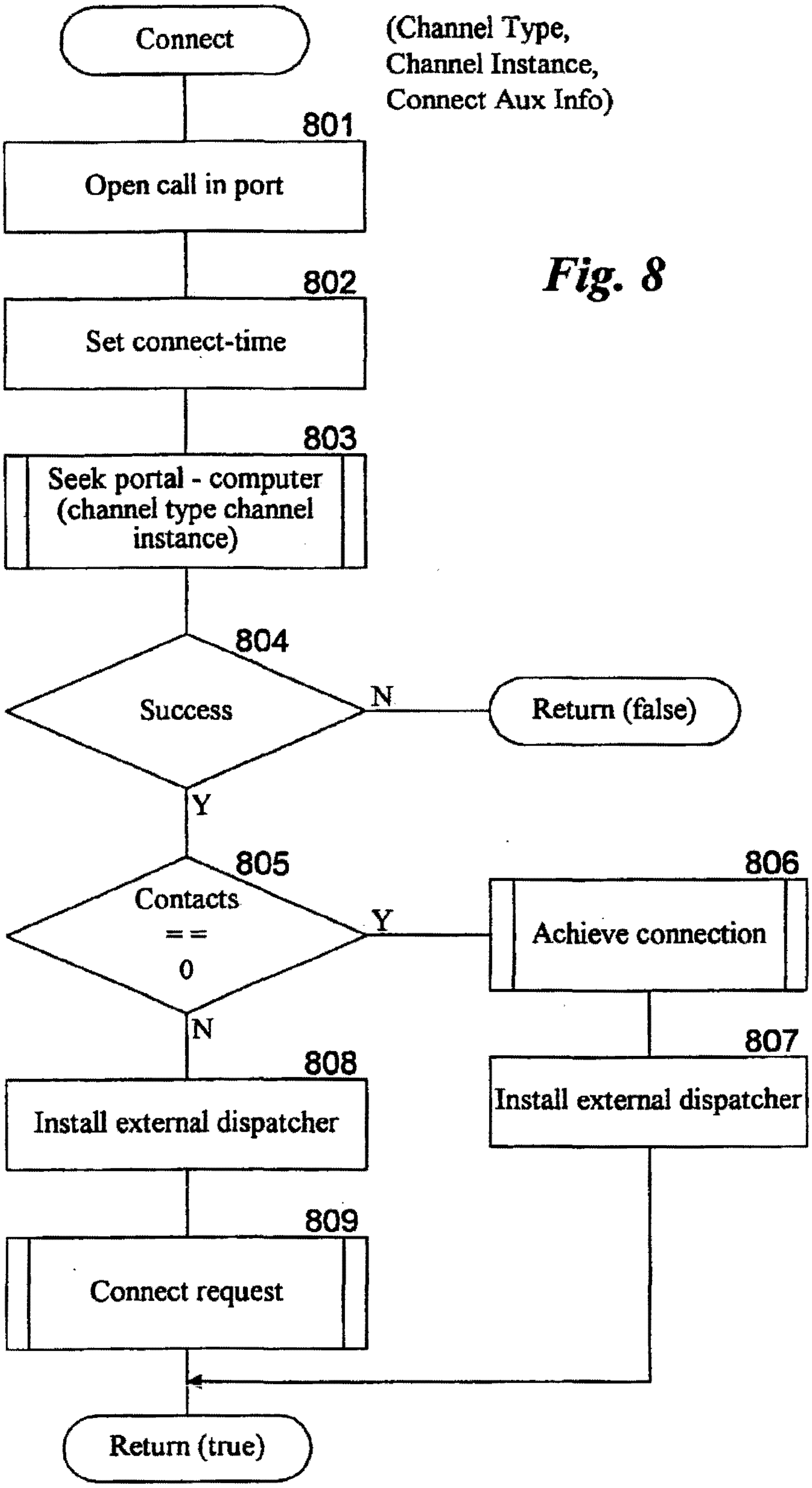

Connect
(Channel Type,
Channel Instance,
Connect Aux Info)
801
Open call in port
802
Set connect-time
803
Seek portal - computer (channel type channel instance)
804
Success
N
Return (false)
Y
805
Contacts == 0
Y
806
Achieve connection
N
808
Install external dispatcher
807
Install external dispatcher
809
Connect request
Return (true)

Fig. 8

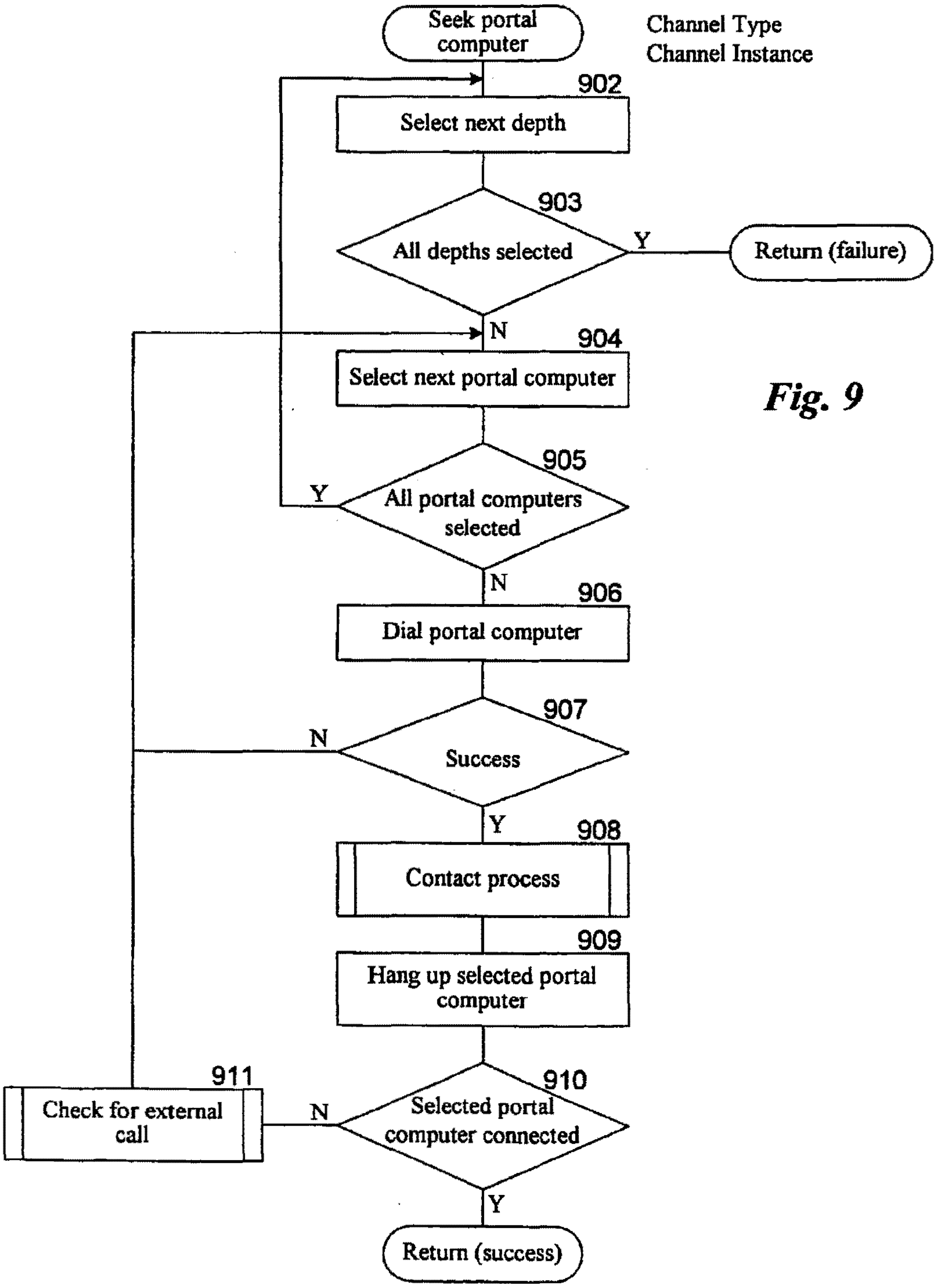

Seek portal computer
Channel Type
Channel Instance
902
Select next depth
903
All depths selected
Y
Return (failure)
N
904
Select next portal computer
905
Y
All portal computers selected
N
906
Dial portal computer
907
N
Success
Y
908
Contact process
909
Hang up selected portal computer
911
Check for external call
N
910
Selected portal computer connected
Y
Return (success)

Fig. 9

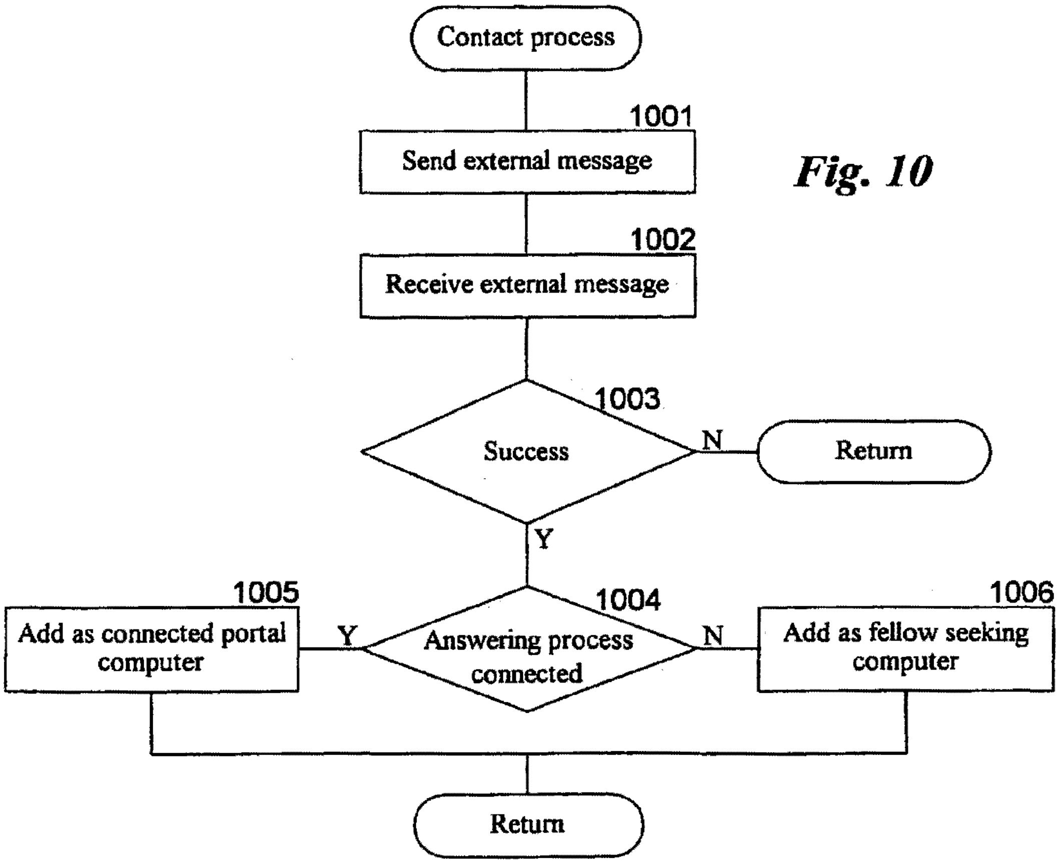
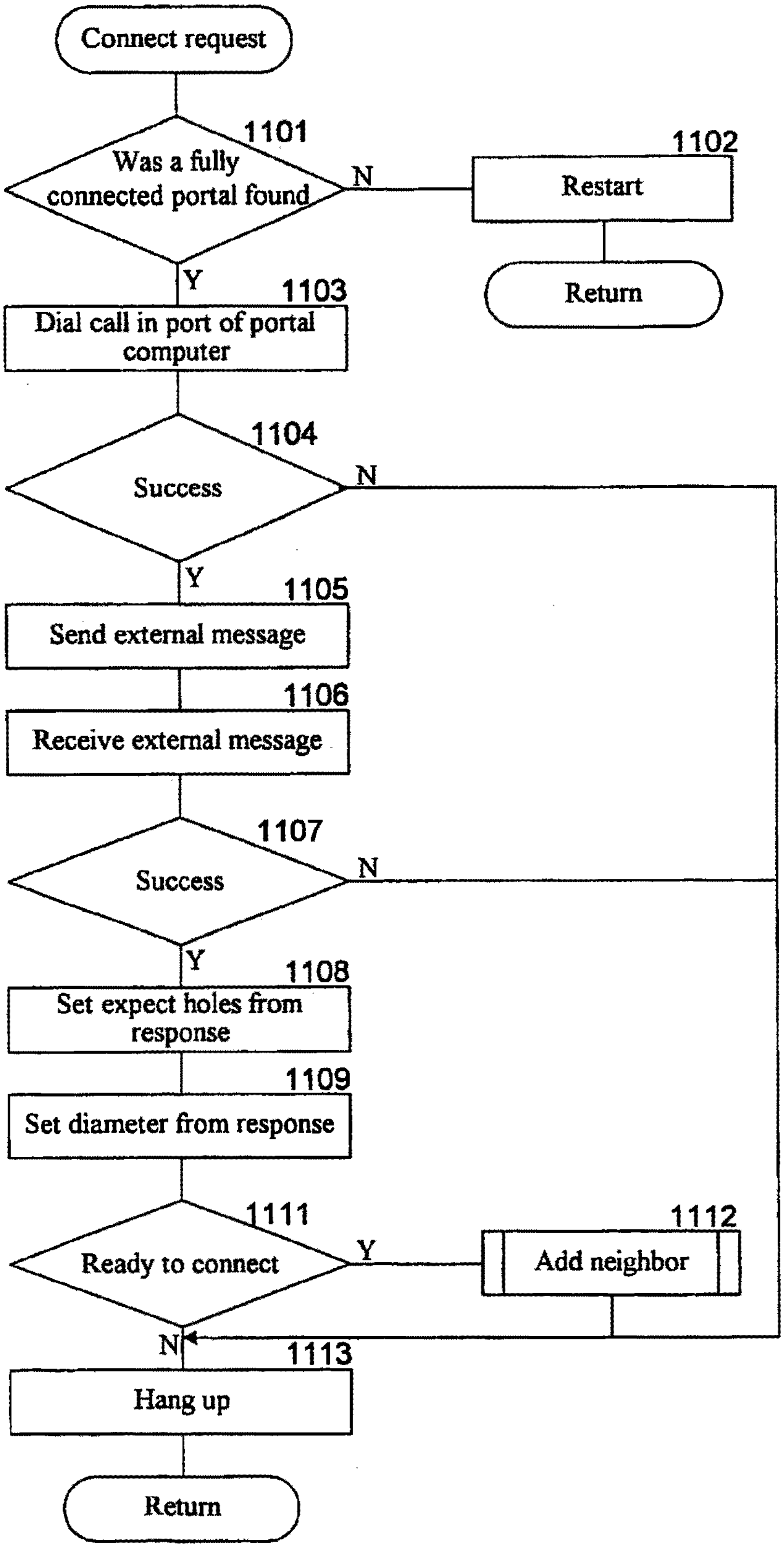

Contact process
1001
Send external message
1002
Receive external message
1003
Success
N
Return
Y
1005
Add as connected portal computer
Y
1004
Answering process connected
N
1006
Add as fellow seeking computer
Return

Fig. 10

Fig. 11


Connect request
1101
Was a fully connected portal found
N
1102
Restart
Return
Y
1103
Dial call in port of portal computer
1104
Success
N
Y
1105
Send external message
1106
Receive external message
1107
Success
N
Y
1108
Set expect holes from response
1109
Set diameter from response
1111
Ready to connect
Y
1112
Add neighbor
N
1113
Hang up
Return

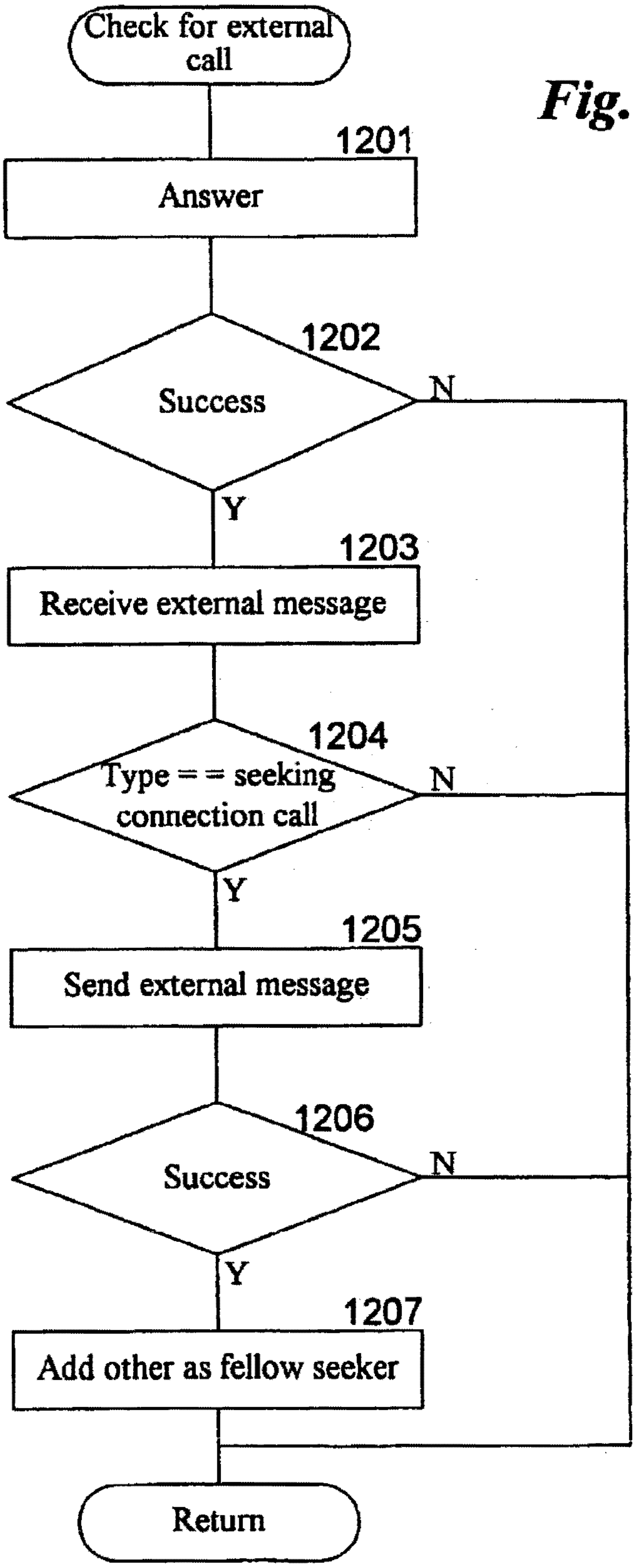

Check for external call
1201
Answer
1202
Success
N
Y
1203
Receive external message
1204
Type = = seeking connection call
N
Y
1205
Send external message
1206
Success
N
Y
1207
Add other as fellow seeker
Return

Fig. 12

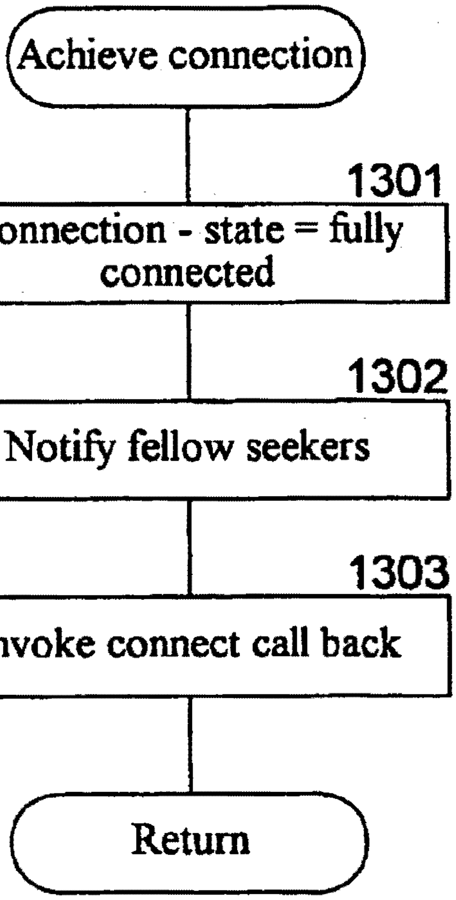

Achieve connection
1301
Connection - state = fully connected
1302
Notify fellow seekers
1303
Invoke connect call back
Return

Fig. 13

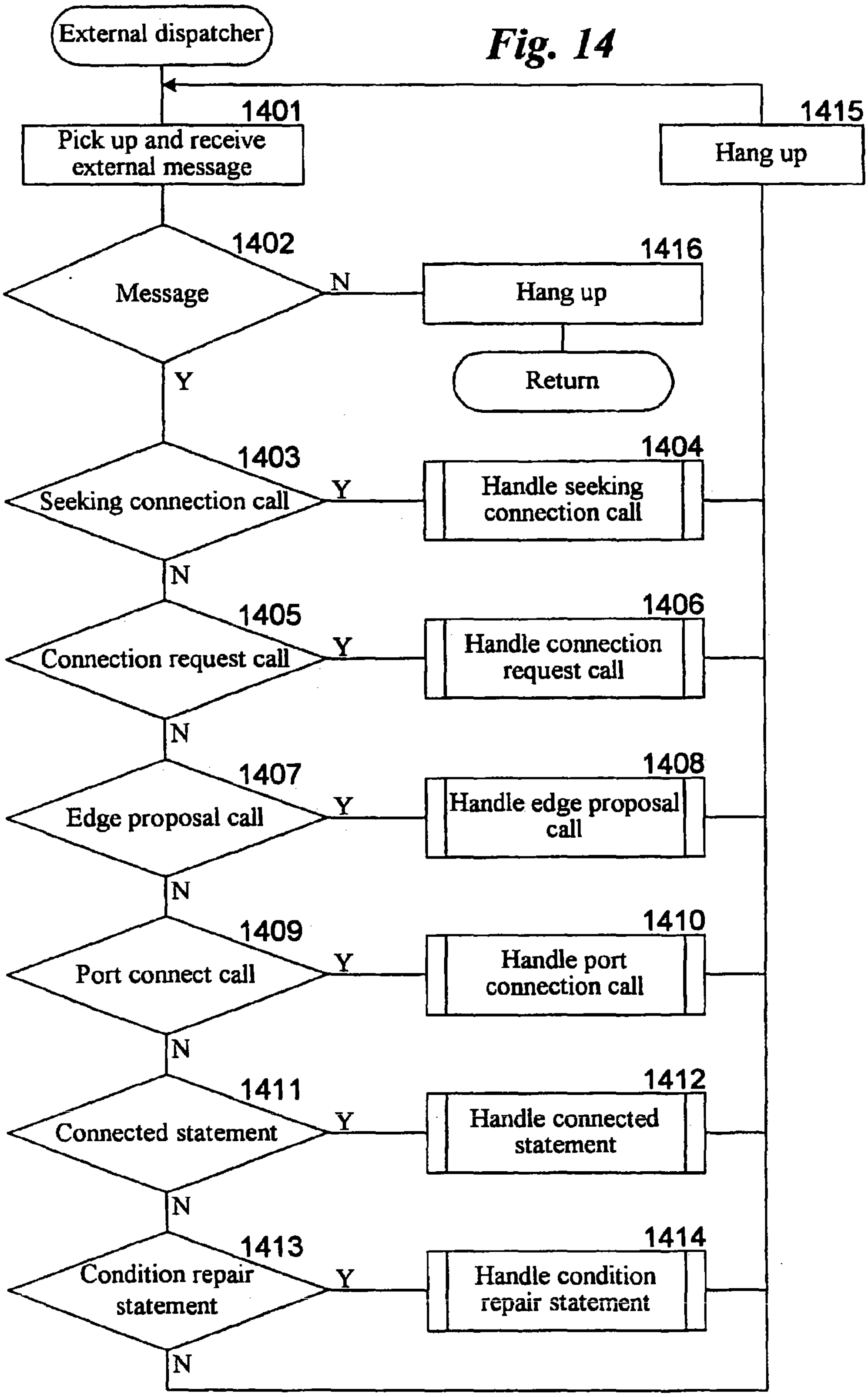

External dispatcher
1401
Pick up and receive external message
1415
Hang up
1402
Message
N
1416
Hang up
Return
Y
1403
Seeking connection call
Y
1404
Handle seeking connection call
N
1405
Connection request call
Y
1406
Handle connection request call
N
1407
Edge proposal call
Y
1408
Handle edge proposal call
N
1409
Port connect call
Y
1410
Handle port connection call
N
1411
Connected statement
Y
1412
Handle connected statement
N
1413
Condition repair statement
Y
1414
Handle condition repair statement
N

Fig. 14

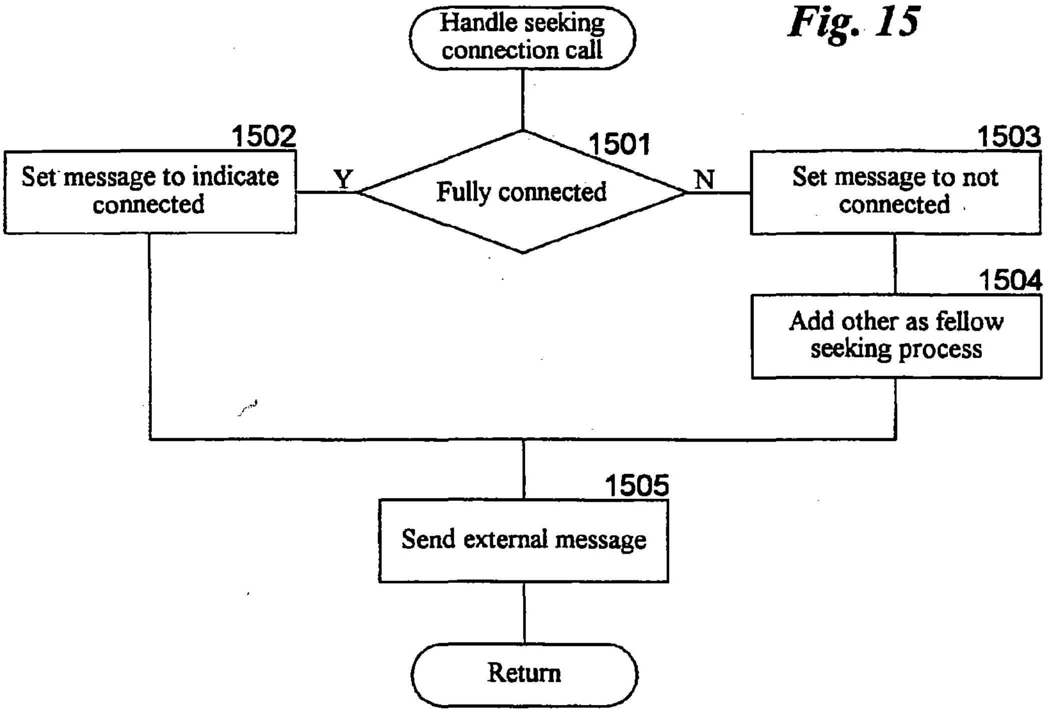

Handle seeking connection call
1501
Fully connected
Y
N
1502
Set message to indicate connected
1503
Set message to not connected
1504
Add other as fellow seeking process
1505
Send external message
Return

Fig. 15

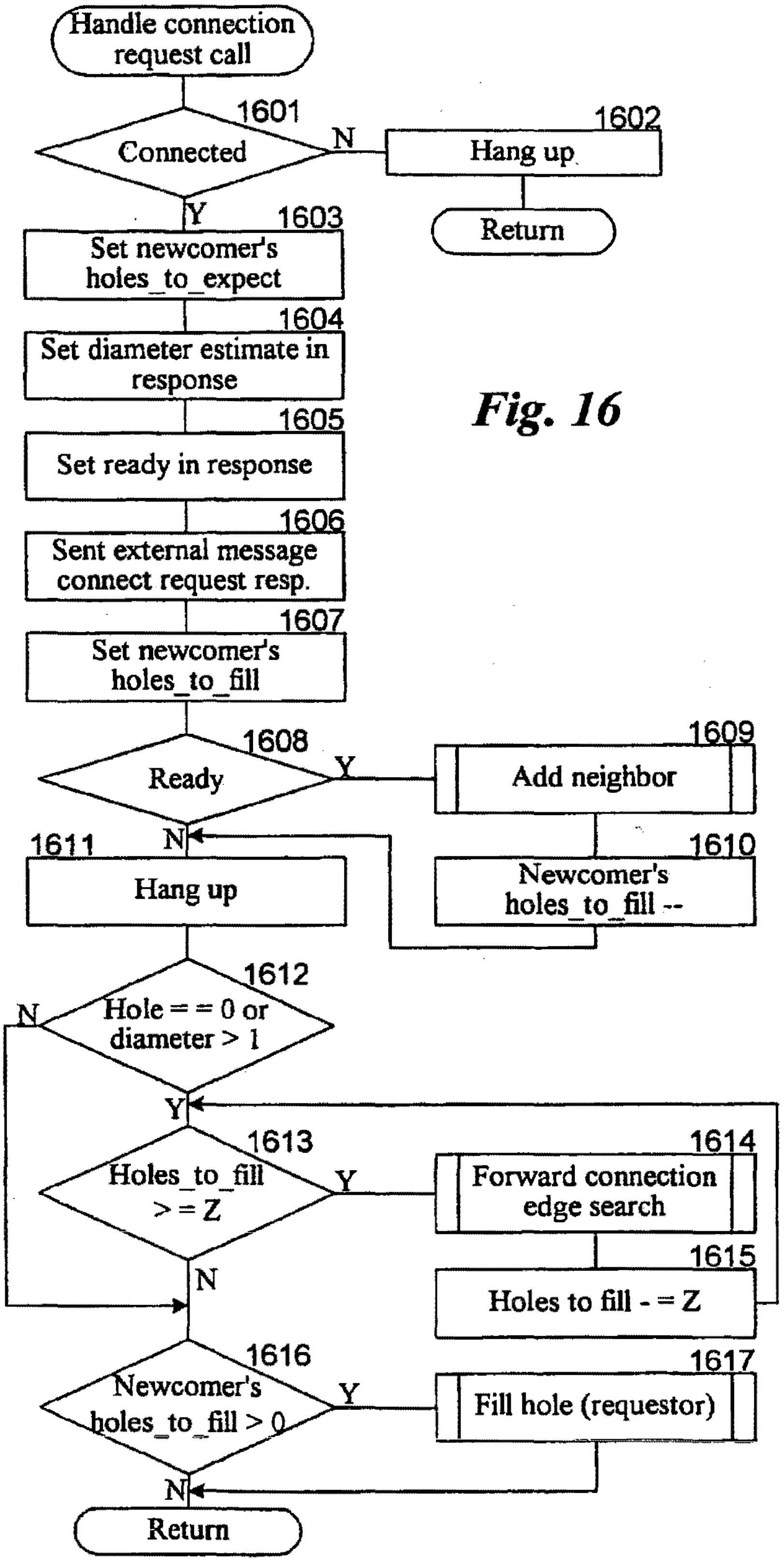

Handle connection request call
1601
Connected
N
1602
Hang up
Return
Y
1603
Set newcomer's holes_to_expect
1604
Set diameter estimate in response
1605
Set ready in response
1606
Sent external message connect request resp.
1607
Set newcomer's holes_to_fill
1608
Ready
Y
1609
Add neighbor
N
1611
Hang up
1610
Newcomer's holes_to_fill --
1612
Hole = = 0 or diameter > 1
N
Y
1613
Holes_to_fill > = Z
Y
1614
Forward connection edge search
1615
Holes to fill - = Z
N
1616
Newcomer's holes_to_fill > 0
Y
1617
Fill hole (requestor)
N
Return

Fig. 16

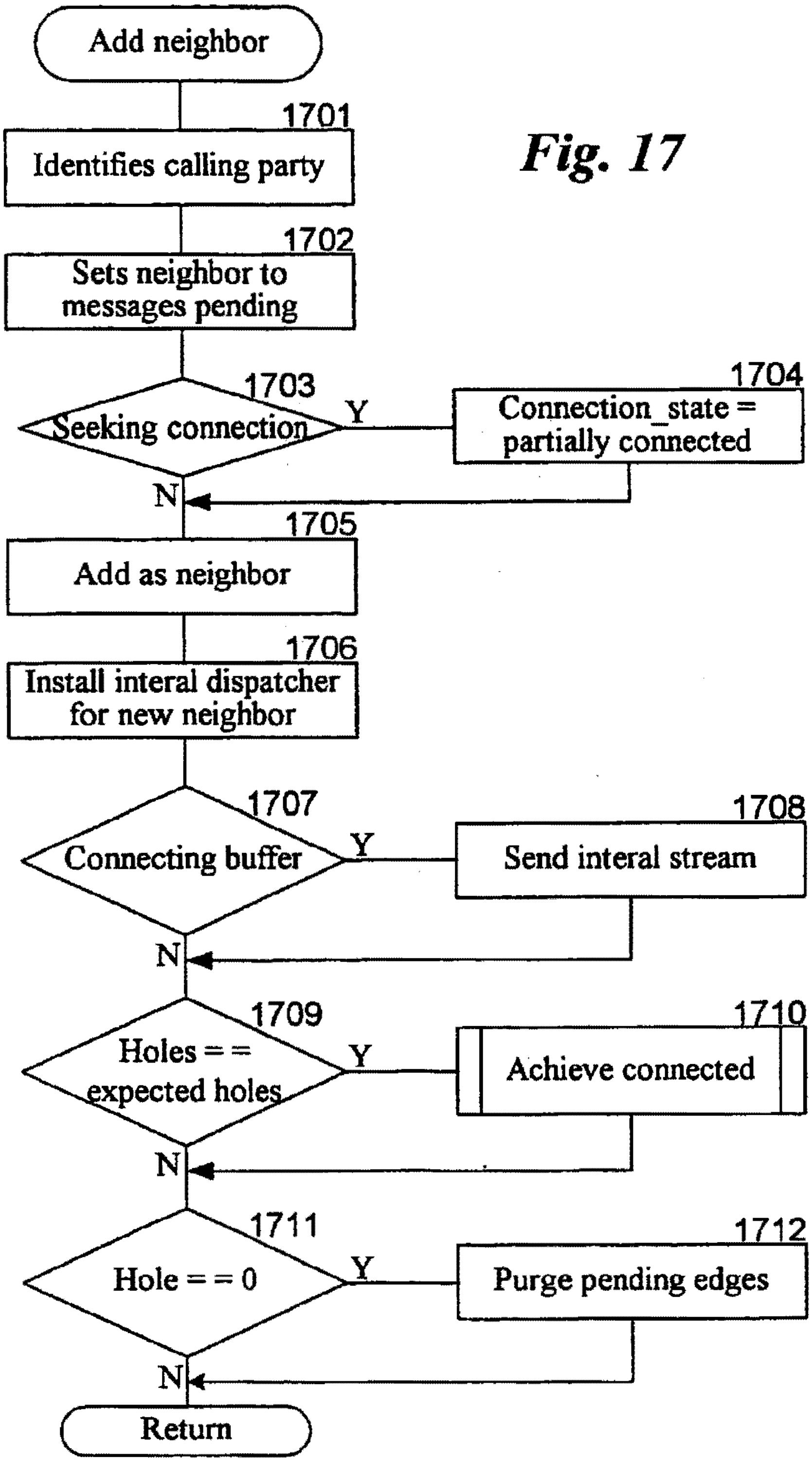
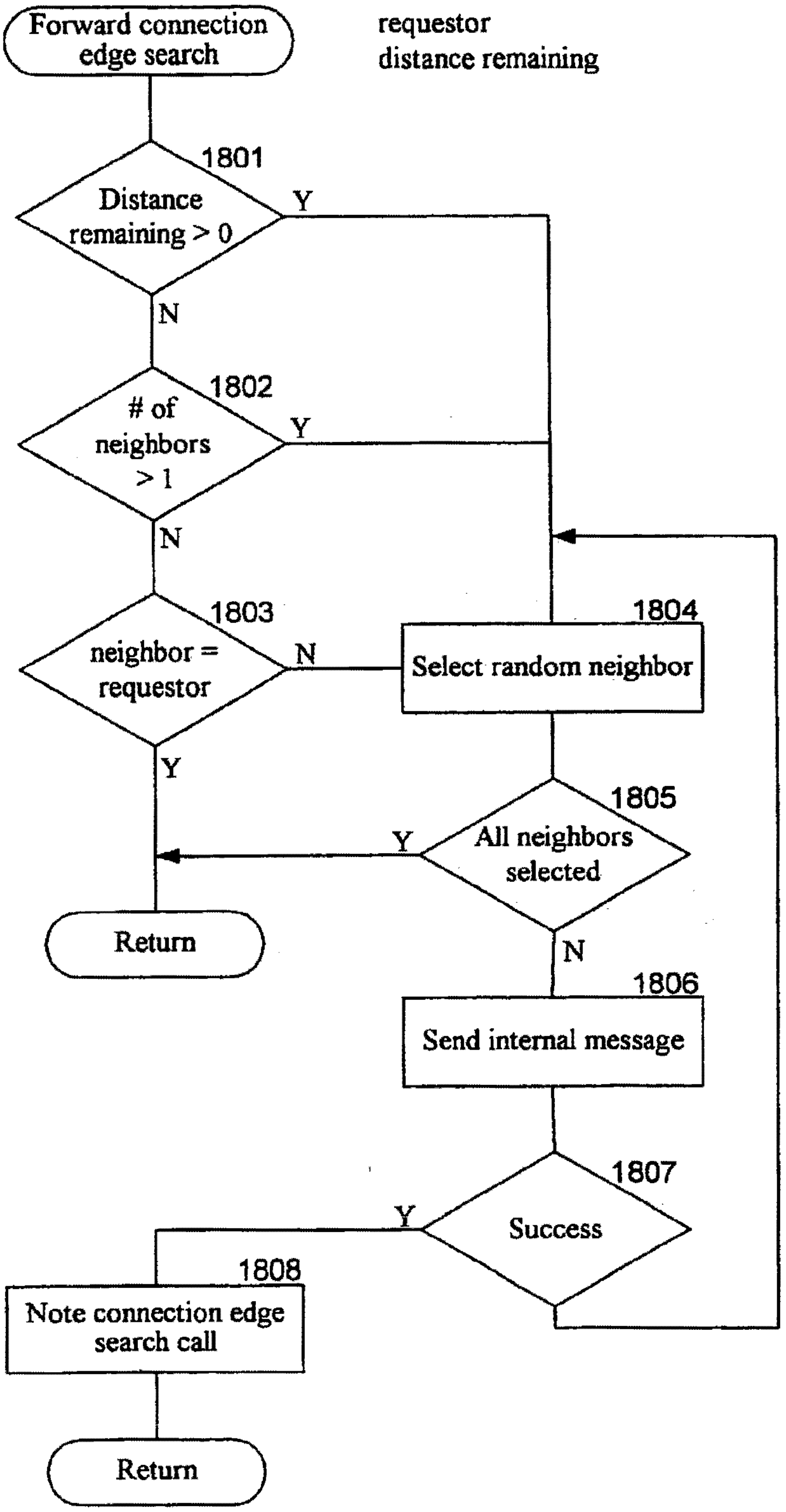

Add neighbor
1701
Identifies calling party
1702
Sets neighbor to messages pending
1703
Seeking connection
Y
1704
Connection_state = partially connected
N
1705
Add as neighbor
1706
Install interal dispatcher for new neighbor
1707
Connecting buffer
Y
1708
Send interal stream
N
1709
Holes = = expected holes
Y
1710
Achieve connected
N
1711
Hole = = 0
Y
1712
Purge pending edges
N
Return

Fig. 17

Fig. 18


Forward connection edge search
requestor
distance remaining
1801
Distance remaining > 0
Y
N
1802
# of neighbors > 1
Y
N
1803
neighbor = requestor
N
Y
1804
Select random neighbor
1805
All neighbors selected
Y
N
Return
1806
Send internal message
1807
Success
Y
1808
Note connection edge search call
Return

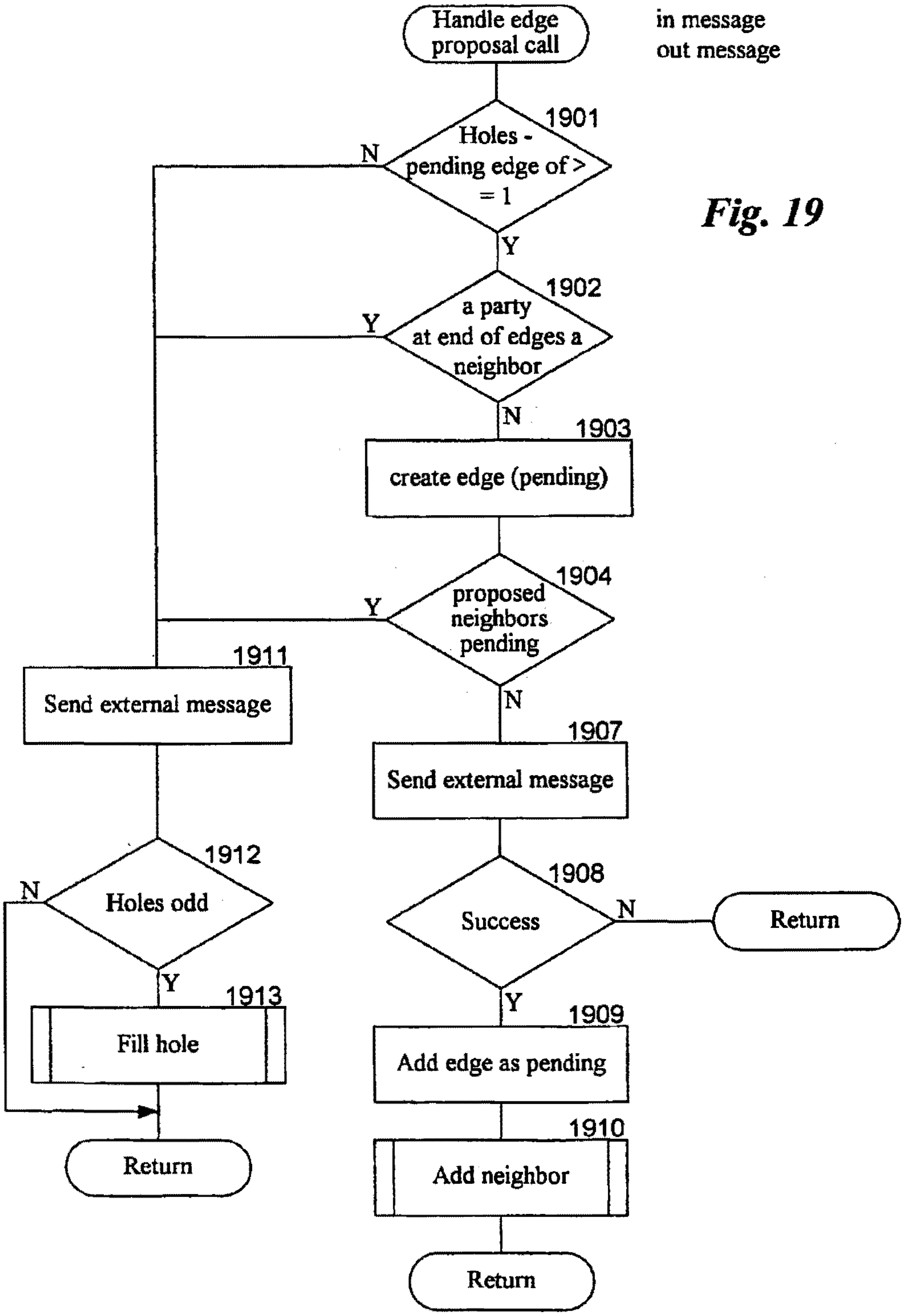

Handle edge proposal call
in message
out message
1901
Holes - pending edge of > = 1
N
Y
1902
a party at end of edges a neighbor
Y
N
1903
create edge (pending)
1904
proposed neighbors pending
Y
N
1911
Send external message
1907
Send external message
1912
Holes odd
N
Y
1913
Fill hole
Return
1908
Success
N
Return
Y
1909
Add edge as pending
1910
Add neighbor
Return

Fig. 19

Fig. 20

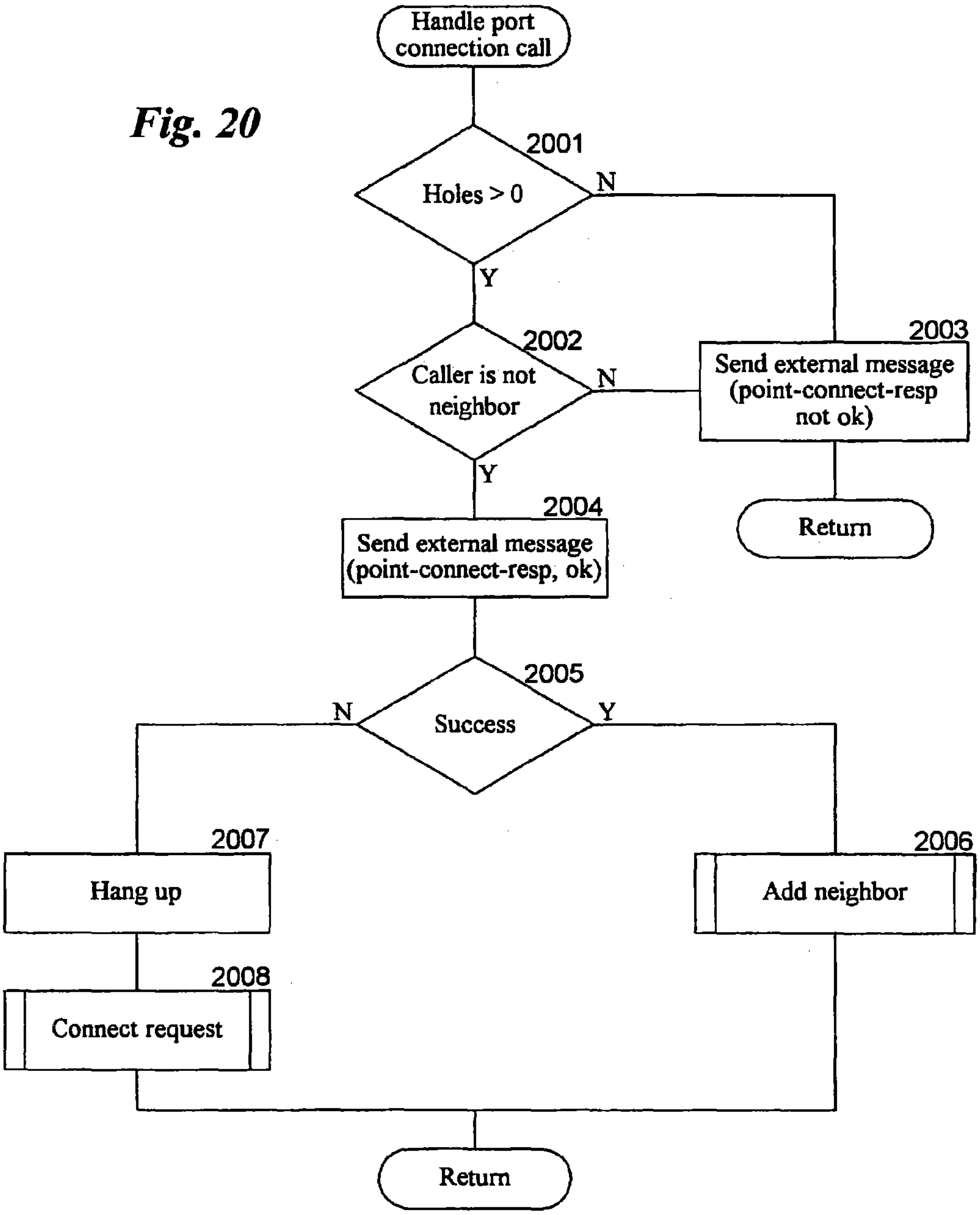

Handle port connection call
2001
Holes > 0
N
Y
2002
Caller is not neighbor
N
Y
2003
Send external message (point-connect-resp not ok)
Return
2004
Send external message (point-connect-resp, ok)
2005
Success
N
Y
2007
Hang up
2008
Connect request
2006
Add neighbor
Return

Fig. 21

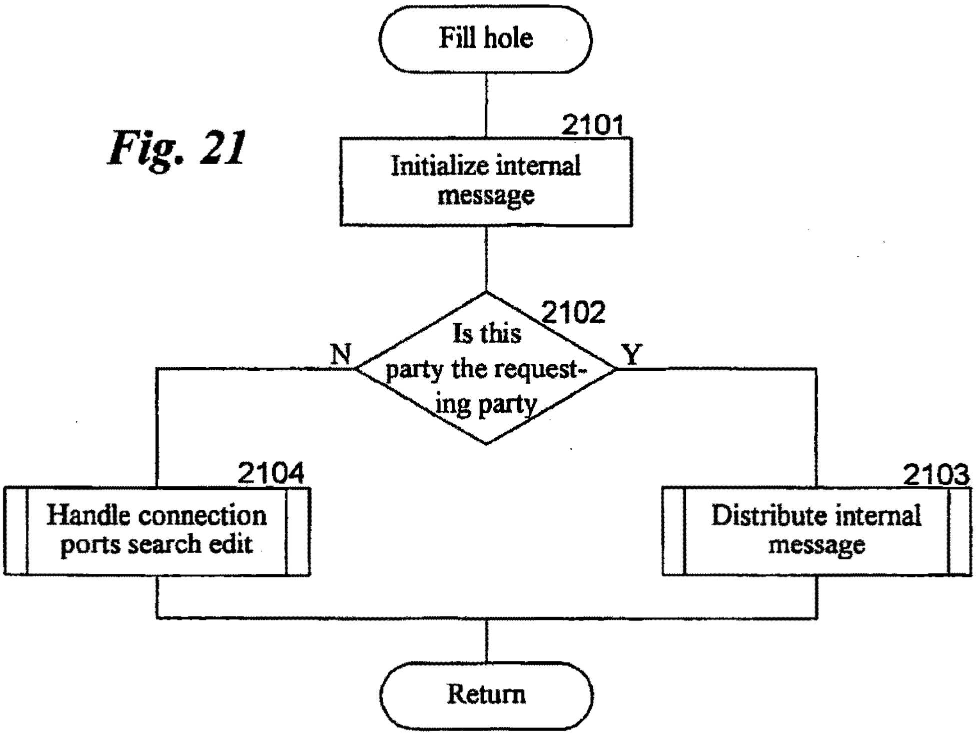

Fill hole
2101
Initialize internal message
2102
Is this party the requesting party
N
Y
2104
Handle connection ports search edit
2103
Distribute internal message
Return

Fig. 22

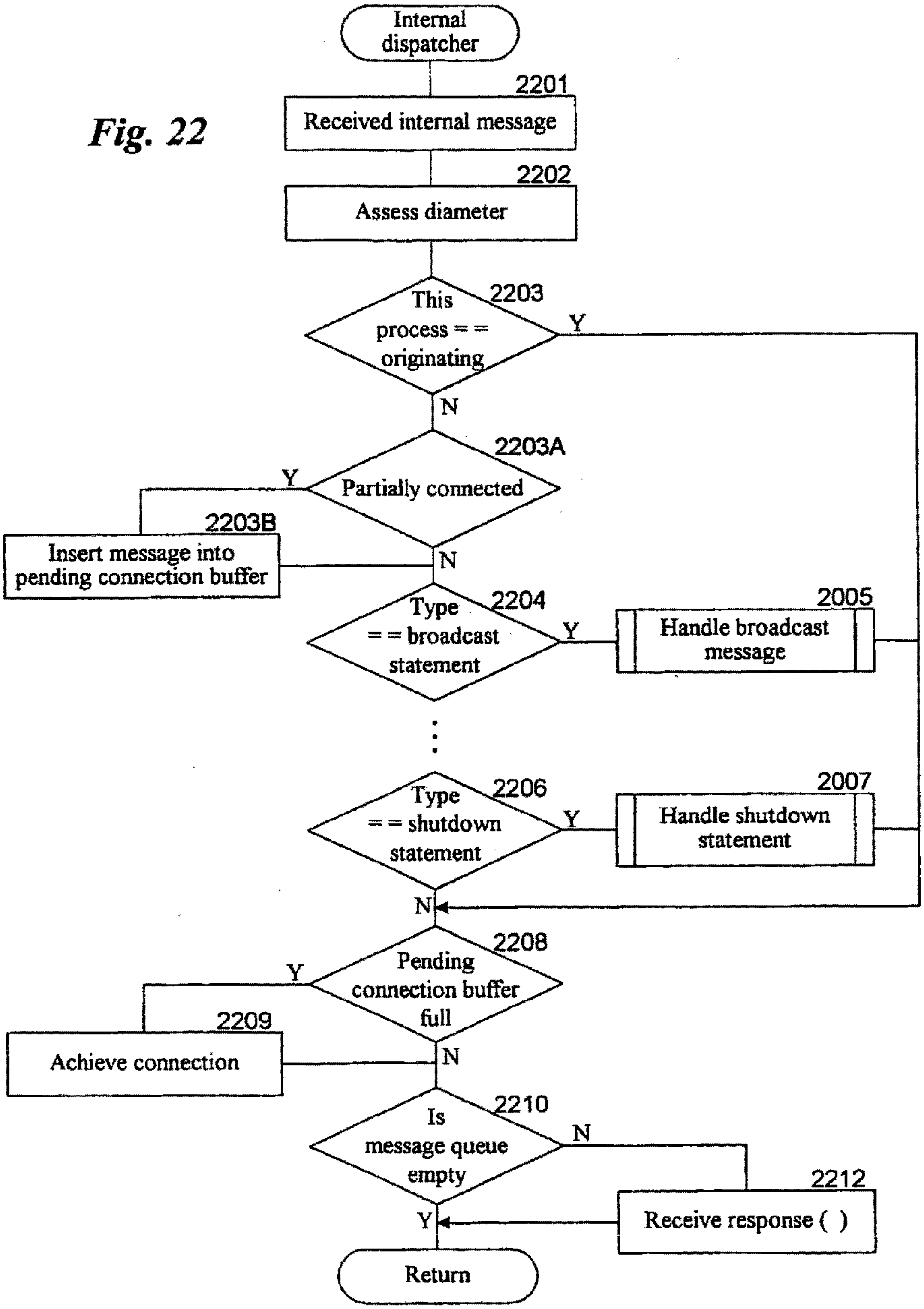

Internal dispatcher
2201
Received internal message
2202
Assess diameter
2203
This process = = originating
Y
N
2203A
Partially connected
Y
2203B
Insert message into pending connection buffer
N
2204
Type = = broadcast statement
Y
2005
Handle broadcast message
2206
Type = = shutdown statement
Y
2007
Handle shutdown statement
N
2208
Pending connection buffer full
Y
2209
Achieve connection
N
2210
Is message queue empty
N
2212
Receive response ( )
Y
Return

Fig. 23

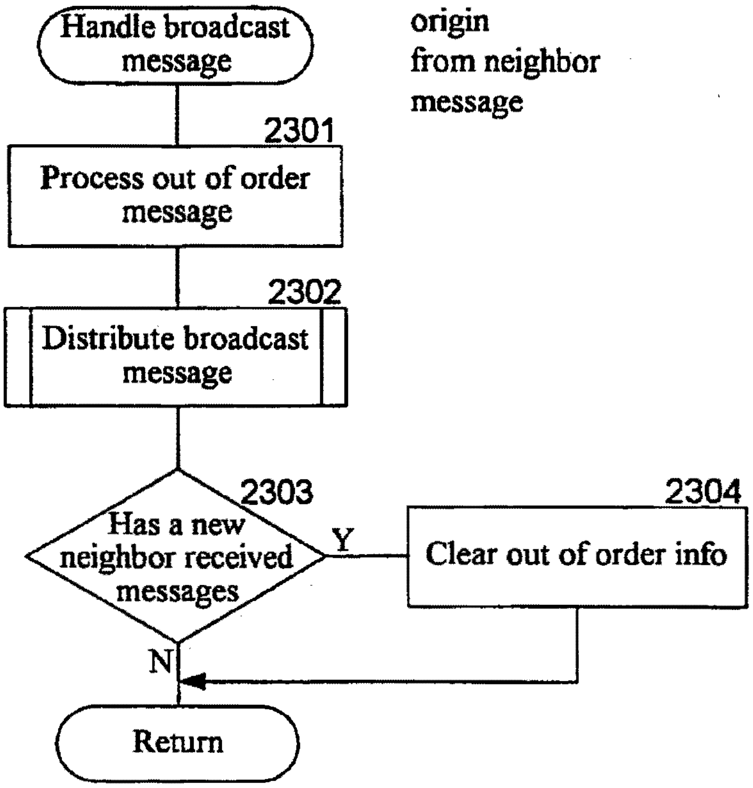
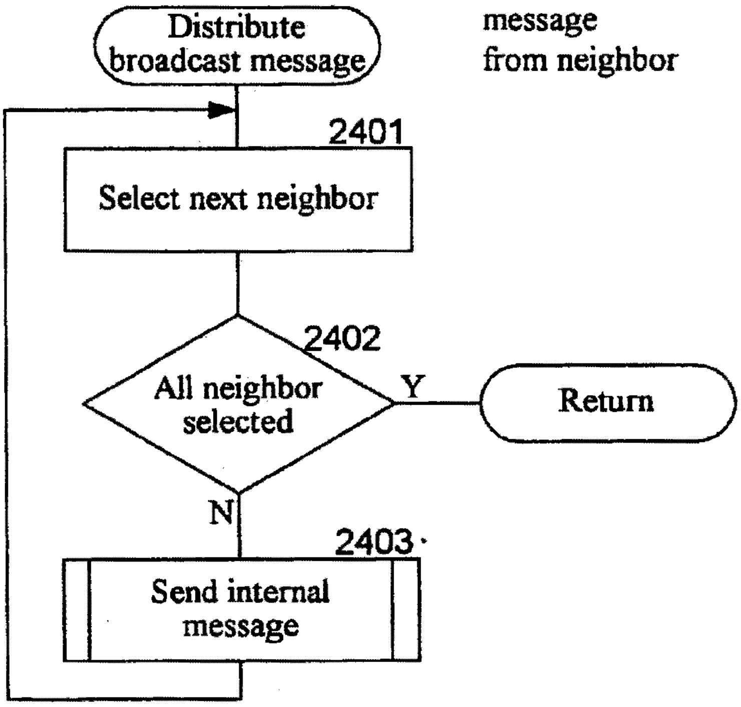

Handle broadcast message
origin
from neighbor
message
2301
Process out of order message
2302
Distribute broadcast message
2303
Has a new neighbor received messages
Y
2304
Clear out of order info
N
Return

Fig. 24


Distribute broadcast message
message from neighbor
2401
Select next neighbor
2402
All neighbor selected
Y
Return
N
2403
Send internal message

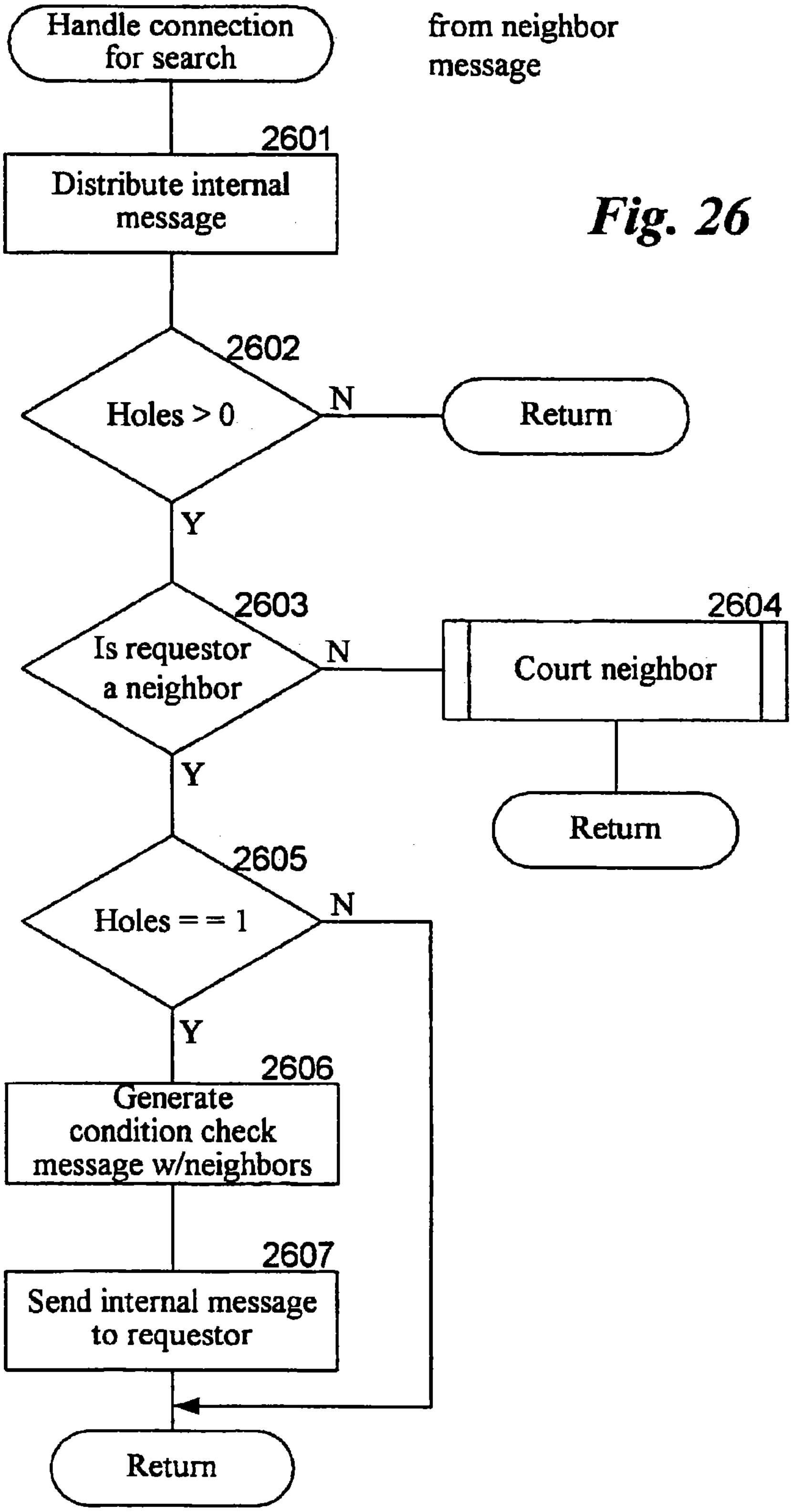

Handle connection for search
from neighbor message
2601
Distribute internal message
2602
Holes > 0
N
Return
Y
2603
Is requestor a neighbor
N
2604
Court neighbor
Return
Y
2605
Holes = = 1
N
Y
2606
Generate condition check message w/neighbors
2607
Send internal message to requestor
Return

Fig. 26

Fig. 27

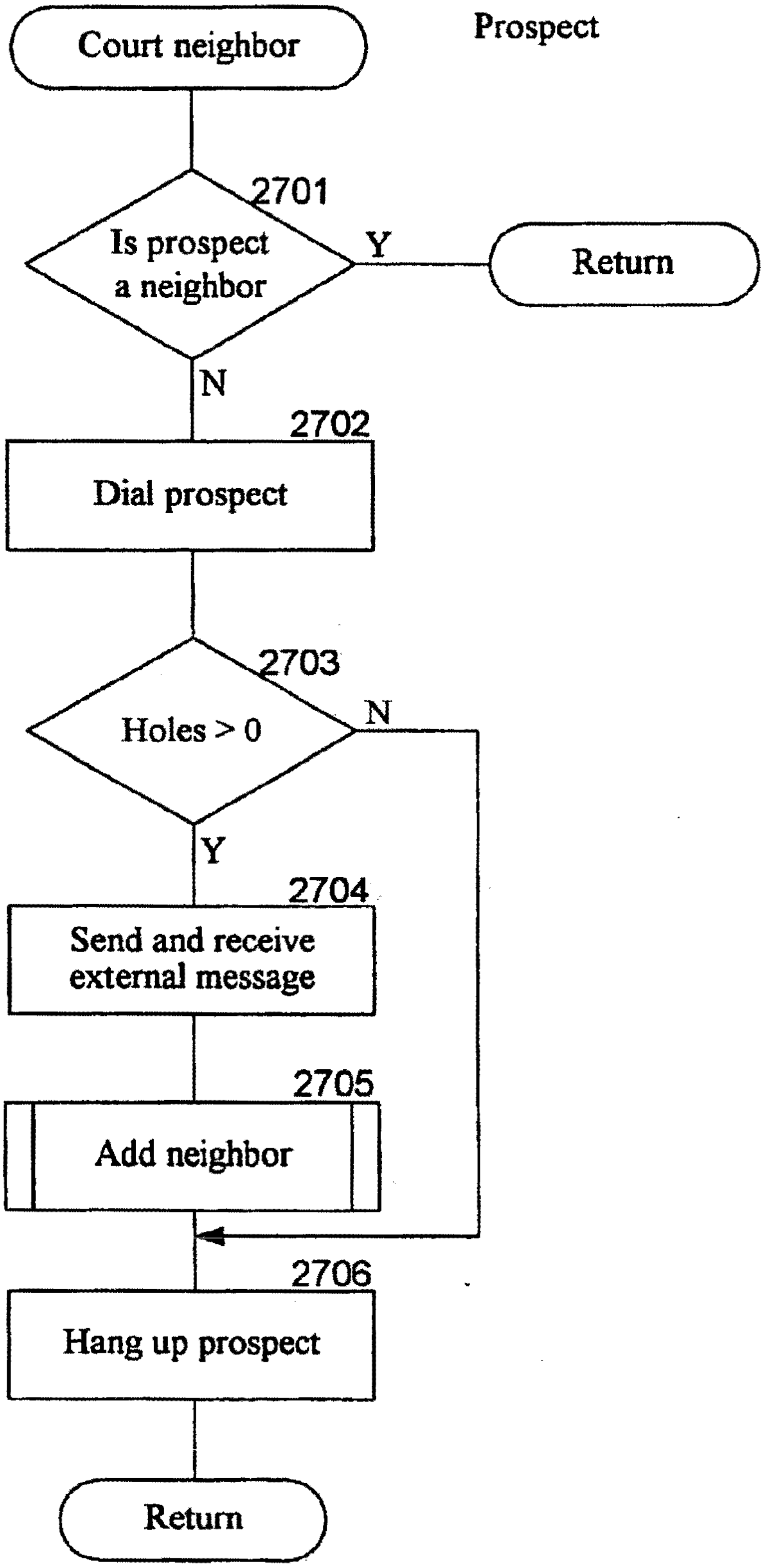

Court neighbor
Prospect
2701
Is prospect a neighbor
Y
Return
N
2702
Dial prospect
2703
Holes > 0
N
Y
2704
Send and receive external message
2705
Add neighbor
2706
Hang up prospect
Return

Fig. 28

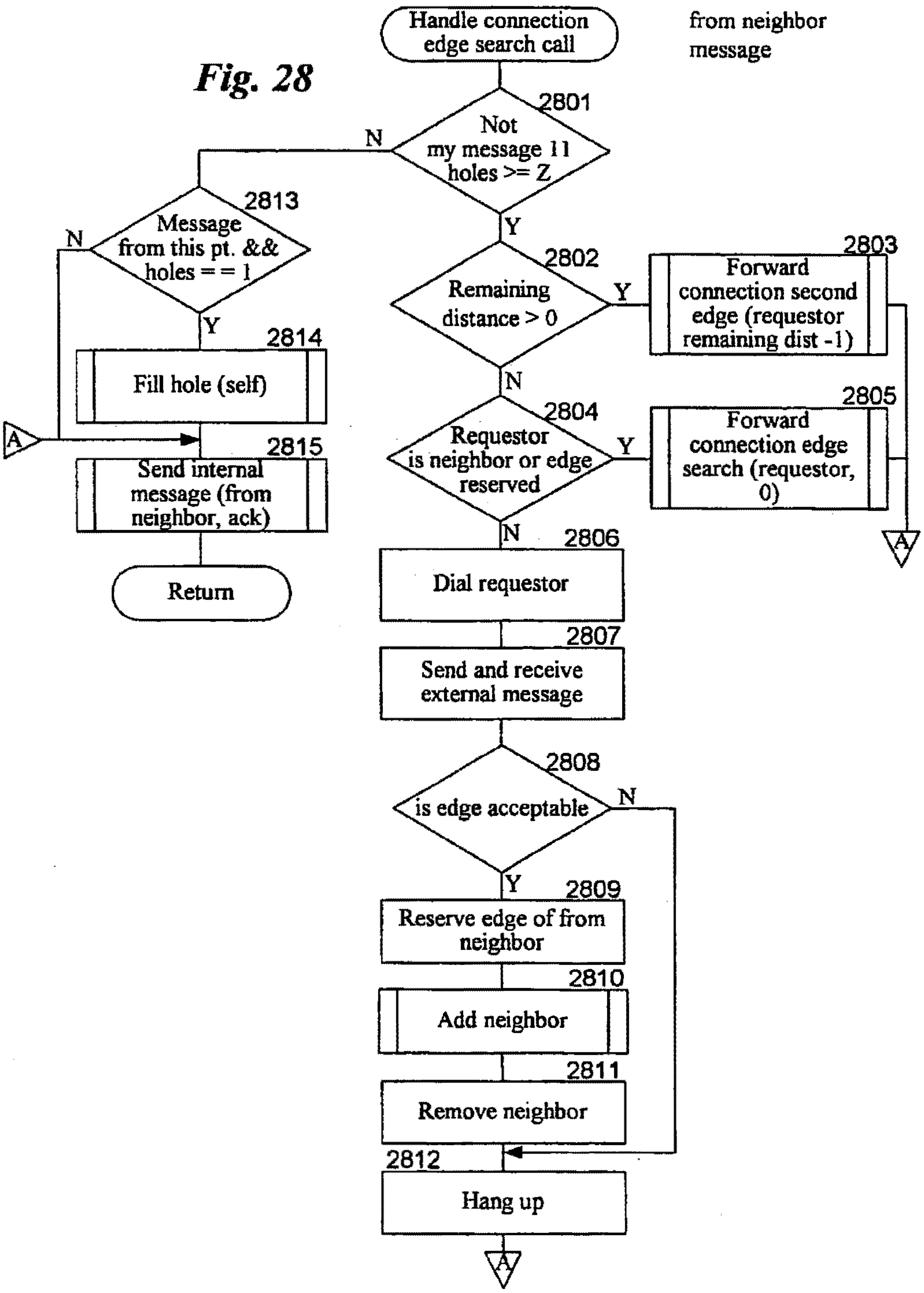
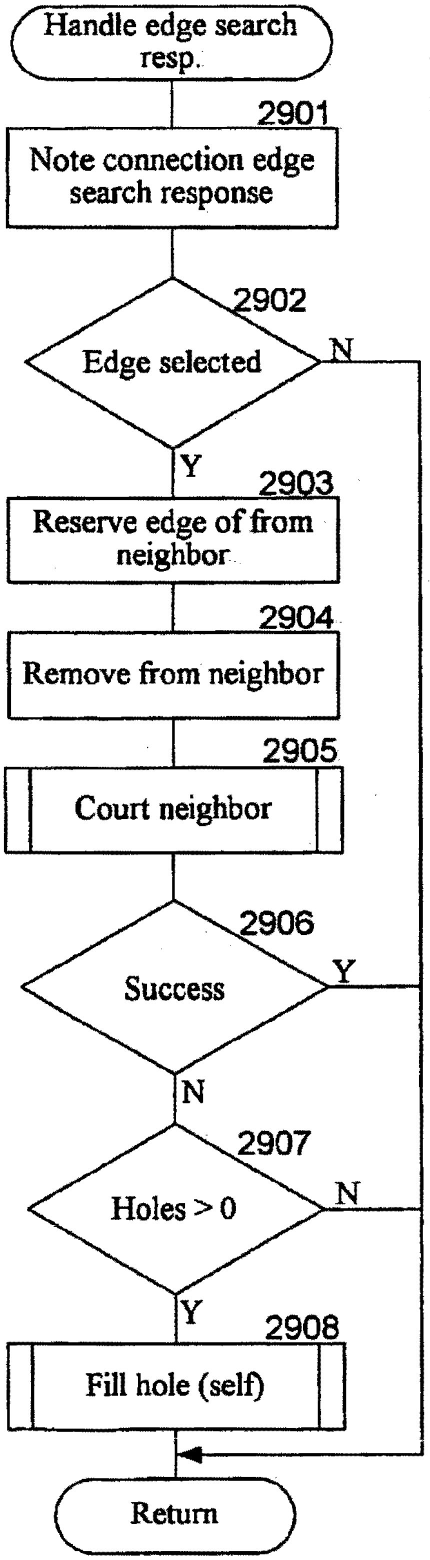
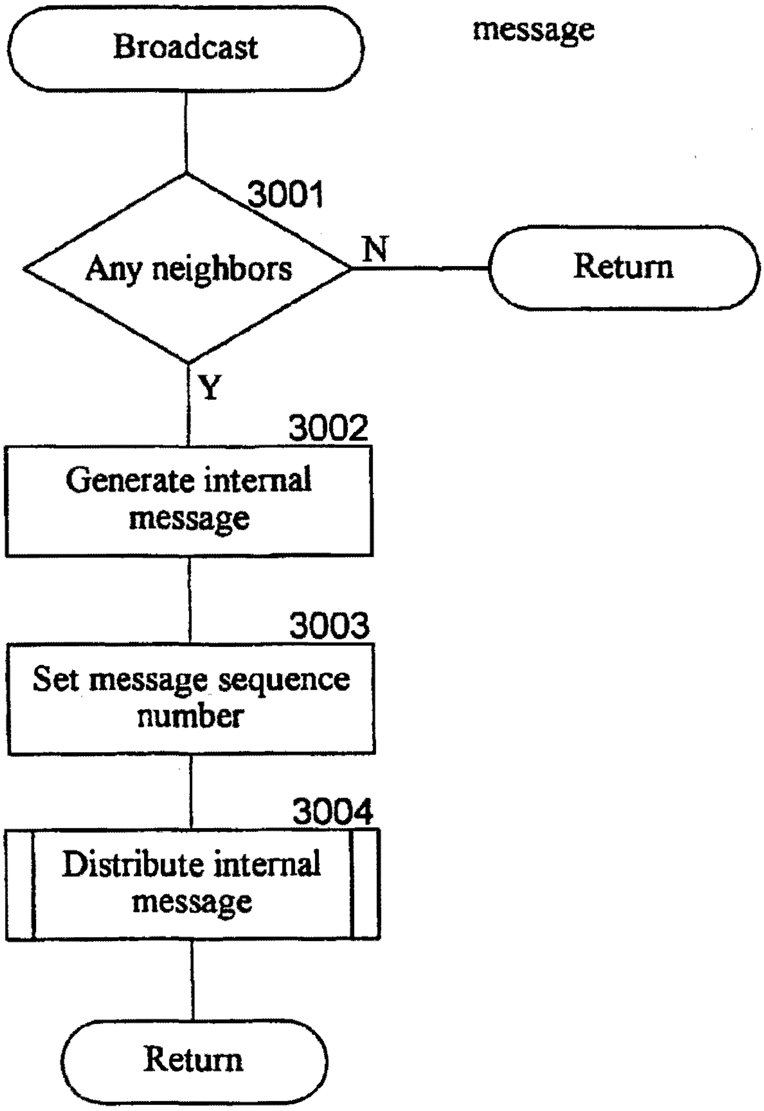

Handle connection edge search call
from neighbor message
2801
Not my message 11 holes >= Z
N
Y
2802
Remaining distance > 0
Y
2803
Forward connection second edge (requestor remaining dist -1)
N
2804
Requestor is neighbor or edge reserved
Y
2805
Forward connection edge search (requestor, 0)
A
N
2806
Dial requestor
2807
Send and receive external message
2808
is edge acceptable
N
Y
2809
Reserve edge of from neighbor
2810
Add neighbor
2811
Remove neighbor
2812
Hang up
A
2813
Message from this pt. && holes = = 1
N
Y
2814
Fill hole (self)
A
2815
Send internal message (from neighbor, ack)
Return

Fig. 29



origin
from neighbor
message

Fig. 30


Broadcast
message
3001
Any neighbors
N
Return
Y
3002
Generate internal message
3003
Set message sequence number
3004
Distribute internal message
Return

message

Fig. 31

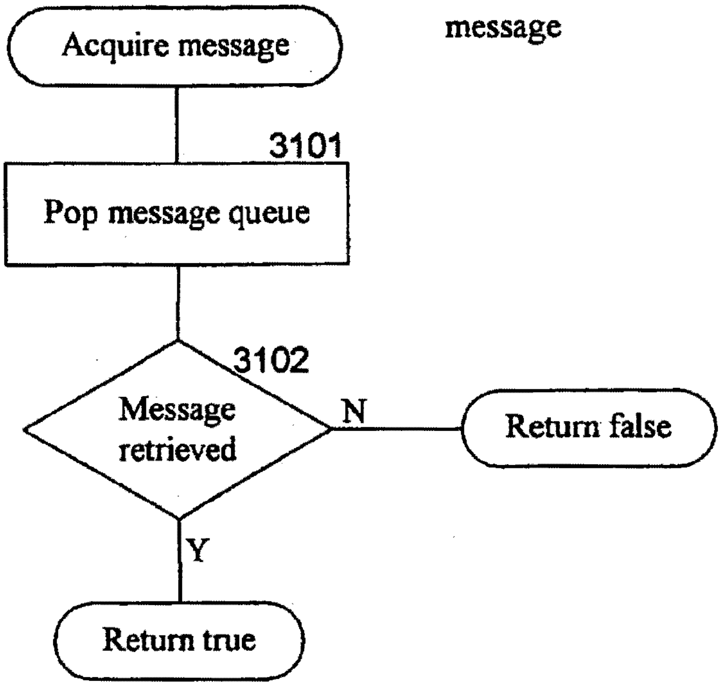


Fig. 32

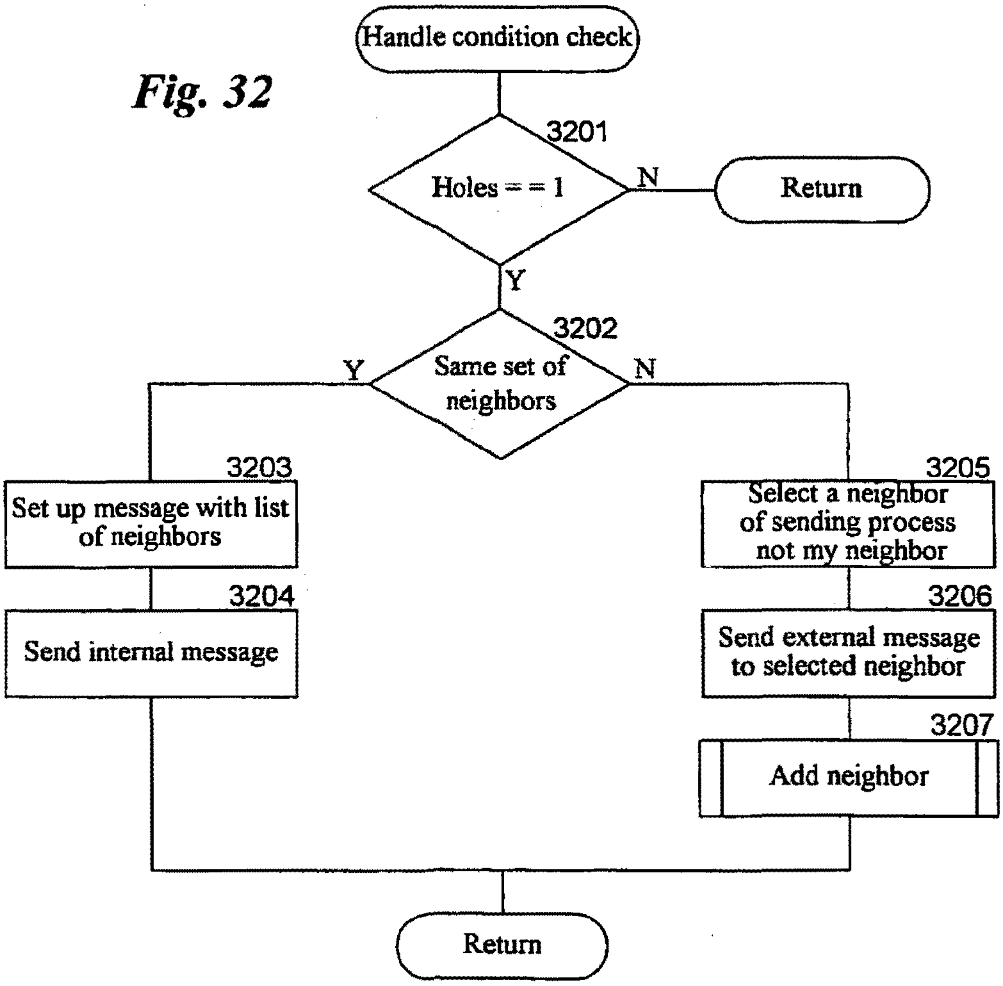
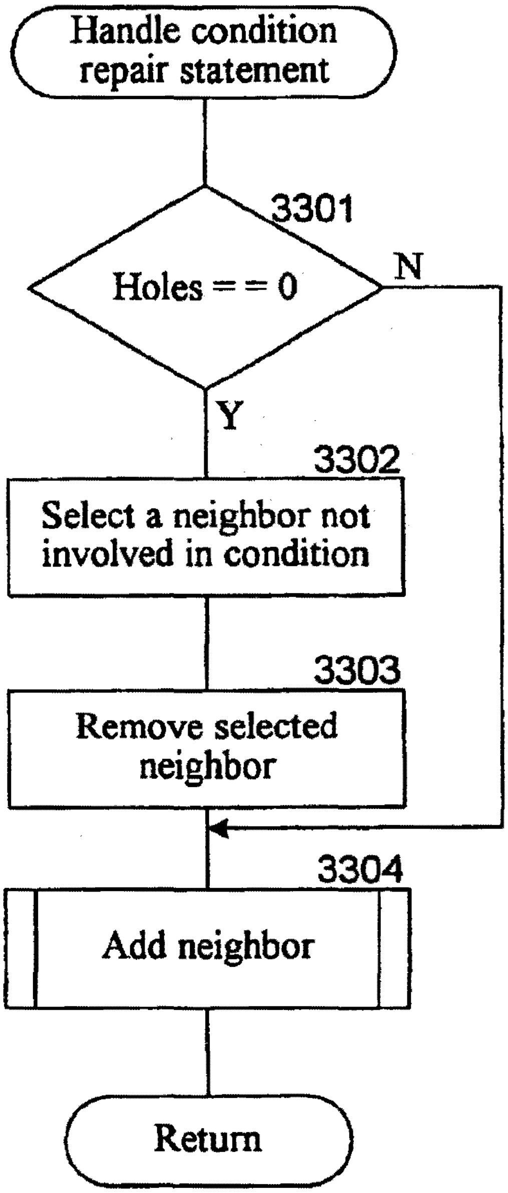

Handle condition check
3201
Holes = = 1
N
Return
Y
3202
Same set of neighbors
Y
N
3203
Set up message with list of neighbors
3204
Send internal message
3205
Select a neighbor of sending process not my neighbor
3206
Send external message to selected neighbor
3207
Add neighbor
Return

Fig. 33


Handle condition repair statement
3301
Holes = = 0
N
Y
3302
Select a neighbor not involved in condition
3303
Remove selected neighbor
3304
Add neighbor
Return

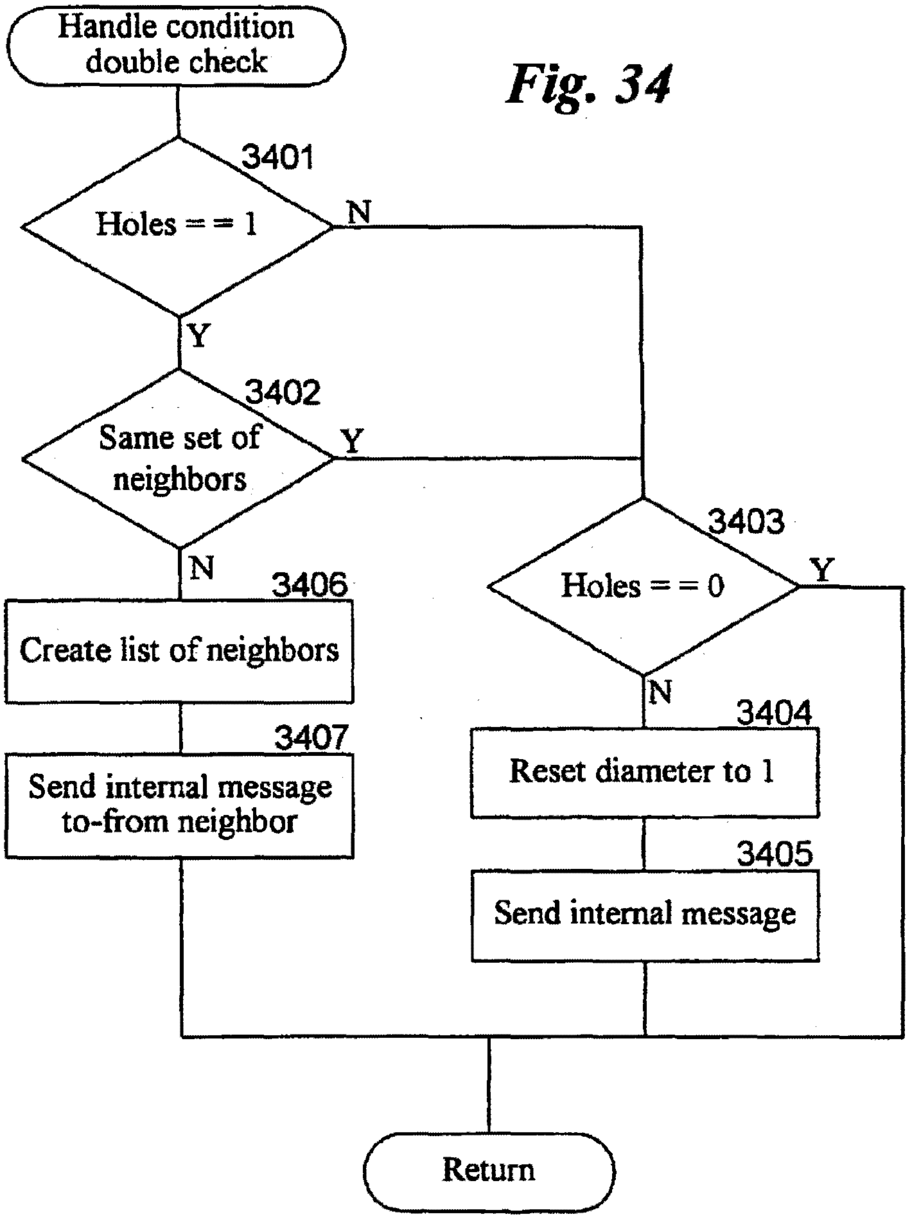

Handle condition double check
3401
Holes = = 1
N
Y
3402
Same set of neighbors
Y
N
3406
Create list of neighbors
3407
Send internal message to-from neighbor
3403
Holes = = 0
Y
N
3404
Reset diameter to 1
3405
Send internal message
Return

Fig. 34

LEAVING A BROADCAST CHANNEL

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is related to U.S. patent application Ser. No. 09/629,576, entitled "BROADCASTING NETWORK," filed on Jul. 31, 2000; U.S. patent application Ser. No. 09/629,570, entitled "JOINING A BROADCAST CHANNEL," filed on Jul. 31, 2000; U.S. patent application Ser. No. 09/629,577, "LEAVING A BROADCAST CHANNEL," filed on Jul. 31, 2000; U.S. patent application Ser. No. 09/629,575, entitled "BROADCASTING ON A BROADCAST CHANNEL," filed on Jul. 31, 2000; U.S. patent application Ser. No. 09/629,572, entitled "CONTACTING A BROADCAST CHANNEL," filed on Jul. 31, 2000; U.S. patent application Ser. No. 09/629,023, entitled "DISTRIBUTED AUCTION SYSTEM," filed on Jul. 31, 2000; U.S. patent application Ser. No. 09/629,043, entitled "AN INFORMATION DELIVERY SERVICE," filed on Jul. 31, 2000; U.S. patent application Ser. No. 09/629,024, entitled "DISTRIBUTED CONFERENCING SYSTEM," filed on Jul. 31, 2000; and U.S. patent application Ser. No. 09/629,042, entitled "DISTRIBUTED GAME ENVIRONMENT," filed on Jul. 31, 2000, the disclosures of which are incorporated herein by reference.

TECHNICAL FIELD

The described technology relates generally to a computer network and more particularly, to a broadcast channel for a subset of a computers of an underlying network.

BACKGROUND

There are a wide variety of computer network communications techniques such as point-to-point network protocols, client/server middleware, multicasting network protocols, and peer-to-peer middleware. Each of these communications techniques have their advantages and disadvantages, but none is particularly well suited to the simultaneous sharing of information among computers that are widely distributed. For example, collaborative processing applications, such as a network meeting programs, have a need to distribute information in a timely manner to all participants who may be geographically distributed.

The point-to-point network protocols, such as UNIX pipes, TCP/IP, and UDP, allow processes on different computers to communicate via point-to-point connections. The interconnection of all participants using point-to-point connections, while theoretically possible, does not scale well as a number of participants grows. For example, each participating process would need to manage its direct connections to all other participating processes. Programmers, however, find it very difficult to manage single connections, and management of multiple connections is much more complex. In addition, participating processes may be limited to the number of direct connections that they can support. This limits the number of possible participants in the sharing of information.

The client/server middleware systems provide a server that coordinates the communications between the various clients who are sharing the information. The server functions as a central authority for controlling access to shared resources. Examples of client/server middleware systems include remote procedure calls ("RPC"), database servers, and the common object request broker architecture ("CORBA"). Client/server middleware systems are not particularly well suited to sharing of information among many participants. In particular, when a client stores information to be shared at the server, each other client would need to poll the server to determine that new information is being shared. Such polling places a very high overhead on the communications network. Alternatively, each client may register a callback with the server, which the server then invokes when new information is available to be shared. Such a callback technique presents a performance bottleneck because a single server needs to call back to each client whenever new information is to be shared. In addition, the reliability of the entire sharing of information depends upon the reliability of the single server. Thus, a failure at a single computer (i.e., the server) would prevent communications between any of the clients.

The multicasting network protocols allow the sending of broadcast messages to multiple recipients of a network. The current implementations of such multicasting network protocols tend to place an unacceptable overhead on the underlying network. For example, UDP multicasting would swamp the Internet when trying to locate all possible participants. IP multicasting has other problems that include needing special-purpose infrastructure (e.g., routers) to support the sharing of information efficiently.

The peer-to-peer middleware communications systems rely on a multicasting network protocol or a graph of point-to-point network protocols. Such peer-to-peer middleware is provided by the T.120 Internet standard, which is used in such products as Data Connection's D.C.-share and Microsoft's NetMeeting. These peer-to-peer middleware systems rely upon a user to assemble a point-to-point graph of the connections used for sharing the information. Thus, it is neither suitable nor desirable to use peer-to-peer middleware systems when more than a small number of participants is desired. In addition, the underlying architecture of the T.120 Internet standard is a tree structure, which relies on the root node of the tree for reliability of the entire network. That is, each message must pass through the root node in order to be received by all participants.

It would be desirable to have a reliable communications network that is suitable for the simultaneous sharing of information among a large number of the processes that are widely distributed.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates a graph that is 4-regular and 4-connected which represents a broadcast channel.

FIG. 2 illustrates a graph representing 20 computers connected to a broadcast channel.

FIGS. 3A and 3B illustrate the process of connecting a new computer Z to the broadcast channel.

FIG. 4A illustrates the broadcast channel of FIG. 1 with an added computer.

FIG. 4B illustrates the broadcast channel of FIG. 4A with an added computer.

FIG. 4C also illustrates the broadcast channel of FIG. 4A with an added computer.

FIG. 5A illustrates the disconnecting of a computer from the broadcast channel in a planned manner.

FIG. 5B illustrates the disconnecting of a computer from the broadcast channel in an unplanned manner.

FIG. 5C illustrates the neighbors with empty ports condition.

FIG. 5D illustrates two computers that are not neighbors who now have empty ports.

FIG. 5E illustrates the neighbors with empty ports condition in the small regime.

FIG. 5F illustrates the situation of FIG. 5E when in the large regime.

FIG. 6 is a block diagram illustrating components of a computer that is connected to a broadcast channel.

FIG. 7 is a block diagram illustrating the sub-components of the broadcaster component in one embodiment.

FIG. 8 is a flow diagram illustrating the processing of the connect routine in one embodiment.

FIG. 9 is a flow diagram illustrating the processing of the seek portal computer routine in one embodiment.

FIG. 10 is a flow diagram illustrating the processing of the contact process routine in one embodiment.

FIG. 11 is a flow diagram illustrating the processing of the connect request routine in one embodiment.

FIG. 12 is a flow diagram of the processing of the check for external call routine in one embodiment.

FIG. 13 is a flow diagram of the processing of the achieve connection routine in one embodiment.

FIG. 14 is a flow diagram illustrating the processing of the external dispatcher routine in one embodiment.

FIG. 15 is a flow diagram illustrating the processing of the handle seeking connection call routine in one embodiment.

FIG. 16 is a flow diagram illustrating processing of the handle connection request call routine in one embodiment.

FIG. 17 is a flow diagram illustrating the processing of the add neighbor routine in one embodiment.

FIG. 18 is a flow diagram illustrating the processing of the forward connection edge search routine in one embodiment.

FIG. 19 is a flow diagram illustrating the processing of the handle edge proposal call routine.

FIG. 20 is a flow diagram illustrating the processing of the handle port connection call routine in one embodiment.

FIG. 21 is a flow diagram illustrating the processing of the fill hole routine in one embodiment.

FIG. 22 is a flow diagram illustrating the processing of the internal dispatcher routine in one embodiment.

FIG. 23 is a flow diagram illustrating the processing of the handle broadcast message routine in one embodiment.

FIG. 24 is a flow diagram illustrating the processing of the distribute broadcast message routine in one embodiment.

FIG. 26 is a flow diagram illustrating the processing of the handle connection port search statement routine in one embodiment.

FIG. 27 is a flow diagram illustrating the processing of the court neighbor routine in one embodiment.

FIG. 28 is a flow diagram illustrating the processing of the handle connection edge search call routine in one embodiment.

FIG. 29 is a flow diagram illustrating the processing of the handle connection edge search response routine in one embodiment.

FIG. 30 is a flow diagram illustrating the processing of the broadcast routine in one embodiment.

FIG. 31 is a flow diagram illustrating the processing of the acquire message routine in one embodiment.

FIG. 32 is a flow diagram illustrating processing of the handle condition check message in one embodiment.

FIG. 33 is a flow diagram illustrating processing of the handle condition repair statement routine in one embodiment.

FIG. 34 is a flow diagram illustrating the processing of the handle condition double check routine.

DETAILED DESCRIPTION

A broadcast technique in which a broadcast channel overlays a point-to-point communications network is provided. The broadcasting of a message over the broadcast channel is effectively a multicast to those computers of the network that are currently connected to the broadcast channel. In one embodiment, the broadcast technique provides a logical broadcast channel to which host computers through their executing processes can be connected. Each computer that is connected to the broadcast channel can broadcast messages onto and receive messages off of the broadcast channel. Each computer that is connected to the broadcast channel receives all messages that are broadcast while it is connected. The logical broadcast channel is implemented using an underlying network system (e.g., the Internet) that allows each computer connected to the underlying network system to send messages to each other connected computer using each computer's address. Thus, the broadcast technique effectively provides a broadcast channel using an underlying network system that sends messages on a point-to-point basis.

The broadcast technique overlays the underlying network system with a graph of point-to-point connections (i.e., edges) between host computers (i.e., nodes) through which the broadcast channel is implemented. In one embodiment, each computer is connected to four other computers, referred to as neighbors. (Actually, a process executing on a computer is connected to four other processes executing on this or four other computers.) To broadcast a message, the originating computer sends the message to each of its neighbors using its point-to-point connections. Each computer that receives the message then sends the message to its three other neighbors using the point-to-point connections. In this way, the message is propagated to each computer using the underlying network to effect the broadcasting of the message to each computer over a logical broadcast channel. A graph in which each node is connected to four other nodes is referred to as a 4-regular graph. The use of a 4-regular graph means that a computer would become disconnected from the broadcast channel only if all four of the connections to its neighbors fail. The graph used by the broadcast technique also has the property that it would take a failure of four computers to divide the graph into disjoint sub-graphs, that is two separate broadcast channels. This property is referred to as being 4-connected. Thus, the graph is both 4-regular and 4-connected.

FIG. 1 illustrates a graph that is 4-regular and 4-connected which represents the broadcast channel. Each of the nine nodes A–I represents a computer that is connected to the broadcast channel, and each of the edges represents an "edge" connection between two computers of the broadcast channel. The time it takes to broadcast a message to each computer on the broadcast channel depends on the speed of the connections between the computers and the number of connections between the originating computer and each other computer on the broadcast channel. The minimum number of connections that a message would need to traverse between each pair of computers is the "distance" between the computers (i.e., the shortest path between the two nodes of the graph). For example, the distance between computers A and F is one because computer A is directly connected to computer F. The distance between computers A and B is two because there is no direct connection between computers A and B, but computer F is directly connected to computer B. Thus, a message originating at computer A would be sent directly to computer F, and then sent from computer F to computer B. The maximum of the distances between the computers is the "diameter" of broadcast channel. The diameter of the broadcast channel represented by FIG. **1** is two. That is, a message sent by any computer would traverse no more than two connections to reach every other computer. FIG. **2** illustrates a graph representing 20 computers connected to a-broadcast channel. The diameter of this broadcast channel is 4. In particular, the shortest path between computers **1** and **3** contains four connections (**1–12, 12–15, 15–18**, and **18–3**).

The broadcast technique includes (1) the connecting of computers to the broadcast channel (i.e., composing the graph), (2) the broadcasting of messages over the broadcast channel (i.e., broadcasting through the graph), and (3) the disconnecting of computers from the broadcast channel (i.e., decomposing the graph) composing the graph.

Composing the Graph

To connect to the broadcast channel, the computer seeking the connection first locates a computer that is currently fully connected to the broadcast channel and then establishes a connection with four of the computers that are already connected to the broadcast channel. (This assumes that there are at least four computers already connected to the broadcast channel. When there are fewer than five computers connected, the broadcast channel cannot be a 4-regular graph. In such a case, the broadcast channel is considered to be in a "small regime." The broadcast technique for the small regime is described below in detail. When five or more computers are connected, the broadcast channel is considered to be in the "large regime." This description assumes that the broadcast channel is in the large regime, unless specified otherwise.) Thus, the process of connecting to the broadcast channel includes locating the broadcast channel, identifying the neighbors for the connecting computer, and then connecting to each identified neighbor. Each computer is aware of one or more "portal computers" through which that computer may locate the broadcast channel. A seeking computer locates the broadcast channel by contacting the portal computers until it finds one that is currently fully connected to the broadcast channel. The found portal computer then directs the identifying of four computers (i.e., to be the seeking computer's neighbors) to which the seeking computer is to connect. Each of these four computers then cooperates with the seeking computer to effect the connecting of the seeking computer to the broadcast channel. A computer that has started the process of locating a portal computer, but does not yet have a neighbor, is in the "seeking connection state." A computer that is connected to at least one neighbor, but not yet four neighbors, is in the "partially connected state." A computer that is currently, or has been, previously connected to four neighbors is in the "fully connected state."

Since the broadcast channel is a 4-regular graph, each of the identified computers is already connected to four computers. Thus, some connections between computers need to be broken so that the seeking computer can connect to four computers. In one embodiment, the broadcast technique identifies two pairs of computers that are currently connected to each other. Each of these pairs of computers breaks the connection between them, and then each of the four computers (two from each pair) connects to the seeking computer. FIGS. **3**A and **3**B illustrate the process of a new computer Z connecting to the broadcast channel. FIG. **3**A illustrates the broadcast channel before computer Z is connected. The pairs of computers B and E and computers C and D are the two pairs that are identified as the neighbors for the new computer Z. The connections between each of these pairs is broken, and a connection between computer Z and each of computers B, C, D, and E is established as indicated by FIG. **3**B. The process of breaking the connection between two neighbors and reconnecting each of the former neighbors to another computer is referred to as "edge pinning" as the edge between two nodes may be considered to be stretched and pinned to a new node.

Each computer connected to the broadcast channel allocates five communications ports for communicating with other computers. Four of the ports are referred to as "internal" ports because they are the ports through which the messages of the broadcast channels are sent. The connections between internal ports of neighbors are referred to as "internal" connections. Thus, the internal connections of the broadcast channel form the 4-regular and 4-connected graph. The fifth port is referred to as an "external" port because it is used for sending non-broadcast messages between two computers. Neighbors can send non-broadcast messages either through their internal ports of their connection or through their external ports. A seeking computer uses external ports when locating a portal computer.

In one embodiment, the broadcast technique establishes the computer connections using the TCP/IP communications protocol, which is a point-to-point protocol, as the underlying network. The TCP/IP protocol provides for reliable and ordered delivery of messages between computers. The TCP/IP protocol provides each computer with a "port space" that is shared among all the processes that may execute on-that computer. The ports are identified by numbers from 0 to 65,535. The first 2056 ports are reserved for specific applications (e.g., port **80** for HTTP messages). The remainder of the ports are user ports that are available to any process. In one embodiment, a set of port numbers can be reserved for use by the computer connected to the broadcast channel. In an alternative embodiment, the port numbers used are dynamically identified by each computer. Each computer dynamically identifies an available port to be used as its call-in port. This call-in port is used to establish connections with the external port and the internal ports. Each computer that is connected to the broadcast channel can receive non-broadcast messages through its external port. A seeking computer tries "dialing" the port numbers of the portal computers until a portal computer "answers," a call on its call-in port. A portal computer answers when it is connected to or attempting to connect to the broadcast channel and its call-in port is dialed. (In this description, a telephone metaphor is used to describe the connections.) When a computer receives a call on its call-in port, it transfers the call to another port. Thus, the seeking computer actually communicates through that transfer-to port, which is the external port. The call is transferred so that other computers can place calls to that computer via the call-in port. The seeking computer then communicates via that external port to request the portal computer to assist in connecting the seeking computer to the broadcast channel. The seeking computer could identify the call-in port number of a portal computer by successively dialing each port in port number order. As discussed below in detail, the broadcast technique uses a hashing algorithm to select the port number order, which may result in improved performance.

A seeking computer could connect to the broadcast channel by connecting to computers either directly connected to the found portal computer or directly connected to one of its neighbors. A possible problem with such a scheme for identifying the neighbors for the seeking computer is that the diameter of the broadcast channel may increase when each seeking computer uses the same found portal computer and establishes a connection to the broadcast channel directly through that found portal computer. Conceptually, the graph becomes elongated in the direction of where the new nodes are added. FIGS. 4A–4C illustrate that possible problem. FIG. 4A illustrates the broadcast channel of FIG. 1 with an added computer. Computer J was connected to the broadcast channel by edge pinning edges C–D and E–H to computer J. The diameter of this broadcast channel is still two. FIG. 4B illustrates the broadcast channel of FIG. 4A with an added computer. Computer K was connected to the broadcast channel by edge pinning edges E–J and B–C to computer K. The diameter of this broadcast channel is three, because the shortest path from computer G to computer K is through edges G–A, A–E, and E–K. FIG. 4C also illustrates the broadcast channel of FIG. 4A with an added computer. Computer K was connected to the broadcast channel by edge pinning edges D–G and E–J to computer K. The diameter of this broadcast channel is, however, still two. Thus, the selection of neighbors impacts the diameter of the broadcast channel. To help minimize the diameter, the broadcast technique uses a random selection technique to identify the four neighbors of a computer in the seeking connection state. The random selection technique tends to distribute the connections to new seeking computers throughout the computers of the broadcast channel which may result in smaller overall diameters.

Broadcasting Through the Graph

As described above, each computer that is connected to the broadcast channel can broadcast messages onto the broadcast channel and does receive all messages that are broadcast on the broadcast channel. The computer that originates a message to be broadcast sends that message to each of its four neighbors using the internal connections. When a computer receives a broadcast message from a neighbor, it sends the message to its three other neighbors. Each computer on the broadcast channel, except the originating computer, will thus receive a copy of each broadcast message from each of its four neighbors. Each computer, however, only sends the first copy of the message that it receives to its neighbors and disregards subsequently received copies. Thus, the total number of copies of a message that is sent between the computers is 3N+1, where N is the number of computers connected to the broadcast channel. Each computer sends three copies of the message, except for the originating computer, which sends four copies of the message.

The redundancy of the message sending helps to ensure the overall reliability of the broadcast channel. Since each computer has four connections to the broadcast channel, if one computer fails during the broadcast of a message, its neighbors have three other connections through which they will receive copies of the broadcast message. Also, if the internal connection between two computers is slow, each computer has three other connections through which it may receive a copy of each message sooner.

Each computer that originates a message numbers its own messages sequentially. Because of the dynamic nature of the broadcast channel and because there are many possible connection paths between computers, the messages may be received out of order. For example, the distance between an originating computer and a certain receiving computer may be four. After sending the first message, the originating computer and receiving computer may become neighbors and thus the distance between them changes to one. The first message may have to travel a distance of four to reach the receiving computer. The second message only has to travel a distance of one. Thus, it is possible for the second message to reach the receiving computer before the first message.

When the broadcast channel is in a steady state (i.e., no computers connecting or disconnecting from the broadcast channel), out-of-order messages are not a problem because each computer will eventually receive both messages and can queue messages until all earlier ordered messages are received. If, however, the broadcast channel is not in a steady state, then problems can occur. In particular, a computer may connect to the broadcast channel after the second message has already been received and forwarded on by its new neighbors. When a new neighbor eventually receives the first message, it sends the message to the newly connected computer. Thus, the newly connected computer will receive the first message, but will not receive the second message. If the newly connected computer needs to process the messages in order, it would wait indefinitely for the second message.

One solution to this problem is to have each computer queue all the messages that it receives until it can send them in their proper order to its neighbors. This solution, however, may tend to slow down the propagation of messages through the computers of the broadcast channel. Another solution that may have less impact on the propagation speed is to queue messages only at computers who are neighbors of the newly, connected computers. Each already connected neighbor would forward messages as it receives them to its other neighbors who are not newly connected, but not to the newly connected neighbor. The already connected neighbor would only forward messages from each originating computer to the newly connected computer when it can ensure that no gaps in the messages from that originating computer will occur. In one embodiment, the already connected neighbor may track the highest sequence number of the messages already received and forwarded on from each originating computer. The already connected computer will send only higher numbered messages from the originating computers to the newly connected computer. Once all lower numbered messages have been received from all originating computers, then the already connected computer can treat the newly connected computer as its other neighbors and simply forward each message as it is received. In another embodiment, each computer may queue messages and only forwards to the newly connected computer those messages as the gaps are filled in. For example, a computer might receive messages **4** and **5** and then receive message **3**. In such a case, the already connected computer would forward queue messages **4** and **5**. When message **3** is finally received, the already connected computer will send messages **3**, **4**, and **5** to the newly connected computer. If messages **4** and **5** were sent to the newly connected computer before message **3**, then the newly connected computer would process messages **4** and **5** and disregard message **3**. Because the already connected computer queues messages **4** and **5**, the newly connected computer will be able to process message **3**. It is possible that a newly connected computer will receive a set of messages from an originating computer through one neighbor and then receive another set of message from the same originating computer through another neighbor. If the second set of messages contains a message that is ordered earlier than the messages of the first set received, then the newly connected computer may ignore that earlier ordered message if the computer already processed those later ordered messages.

Decomposing the Graph

A connected computer disconnects from the broadcast channel either in a planned or unplanned manner. When a computer disconnects in a planned manner, it sends a disconnect message to each of its four neighbors. The disconnect message includes a list that identifies the four neighbors of the disconnecting computer. When a neighbor receives the disconnect message, it tries to connect to one of the computers on the list. In one embodiment, the first computer in the list will try to connect to the second computer in the list, and the third computer in the list will try to connect to the fourth computer in the list. If a computer cannot connect (e.g., the first and second computers are already connected), then the computers may try connecting in various other combinations. If connections cannot be established, each computer broadcasts a message that it needs to establish a connection with another computer. When a computer with an available internal port receives the message, it can then establish a connection with the computer that broadcast the message. FIGS. 5A–5D illustrate the disconnecting of a computer from the broadcast channel. FIG. 5A illustrates the disconnecting of a computer from the broadcast channel in a planned manner. When computer H decides to disconnect, it sends its list of neighbors to each of its neighbors (computers A, E, F and I) and then disconnects from each of its neighbors. When computers A and I receive the message they establish a connection between them as indicated by the dashed line, and similarly for computers E and F.

When a computer disconnects in an unplanned manner, such as resulting from a power failure, the neighbors connected to the disconnected computer recognize the disconnection when each attempts to send its next message to the now disconnected computer. Each former neighbor of the disconnected computer recognizes that it is short one connection (i.e., it has a hole or empty port). When a connected computer detects that one of its neighbors is now disconnected, it broadcasts a port connection request on the broadcast channel, which indicates that it has one internal port that needs a connection. The port connection request identifies the call-in port of the requesting computer. When a connected computer that is also short a connection receives the connection request, it communicates with the requesting computer through its external port to establish a connection between the two computers. FIG. 5B illustrates the disconnecting of a computer from the broadcast channel in an unplanned manner. In this illustration, computer H has disconnected in an unplanned manner. When each of its neighbors, computers A, E, F, and I, recognizes the disconnection, each neighbor broadcasts a port connection request indicating that it needs to fill an empty port. As shown by the dashed lines, computers F and I and computers A and E respond to each other's requests and establish a connection.

It is possible that a planned or unplanned disconnection may result in two neighbors each having an empty internal port. In such a case, since they are neighbors, they are already connected and cannot fill their empty ports by connecting to each other. Such a condition is referred to as the "neighbors with empty ports" condition. Each neighbor broadcasts a port connection request when it detects that it has an empty port as described above. When a neighbor receives the port connection request from the other neighbor, it will recognize the condition that its neighbor also has an empty port. Such a condition may also occur when the broadcast channel is in the small regime. The condition can only be corrected when in the large regime. When in the small regime, each computer will have less than four neighbors. To detect this condition in the large regime, which would be a problem if not repaired, the first neighbor to

receive the port connection request recognizes the condition and sends a condition check message to the other neighbor. The condition check message includes a list of the neighbors of the sending computer. When the receiving computer receives the list, it compares the list to its own list of neighbors. If the lists are different, then this condition has occurred in the large regime and repair is needed. To repair this condition, the receiving computer will send a condition repair request to one of the neighbors of the sending computer which is not already a neighbor of the receiving computer. When the computer receives the condition repair request, it disconnects from one of its neighbors (other than the neighbor that is involved with the condition) and connects to the computer that sent the condition repair request. Thus, one of the original neighbors involved in the condition will have had a port filled. However, two computers are still in need of a connection, the other original neighbor and the computer that is now disconnected from the computer that received the condition repair request. Those two computers send out port connection requests. If those two computers are not neighbors, then they will connect to each other when they receive the requests. If, however, the two computers are neighbors, then they repeat the condition repair process until two non-neighbors are in need of connections.

It is possible that the two original neighbors with the condition may have the same set of neighbors. When the neighbor that receives the condition check message determines that the sets of neighbors are the same, it sends a condition double check message to one of its neighbors other than the neighbor who also has the condition. When the computer receives the condition double check message, it determines whether it has the same set of neighbors as the sending computer. If so, the broadcast channel is in the small regime and the condition is not a problem. If the set of neighbors are different, then the computer that received the condition double check message sends a condition check message to the original neighbors with the condition. The computer that receives that condition check message directs one of it neighbors to connect to one of the original neighbors with the condition by sending a condition repair message. Thus, one of the original neighbors with the condition will have its port filled.

FIG. 5C illustrates the neighbors with empty ports condition. In this illustration, computer H disconnected in an unplanned manner, but computers F and I responded to the port connection request of the other and are now connected together. The other former neighbors of computer H, computers A and E, are already neighbors, which gives rise to the neighbors with empty ports condition. In this example, computer E received the port connection request from computer A, recognized the possible condition, and sent (since they are neighbors via the internal connection) a condition check message with a list of its neighbors to computer A. When computer A received the list, it recognized that computer E has a different set of neighbor (i.e., the broadcast channel is in the large regime). Computer A selected computer D, which is a neighbor of computer E and sent it a condition repair request. When computer D received the condition repair request, it disconnected from one of its neighbors (other than computer E), which is computer G in this example. Computer D then connected to computer A. FIG. 5D illustrates two computers that are not neighbors who now have empty ports. Computers E and G now have empty ports and are not currently neighbors. Therefore, computers E and G can connect to each other.

FIGS. 5E and 5F further illustrate the neighbors with empty ports condition. FIG. 5E illustrates the neighbors with empty ports condition in the small regime. In this example, if computer E disconnected in an unplanned manner, then each computer broadcasts a port connection request when it detects the disconnect. When computer A receives the port connection request form computer B, it detects the neighbors with empty ports condition and sends a condition check message to computer B. Computer B recognizes that it has the same set of neighbors (computer C and D) as computer A and then sends a condition double check message to computer C. Computer C recognizes that the broadcast channel is in the small regime because is also has the same set of neighbors as computers A and B, computer C may then broadcast a message indicating that the broadcast channel is in the small regime.

FIG. 5F illustrates the situation of FIG. 5E when in the large regime. As discussed above, computer C receives the condition double check message from computer B. In this case, computer C recognizes that the broadcast channel is in the large regime because it has a set of neighbors that is different from computer B. The edges extending up from computer C and D indicate connections to other computers. Computer C then sends a condition check message to computer B. When computer B receives the condition check message, it sends a condition repair message to one of the neighbors of computer C. The computer that receives the condition repair message disconnects from one of its neighbors, other than computer C, and tries to connect to computer B and the neighbor from which it disconnected tries to connect to computer A.

Port Selection

As described above, the TCP/IP protocol designates ports above number 2056 as user ports. The broadcast technique uses five user port numbers on each computer: one external port and four internal ports. Generally, user ports cannot be statically allocated to an application program because other applications programs executing on the same computer may use conflicting port numbers. As a result, in one embodiment, the computers connected to the broadcast channel dynamically allocate their port numbers. Each computer could simply try to locate the lowest number unused port on that computer and use that port as the call-in port. A seeking computer, however, does not know in advance the call-in port number of the portal computers when the port numbers are dynamically allocated. Thus, a seeking computer needs to dial ports of a portal computer starting with the lowest port number when locating the call-in port of a portal computer. If the portal computer is connected to (or attempting to connect to) the broadcast channel, then the seeking computer would eventually find the call-in port. If the portal computer is not connected, then the seeking computer would eventually dial every user port. In addition, if each application program on a computer tried to allocate low-ordered port numbers, then a portal computer may end up with a high-numbered port for its call-in port because many of the low-ordered port numbers would be used by other application programs. Since the dialing of a port is a relatively slow process, it would take the seeking computer a long time to locate the call-in port of a portal computer. To minimize this time, the broadcast technique uses a port ordering algorithm to identify the port-number order that a portal computer should use when finding an available port for its call-in port. In one embodiment, the broadcast technique uses a hashing algorithm to identify the port-order. The algorithm preferably distributes the ordering of the port numbers randomly through out the user port number space and only selects each port number once. In addition, every time the algorithm is executed on any computer for a given channel type and channel instance, it generates the same port ordering. As described below, it is possible for a computer to be connected to multiple broadcast channels that are uniquely identified by channel type and channel instance. The algorithm may be "seeded" with channel type and channel instance in order to generate a unique ordering of port numbers for each broadcast channel. Thus, a seeking computer will dial the ports of a portal computer in the same order as the portal computer used when allocating its call-in port.

If many computers are at the same time seeking connection to a broadcast channel through a single portal computer, then the ports of the portal computer may be busy when called by seeking computers. The seeking computers would typically need to keep on redialing a busy port. The process of locating a call-in port may be significantly slowed by such redialing. In one embodiment, each seeking computer may each reorder the first few port numbers generated by the hashing algorithm. For example, each seeking computer could randomly reorder the first eight port numbers generated by the hashing algorithm. The random ordering could also be weighted where the first port number generated by the hashing algorithm would have a 50% chance of being first in the reordering, the second port number would have a 25% chance of being first in the reordering, and so on. Because the seeking computers would use different orderings, the likelihood of finding a busy port is reduced. For example, if the first eight port numbers are randomly selected, then it is possible that eight seeking computers could be simultaneously dialing ports in different sequences which would reduce the chances of dialing a busy port.

Locating a Portal Computer

Each computer that can connect to the broadcast channel has a list of one or more portal computers through which it can connect to the broadcast channel. In one embodiment, each computer has the same set of portal computers. A seeking computer locates a portal computer that is connected to the broadcast channel by successively dialing the ports of each portal computer in the order specified by an algorithm. A seeking computer could select the first portal computer and then dial all its ports until a call-in port of a computer that is fully connected to the broadcast channel is found. If no call-in port is found, then the seeking computer would select the next portal computer and repeat the process until a portal computer with such a call-in port is found. A problem with such a seeking technique is that all user ports of each portal computer are dialed until a portal computer fully connected to the broadcast channel is found. In an alternate embodiment, the seeking computer selects a port number according to the algorithm and then dials each portal computer at that port number. If no acceptable call-in port to the broadcast channel is found, then the seeking computer selects the next port number and repeats the process. Since the call-in ports are likely allocated at lower-ordered port numbers, the seeking computer first dials the port numbers that are most likely to be call-in ports of the broadcast channel. The seeking computers may have a maximum search depth, that is the number of ports that it will dial when seeking a portal computer that is fully connected. If the seeking computer exhausts its search depth, then either the broadcast channel has not yet been established or, if the seeking computer is also a portal computer, it can then establish the broadcast channel with itself as the first fully connected computer.

When a seeking computer locates a portal computer that is itself not fully connected, the two computers do not connect when they first locate each other because the broadcast channel may already be established and accessible through a higher-ordered port number on another portal computer. If the two seeking computers were to connect to each other, then two disjoint broadcast channels would be formed. Each seeking computer can share its experience in trying to locate a portal computer with the other seeking computer. In particular, if one seeking computer has searched all the portal computers to a depth of eight, then the one seeking computer can share that it has searched to a depth of eight with another seeking computer. If that other seeking computer has searched to a depth of, for example, only four, it can skip searching through depths five through eight and that other seeking computer can advance its searching to a depth of nine.

In one embodiment, each computer may have a different set of portal computers and a different maximum search depth. In such a situation, it may be possible that two disjoint broadcast channels are formed because a seeking computer cannot locate a fully connected port computer at a higher depth. Similarly, if the set of portal computers are disjoint, then two separate broadcast channels would be formed.

Identifying Neighbors for a Seeking Computer

As described above, the neighbors of a newly connecting computer are preferably selected randomly from the set of currently connected computers. One advantage of the broadcast channel, however, is that no computer has global knowledge of the broadcast channel. Rather, each computer has local knowledge of itself and its neighbors. This limited local knowledge has the advantage that all the connected computers are peers (as far as the broadcasting is concerned) and the failure of any one computer (actually any three computers when in the 4-regular and 4-connect form) will not cause the broadcast channel to fail. This local knowledge makes it difficult for a portal computer to randomly select four neighbors for a seeking computer.

To select the four computers, a portal computer sends an edge connection request message through one of its internal connections that is randomly selected. The receiving computer again sends the edge connection request message through one of its internal connections that is randomly selected. This sending of the message corresponds to a random walk through the graph that represents the broadcast channel. Eventually, a receiving computer will decide that the message has traveled far enough to represent a randomly selected computer. That receiving computer will offer the internal connection upon which it received the edge connection request message to the seeking computer for edge pinning. Of course, if either of the computers at the end of the offered internal connection are already neighbors of the seeking computer, then the seeking computer-cannot connect through that internal connection. The computer that decided that the message has traveled far enough will detect this condition of already being a neighbor and send the message to a randomly selected neighbor.

In one embodiment, the distance that the edge connection request message travels is established by the portal computer to be approximately twice the estimated diameter of the broadcast channel. The message includes an indication of the distance that it is to travel. Each receiving computer decrements that distance to travel before sending the message on. The computer that receives a message with a distance to travel that is zero is considered to be the randomly selected computer. If that randomly selected computer cannot connect to the seeking computer (e.g., because it is already connected to it), then that randomly selected computer forwards the edge connection request to onc of its neighbors with a new distance to travel. In one embodiment, the forwarding computer toggles the new distance to travel between zero and one to help prevent two computers from sending the message back and forth between each other.

Because of the local nature of the information maintained by each computer connected to the broadcast channel, the computers need not generally be aware of the diameter of the broadcast channel. In one embodiment, each message sent through the broadcast channel has a distance traveled field. Each computer that forwards a message increments the distance traveled field. Each computer also maintains an estimated diameter of the broadcast channel. When a computer receives a message that has traveled a distance that indicates that the estimated diameter is too small, it updates its estimated diameter and broadcasts an estimated diameter message. When a computer receives an estimated diameter message that indicates a diameter that is larger than its own estimated diameter, it updates its own estimated diameter. This estimated diameter is used to establish the distance that an edge connection request message should travel.

External Data Representation

The computers connected to the broadcast channel may internally store their data in different formats. For example, one computer may use 32-bit integers, and another computer may use 64-bit integers. As another example, one computer may use ASCII to represent text and another computer may use Unicode. To allow communications between heterogeneous computers, the messages sent over the broadcast channel may use the XDR ("eXternal Data Representation") format.

The underlying peer-to-peer communications protocol may send multiple messages in a single message stream. The traditional technique for retrieving messages from a stream has been to repeatedly invoke an operating system routine to retrieve the next message in the stream. The retrieval of each message may require two calls to the operating system: one to retrieve the size of the next message and the other to retrieve the number of bytes indicated by the retrieved size. Such calls to the operating system can, however, be very slow in comparison to the invocations of local routines. To overcome the inefficiencies of such repeated calls, the broadcast technique in one embodiment, uses XDR to identify the message boundaries in a stream of messages. The broadcast technique may request the operating system to provide the next, for example, 1,024 bytes from the stream. The broadcast technique can then repeatedly invoke the XDR routines to retrieve the messages and use the success or failure of each invocation to determine whether another block of 1,024 bytes needs to be retrieved from the operating system. The invocation of XDR routines do not involve system calls and are thus more efficient than repeated system calls.

M-Regular.

In the embodiment described above, each fully connected computer has four internal connections. The broadcast technique can be used with other numbers of internal connections. For example, each computer could have 6, 8, or any even number of internal connections. As the number of internal connections increase, the diameter of the broadcast channel tends to decrease, and thus propagation time for a message tends to decrease. The time that it takes to connect a seeking computer to the broadcast channel may, however, increase as the number of internal connections increases. When the number of internal connectors is even, then the broadcast channel can be maintained as m-regular and m-connected (in the steady state). If the number of internal connections is odd, then when the broadcast channel has an odd number of computers connected, one of the computers will have less than that odd number of internal connections.

In such a situation, the broadcast network is neither m-regular nor m-connected. When the next computer connects to the broadcast channel, it can again become m-regular and m-connected. Thus, with an odd number of internal connections, the broadcast channel toggles between being and not being m-regular and m-connected.

Components

FIG. 6 is a block diagram illustrating components of a computer that is connected to a broadcast channel. The above description generally assumed that there was only one broadcast channel and that each computer had only one connection to that broadcast channel. More generally, a network of computers may have multiple broadcast channels, each computer may be connected to more than one broadcast channel, and each computer can have multiple connections to the same broadcast channel. The broadcast channel is well suited for computer processes (e.g., application programs) that execute collaboratively, such as network meeting programs. Each computer process can connect to one or more broadcast channels. The broadcast channels can be identified by channel type (e.g., application program name) and channel instance that represents separate broadcast channels for that channel type. When a process attempts to connect to a broadcast channel, it seeks a process currently connected to that broadcast channel that is executing on a portal computer. The seeking process identifies the broadcast channel by channel type and channel instance.

Computer **600** includes multiple application programs **601** executing as separate processes. Each application program interfaces with a broadcaster component **602** for each broadcast channel to which it is connected. The broadcaster component may be implement as an object that is instantiated within the process space of the application program. Alternatively, the broadcaster component may execute as a separate process or thread from the application program. In one embodiment, the broadcaster component provides functions (e.g., methods of class) that can be invoked by the application programs. The primary functions provided may include a connect function that an application program invokes passing an indication of the broadcast channel to which the application program wants to connect. The application program may provide a callback routine that the broadcaster component invokes to notify the application program that the connection has been completed, that is the process enters the fully connected state. The broadcaster component may also provide an acquire message function that the application program can invoke to retrieve the next message that is broadcast on the broadcast channel. Alternatively, the application program may provide a callback routine (which may be a virtual function provided by the application program) that the broadcaster component invokes to notify the application program that a broadcast message has been received. Each broadcaster component allocates a call-in port using the hashing algorithm. When calls are answered at the call-in port, they are transferred to other ports that serve as the external and internal ports.

The computers connecting to the broadcast channel may include a central processing unit, memory, input devices (e.g., keyboard and pointing device), output devices (e.g., display devices), and storage devices (e.g., disk drives). The memory and storage devices are computer-readable medium that may contain computer instructions that implement the broadcaster component. In addition, the data structures and message structures may be stored or transmitted via a signal transmitted on a computer-readable media, such as a communications link.

FIG. 7 is a block diagram illustrating the sub-components of the broadcaster component in one embodiment. The broadcaster component includes a connect component **701**, an external dispatcher **702**, an internal dispatcher **703** for each internal connection, an acquire message component **704** and a broadcast component **712**. The application program may provide a connect callback component **710** and a receive response component **711** that are invoked by the broadcaster component. The application program invokes the connect component to establish a connection to a designated broadcast channel. The connect component identifies the external port and installs thc external dispatcher for handling messages that are received on the external port. The connect component invokes the seek portal computer component **705** to identify a portal computer that is connected to the broadcast channel and invokes the connect request component **706** to ask the portal computer (if fully connected) to select neighbor processes for the newly connecting process. The external dispatcher receives external messages, identifies the type of message, and invokes the appropriate handling routine **707**. The internal dispatcher receives the internal messages, identifies the type of message, and invokes the appropriate handling routine **708**. The received broadcast messages are stored in the broadcast message queue **709**. The acquire message component is invoked to retrieve messages from the broadcast queue. The broadcast component is invoked by the application program to broadcast messages in the broadcast channel.

The following tables list messages sent by the broadcaster components.

EXTERNAL MESSAGES

Message Type	Description
seeking_connection_call	Indicates that a seeking process would like to know whether the receiving process is fully connected to the broadcast channel
connection_request_call	Indicates that the sending process would like the receiving process to initiate a connection of the sending process to the broadcast channel
edge_proposal_call	Indicates that the sending process is proposing an edge through which the receiving process can connect to the broadcast channel (i.e., edge pinning)
port_connection_call	Indicates that the sending process is proposing a port through which the receiving process can connect to the broadcast channel
connected_stmt	Indicates that the sending process is connected to the broadcast channel
condition_repair_stmt	Indicates that the receiving process should disconnect from one of its neighbors and connect to one of the processes involved in the neighbors with empty port condition

INTERNAL MESSAGES

Message Type	Description
broadcast_stmt	Indicates a message that is being broadcast through the broadcast channel for the application programs
connection_port_search_stmt	Indicates that the designated process is looking for a port through which it can connect to the broadcast channel
connection_edge_search_call	Indicates that the requesting process is looking for an edge through which it can connect to the broadcast channel

-continued

INTERNAL MESSAGES

Message Type	Description
connection_edge_search_resp	Indicates whether the edge between this process and the sending neighbor has been accepted by the requesting party
diameter_estimate_stmt	Indicates an estimated diameter of the broadcast channel
diameter_reset_stmt	Indicates to reset the estimated diameter to indicated diameter
disconnect_stmt	Indicates that the sending neighbor is disconnecting from the broadcast channel
condition_check_stmt	Indicates that neighbors with empty port condition have been detected
condition_double_check_stmt	Indicates that the neighbors with empty ports have the same set of neighbors
shutdown_stmt	Indicates that the broadcast channel is being shutdown

Flow Diagrams

FIGS. **8**–**34** are flow diagrams illustrating the processing of the broadcaster component in one embodiment. FIG. **8** is a flow diagram illustrating the processing of the connect routine in one embodiment. This routine is passed a channel type (e.g., application name) and channel instance (e.g., session identifier), that identifies the broadcast channel to which this process wants to connect. The routine is also passed auxiliary information that includes the list of portal computers and a connection callback routine. When the connection is established, the connection callback routine is invoked to notify the application program. When this process invokes this routine, it is in the seeking connection state. When a portal computer is located that is connected and this routine connects to at least one neighbor, this process enters the partially connected state, and when the process eventually connects to four neighbors, it enters the fully connected state. When in the small regime, a fully connected process may have less than four neighbors. In block **801**, the routine opens the call-in port through which the process is to communicate with other processes when establishing external and internal connections. The port is selected as the first available port using the hashing algorithm described above. In block **802**, the routine sets the connect time to the current time. The connect time is used to identify the instance of the process that is connected through this external port. One process may connect to a broadcast channel of a certain channel type and channel instance using one call-in port and then disconnects, and another process may then connect to that same broadcast channel using the same call-in port. Before the other process becomes fully connected, another process may try to communicate with it thinking it is the fully connected old process. In such a case, the connect time can be used to identify this situation. In block **803**, the routine invokes the seek portal computer routine passing the channel type and channel instance. The seek portal computer routine attempts to locate a portal computer through which this process can connect to the broadcast channel for the passed type and instance. In decision block **804**, if the seek portal computer routine is successful in locating a fully connected process on that portal computer, then the routine continues at block **805**, else the routine returns an unsuccessful indication. In decision block **805**, if no portal computer other than the portal computer on which the process is executing was located, then this is the first process to fully connect to broadcast channel and the routine continues at block **806**, else the routine continues at block **808**. In block **806**, the routine invokes the achieve connection routine to change the state of this process to fully connected. In block **807**, the routine installs the external dispatcher for processing messages received through this process' external port for the passed channel type and channel instance. When a message is received through that external port, the external dispatcher is invoked. The routine then returns. In block **808**, the routine installs an external dispatcher. In block **809**, the routine invokes the connect request routine to initiate the process of identifying neighbors for the seeking computer. The routine then returns.

FIG. **9** is a flow diagram illustrating the processing of the seek portal computer routine in one embodiment. This routine is passed the channel type and channel instance of the broadcast channel to which this process wishes to connect. This routine, for each search depth (e.g., port number), checks the portal computers at that search depth. If a portal computer is located at that search depth with a process that is fully connected to the broadcast channel, then the routine returns an indication of success. In blocks **902**–**911**, the routine loops selecting each search depth until a process is located. In block **902**, the routine selects the next search depth using a port number ordering algorithm. In decision block **903**, if all the search depths have already been selected during this execution of the loop, that is for the currently selected depth, then the routine returns a failure indication, else the routine continues at block **904**. In blocks **904**–**911**, the routine loops selecting each portal computer and determining whether a process of that portal computer is connected to (or attempting to connect to) the broadcast channel with the passed channel type and channel instance. In block **904**, the routine selects the next portal computer. In decision block **905**, if all the portal computers have already been selected, then the routine loops to block **902** to select the next search depth, else the routine continues at block **906**. In block **906**, the routine dials the selected portal computer through the port represented by the search depth. In decision block **907**, if the dialing was successful, then the routine continues at block **908**, else the routine loops to block **904** to select the next portal computer. The dialing will be successful if the dialed port is the call-in port of the broadcast channel of the passed channel type and channel instance of a process executing on that portal computer. In block **908**, the routine invokes a contact process routine, which contacts the answering process of the portal computer through the dialed port and determines whether that process is fully connected to the broadcast channel. In block **909**, the routine hangs up on the selected portal computer. In decision block **910**, if the answering process is fully connected to the broadcast channel, then the routine returns a success indicator, else the routine continues at block **911**. In block **911**, the routine invokes the check for external call routine to determine whether an external call has been made to this process as a portal computer and processes that call. The routine then loops to block **904** to select the next portal computer.

FIG. **10** is a flow diagram illustrating the processing of the contact process routine in one embodiment. This routine determines whether the process of the selected portal computer that answered the call-in to the selected port is fully connected to the broadcast channel. In block **1001**, the routine sends an external message (i.e., seeking_connection_call) to the answering process indicating that a seeking process wants to know whether the answering process is fully connected to the broadcast channel. In block **1002**, the routine receives the external response message from the answering process. In decision block **1003**, if the external response message is successfully received (i.e., seeking_connection_resp), then the routine continues at block **1004**, else the routine returns. Wherever the broadcast component requests to receive an external message, it sets a time out period. If the external message is not received within that time out period, the broadcaster component checks its own call-in port to see if another process is calling it. In particular, the dialed process may be calling the dialing process, which may result in a deadlock situation. The broadcaster component may repeat the receive request several times. If the expected message is not received, then the broadcaster component handles the error as appropriate. In decision block **1004**, if the answering process indicates in its response message that it is fully connected to the broadcast channel, then the routine continues at block **1005**, else the routine continues at block **1006**. In block **1005**, the routine adds the selected portal computer to a list of connected portal computers and then returns. In block **1006**, the routine adds the answering process to a list of fellow seeking processes and then returns.

FIG. **11** is a flow diagram illustrating the processing of the connect request routine in one embodiment. This routine requests a process of a portal computer that was identified as being fully connected to the broadcast channel to initiate the connection of this process to the broadcast channel. In decision block **1101**, if at least one process of a portal computer was located that is fully connected to the broadcast channel, then the routine continues at block **1103**, else the routine continues at block **1102**. A process of the portal computer may no longer be in the list if it recently disconnected from the broadcast channel. In one embodiment, a seeking computer may always search its entire search depth and find multiple portal computers through which it can connect to the broadcast channel. In block **1102**, the routine restarts the process of connecting to the broadcast channel and returns. In block **1103**, the routine dials the process of one of the found portal computers through the call-in port. In decision block **1104**, if the dialing is successful, then the routine continues at block **1105**, else the routine continues at block **1113**. The dialing may be unsuccessful if, for example, the dialed process recently disconnected from the broadcast channel. In block **1105**, the routine sends an external message to the dialed process requesting a connection to the broadcast channel (i.e., connection_request_call). In block **1106**, the routine receives the response message (i.e., connection_request_resp). In decision block **1107**, if the response message is successfully received, then the routine continues at block **1108**, else the routine continues at block **1113**. In block **1108**, the routine sets the expected number of holes (i.e., empty internal connections) for this process based on the received response. When in the large regime, the expected number of holes is zero. When in the small regime, the expected number of holes varies from one to three. In block **1109**, the routine sets the estimated diameter of the broadcast channel based on the received response. In decision block **1111**, if the dialed process is ready to connect to this process as indicated by the response message, then the routine continues at block **1112**, else the routine continues at block **1113**. In block **1112**, the routine invokes the add neighbor routine to add the answering process as a neighbor to this process. This adding of the answering process typically occurs when the broadcast channel is in the small regime. When in the large regime, the random walk search for a neighbor is performed. In block **1113**, the routine hangs up the external connection with the answering process computer and then returns.

FIG. **12** is a flow diagram of the processing of the check for external call routine in one embodiment. This routine is invoked to identify whether a fellow seeking process is attempting to establish a connection to the broadcast channel through this process. In block **1201**, the routine attempts to answer a call on the call-in port. In decision block **1202**, if the answer is successful, then the routine continues at block **1203**, else the routine returns. In block **1203**, the routine receives the external message from the external port. In decision block **1204**, if the type of the message indicates that a seeking process is calling (i.e., seeking_connection_call), then the routine continues at block **1205**, else the routine returns. In block **1205**, the routine sends an external message (i.e., seeking_connection_resp) to the other seeking process indicating that this process is also is seeking a connection. In decision block **1206**, if the sending of the external message is successful, then the routine continues at block **1207**, else the routine returns. In block **1207**, the routine adds the other seeking process to a list of fellow seeking processes and then returns. This list may be used if this process can find no process that is fully connected to the broadcast channel. In which case, this process may check to see if any fellow seeking process were successful in connecting to the broadcast channel. For example, a fellow seeking process may become the first process fully connected to the broadcast channel.

FIG. **13** is a flow diagram of the processing of the achieve connection routine in one embodiment. This routine sets the state of this process to fully connected to the broadcast channel and invokes a callback routine to notify the application program that the process is now fully connected to the requested broadcast channel. In block **1301**, the routine sets the connection state of this process to fully connected. In block **1302**, the routine notifies fellow seeking processes that it is fully connected by sending a connected external message to them (i.e., connected_stmt). In block **1303**, the routine invokes the connect callback routine to notify the application program and then returns.

FIG. **14** is a flow diagram illustrating the processing of the external dispatcher routine in one embodiment. This routine is invoked when the external port receives a message. This routine retrieves the message, identifies the external message type, and invokes the appropriate routine to handle that message. This routine loops processing each message until all the received messages have been handled. In block **1401**, the routine answers (e.g., picks up) the external port and retrieves an external message. In decision block **1402**, if a message was retrieved, then the routine continues at block **1403**, else the routine hangs up on the external port in block **1415** and returns. In decision block **1403**, if the message type is for a process seeking a connection (i.e., seeking_connection_call), then the routine invokes the handle seeking connection call routine in block **1404**, else the routine continues at block **1405**. In decision block **1405**, if the message type is for a connection request call (i.e., connection_request_call), then the routine invokes the handle connection request call routine in block **1406**, else the routine continues at block **1407**. In decision block **1407**, if the message type is edge proposal call (i.e., edge_proposal_call), then the routine invokes the handle edge proposal call routine in block **1408**, else the routine continues at block **1409**. In decision block **1409**, if the message type is port connect call (i.e., port_connect_call), then the routine invokes the handle port connection call routine in block **1410**, else the routine continues at block **1411**. In decision block **1411**, if the message type is a connected statement (i.e., connected_stmt), the routine invokes the handle connected statement in block **1112**, else the routine continues at block **1212**. In decision block **1412**, if the message type is a condition repair statement (i.e., condition_repair_stmt), then the routine invokes the handle condition repair routine in block **1413**, else the routine loops to block **1414** to process the next message. After each handling routine is invoked, the routine loops to block **1414**. In block **1414**, the routine hangs up on the external port and continues at block **1401** to receive the next message.

FIG. **15** is a flow diagram illustrating the processing of the handle seeking connection call routine in one embodiment. This routine is invoked when a seeking process is calling to identify a portal computer through which it can connect to the broadcast channel. In decision block **1501**, if this process is currently fully connected to the broadcast channel identified in the message, then the routine continues at block **1502**, else the routine continues at block **1503**. In block **1502**, the routine sets a message to indicate that this process is fully connected to the broadcast channel and continues at block **1505**. In block **1503**, the routine sets a message to indicate that this process is not fully connected. In block **1504**, the routine adds the identification of the seeking process to a list of fellow seeking processes. If this process is not fully connected, then it is attempting to connect to the broadcast channel. In block **1505**, the routine sends the external message response (i.e., seeking_connection_resp) to the seeking process and then returns.

FIG. **16** is a flow diagram illustrating processing of the handle connection request call routine in one embodiment. This routine is invoked when the calling process wants this process to initiate the connection of the process to the broadcast channel. This routine either allows the calling process to establish an internal connection with this process (e.g., if in the small regime) or starts the process of identifying a process to which the calling process can connect. In decision block **1601**, if this process is currently fully connected to the broadcast channel, then the routine continues at block **1603**, else the routine hangs up on the external port in block **1602** and returns. In block **1603**, the routine sets the number of holes that the calling process should expect in the response message. In block **1604**, the routine sets the estimated diameter in the response message. In block **1605**, the routine indicates whether this process is ready to connect to the calling process. This process is ready to connect when the number of its holes is greater than zero and the calling process is not a neighbor of this process. In block **1606**, the routine sends to the calling process an external message that is responsive to the connection request call (i.e., connection_request_resp). In block **1607**, the routine notes the number of holes that the calling process needs to fill as indicated in the request message. In decision block **1608**, if this process is ready to connect to the calling process, then the routine continues at block **1609**, else the routine continues at block **1611**. In block **1609**, the routine invokes the add neighbor routine to add the calling process as a neighbor. In block **1610**, the routine decrements the number of holes that the calling process needs to fill and continues at block **1611**. In block **1611**, the routine hangs up on the external port. In decision block **1612**, if this process has no holes or the estimated diameter is greater than one (i.e., in the large regime), then the routine continues at block **1613**, else the routine continues at block **1616**. In blocks **1613–1615**, the routine loops forwarding a request for an edge through which to connect to the calling process to the broadcast channel. One request is forwarded for each pair of holes of the calling process that needs to be filled. In decision block **1613**, if the number of holes of the calling process to be filled is greater than or equal to two, then the routine continues at block **1614**, else the routine continues at block **1616**. In block **1614**, the routine invokes the forward connection edge search routine. The invoked routine is passed to an indication of the calling process and the random walk distance. In one embodiment, the distance is twice in the estimated diameter of the broadcast channel. In block **1614**, the routine decrements the holes left to fill by two and loops to block **1613**. In decision block **1616**, if there is still a hole to fill, then the routine continues at block **1617**, else the routine returns. In block **1617**, the routine invokes the fill hole routine passing the identification of the calling process. The fill hole routine broadcasts a connection port search statement (i.e., connection_port_search_stmt) for a hole of a connected process through which the calling process can connect to the broadcast channel. The routine then returns.

FIG. **17** is a flow diagram illustrating the processing of the add neighbor routine in one embodiment. This routine adds the process calling on the external port as a neighbor to this process. In block **1701**, the routine identifies the calling process on the external port. In block **1702**, the routine sets a flag to indicate that the neighbor has not yet received the broadcast messages from this process. This flag is used to ensure that there are no gaps in the messages initially sent to the new neighbor. The external port becomes the internal port for this connection. In decision block **1703**, if this process is in the seeking connection state, then this process is connecting to its first neighbor and the routine continues at block **1704**, else the routine continues at block **1705**. In block **1704**, the routine sets the connection state of this process to partially connected. In block **1705**, the routine adds the calling process to the list of neighbors of this process. In block **1706**, the routine installs an internal dispatcher for the new neighbor. The internal dispatcher is invoked when a message is received from that new neighbor through the internal port of that new neighbor. In decision block **1707**, if this process buffered up messages while not fully connected, then the routine continues at block **1708**, else the routine continues at block **1709**. In one embodiment, a process that is partially connected may buffer the messages that it receives is through an internal connection so that it can send these messages as it connects to new neighbors. In block **1708**, the routine sends the buffered messages to the new neighbor through the internal port. In decision block **1709**, if the number of holes of this process equals the expected number of holes, then this process is fully connected and the routine continues at block **1710**, else the routine continues at block **1711**. In block **1710**, the routine invokes the achieve connected routine to indicate that this process is fully connected. In decision block **1711**, if the number of holes for this process is zero, then the routine continues at block **1712**, else the routine returns. In block **1712**, the routine deletes any pending edges and then returns. A pending edge is an edge that has been proposed to this process for edge pinning, which in this case is no longer needed.

FIG. **18** is a flow diagram illustrating the processing of the forward connection edge search routine in one embodiment. This routine is responsible for passing along a request to connect a requesting process to a randomly selected neighbor of this process through the internal port of the selected neighbor, that is part of the random walk. In decision block **1801**, if the forwarding distance remaining is greater than zero, then the routine continues at block **1804**, else the routine continues at block **1802**. In decision block **1802**, if the number of neighbors of this process is greater than one, then the routine continues at block **1804**, else this broadcast channel is in the small regime and the routine continues at block **1803**. In decision block **1803**, if the requesting process is a neighbor of this process, then the routine returns, else the routine continues at block **1804**. In blocks **1804–1807**, the routine loops attempting to send a connection edge search call internal message (i.e., connection_edge_search_call) to a randomly selected neighbor. In block **1804**, the routine randomly selects a neighbor of this process. In decision block **1805**, if all the neighbors of this process have already been selected, then the routine cannot forward the message and the routine returns, else the routine continues at block **1806**. In block **1806**, the routine sends a connection edge search call internal message to the selected neighbor. In decision block **1807**, if the sending of the message is successful, then the routine continues at block **1808**, else the routine loops to block **1804** to select the next neighbor. When the sending of an internal message is unsuccessful, then the neighbor may have disconnected from the broadcast channel in an unplanned manner. Whenever such a situation is detected by the broadcaster component, it attempts to find another neighbor by invoking the fill holes routine to fill a single hole or the forward connecting edge search routine to fill two holes. In is block **1808**, the routine notes that the recently sent connection edge search call has not yet been acknowledged and indicates that the edge to this neighbor is reserved if the remaining forwarding distance is less than or equal to one. It is reserved because the selected neighbor may offer this edge to the requesting process for edge pinning. The routine then returns.

FIG. **19** is a flow diagram illustrating the processing of the handle edge proposal call routine. This routine is invoked when a message is received from a proposing process that proposes to connect an edge between the proposing process and one of its neighbors to this process for edge pinning. In decision block **1901**, if the number of holes of this process minus the number of pending edges is greater than or equal to one, then this process still has holes to be filled and the routine continues at block **1902**, else the routine continues at block **1911**. In decision block **1902**, if the proposing process or its neighbor is a neighbor of this process, then the routine continues at block **1911**, else the routine continues at block **1903**. In block **1903**, the routine indicates that the edge is pending between this process and the proposing process. In decision block **1904**, if a proposed neighbor is already pending as a proposed neighbor, then the routine continues at block **1911**, else the routine continues at block **1907**. In block **1907**, the routine sends an edge proposal response as an external message to the proposing process (i.e., edge_proposal_resp) indicating that the proposed edge is accepted. In decision block **1908**, if the sending of the message was successful, then the routine continues at block **1909**, else the routine returns. In block **1909**, the routine adds the edge as a pending edge. In block **1910**, the routine invokes the add neighbor routine to add the proposing process on the external port as a neighbor. The routine then returns. In block **1911**, the routine sends an external message (i.e., edge_proposal_resp) indicating that this proposed edge is not accepted. In decision block **1912**, if the number of holes is odd, then the routine continues at block **1913**, else the routine returns. In block **1913**, the routine invokes the fill hole routine and then returns.

FIG. **20** is a flow diagram illustrating the processing of the handle port connection call routine in one embodiment. This routine is invoked when an external message is received then indicates that the sending process wants to connect to one hole of this process. In decision block **2001**, if the number of holes of this process is greater than zero, then the routine continues at block **2002**, else the routine continues at block **2003**. In decision block **2002**, if the sending process is not a neighbor, then the routine continues at block **2004**, else the routine continues to block **2003**. In block **2003**, the routine sends a port connection response external message (i.e., port_connection_resp) to the sending process that indicates that it is not okay to connect to this process. The routine then returns. In block **2004**, the routine sends a port connection response external message to the sending process that indicates that is okay to connect this process. In decision block **2005**, if the sending of the message was successful, then the routine continues at block **2006**, else the routine continues at block **2007**. In block **2006**, the routine invokes the add neighbor routine to add the sending process as a neighbor of this process and then returns. In block **2007**, the routine hangs up the external connection. In block **2008**, the routine invokes the connect request routine to request that a process connect to one of the holes of this process. The routine then returns.

FIG. **21** is a flow diagram illustrating the processing of the fill hole routine in one embodiment. This routine is passed an indication of the requesting process. If this process is requesting to fill a hole, then this routine sends an internal message to other processes. If another process is requesting to fill a hole, then this routine invokes the routine to handle a connection port search request. In block **2101**, the routine initializes a connection port search statement internal, message (i.e., connection_port_search_stmt). In decision block **2102**, if this process is the requesting process, then the routine continues at block **2103**, else the routine continues at block **2104**. In block **2103**, the routine distributes the message to the neighbors of this process through the internal ports and then returns. In block **2104**, the routine invokes the handle connection port search routine and then returns.

FIG. **22** is a flow diagram illustrating the processing of the internal dispatcher routine in one embodiment. This routine is passed an indication of the neighbor who sent the internal message. In block **2201**, the routine receives the internal message. This routine identifies the message type and invokes the appropriate routine to handle the message. In block **2202**, the routine assesses whether to change the estimated diameter of the broadcast channel based on the information in the received message. In decision block **2203**, if this process is the originating process of the message or the message has already been received (i.e., a duplicate), then the routine ignores the message and continues at block **2208**, else the routine continues at block **2203A**. In decision block **2203A**, if the process is partially connected, then the routine continues at block **2203B**, else the routine continues at block **2204**. In block **2203B**, the routine adds the message to the pending connection buffer and continues at block **2204**. In decision blocks **2204–2207**, the routine decodes the message type and invokes the appropriate routine to handle the message. For example, in decision block **2204**, if the type of the message is broadcast statement (i.e., broadcast_stmt), then the routine invokes the handle broadcast message routine in block **2205**. After invoking the appropriate handling routine, the routine continues at block **2208**. In decision block **2208**, if the partially connected buffer is full, then the routine continues at block **2209**, else the routine continues at block **2210**. The broadcaster component collects all its internal messages in a buffer while partially connected so that it can forward the messages as it connects to new neighbors. If, however, that buffer becomes full, then the process assumes that it is now fully connected and that the expected number of connections was too high, because the broadcast channel is now in the small regime. In block **2209**, the routine invokes the achieve connection routine and then continues in block **2210**. In decision block **2210**, if the application program message queue is empty, then the routine returns, else the routine continues at block **2212**. In block **2212**, the routine invokes the receive response routine passing the acquired message and then returns. The received response routine is a callback routine of the application program.

FIG. **23** is a flow diagram illustrating the processing of the handle broadcast message routine in one embodiment. This routine is passed an indication of the originating process, an indication of the neighbor who sent the broadcast message, and the broadcast message itself. In block **2301**, the routine performs the out of order processing for this message. The broadcaster component queues messages from each originating process until it can send them in sequence number order to the application program. In block **2302**, the routine invokes the distribute broadcast message routine to forward the message to the neighbors of this process. In decision block **2303**, if a newly connected neighbor is waiting to receive messages, then the routine continues at block **2304**, else the routine returns. In block **2304**, the routine sends the messages in the correct order if possible for each originating process and then returns.

FIG. **24** is a flow diagram illustrating the processing of the distribute broadcast message routine in one embodiment. This routine sends the broadcast message to each of the neighbors of this process, except for the neighbor who sent the message to this process. In block **2401**, the routine selects the next neighbor other than the neighbor who sent the message. In decision block **2402**, if all such neighbors have already been selected, then the routine returns. In block **2403**, the routine sends the message to the selected neighbor and then loops to block **2401** to select the next neighbor.

FIG. **26** is a flow diagram illustrating the processing of the handle connection port search statement routine in one embodiment. This routine is passed an indication of the neighbor that sent the message and the message itself. In block **2601**, the routine invokes the distribute internal message which sends the message to each of its neighbors other than the sending neighbor. In decision block **2602**, if the number of holes of this process is greater than zero, then the routine continues at block **2603**, else the routine returns. In decision block **2603**, if the requesting process is a neighbor, then the routine continues at block **2605** else the routine continues at block **2604**, In block **2604**, the routine invokes the court neighbor routine and then returns. The court neighbor routine connects this process to the requesting process if possible. In block **2605**, if this process has one hole, then the neighbors with empty ports condition exists and the routine continues at block **2606**, else the routine returns. In block **2606**, the routine generates a condition check message (i.e., condition_check) that includes a list of this process' neighbors. In block **2607**, the routine sends the message to the requesting neighbor.

FIG. **27** is a flow diagram illustrating the processing of the court neighbor routine in one embodiment. This routine is passed an indication of the prospective neighbor for this process. If this process can connect to the prospective neighbor, then it sends a port connection call external message to the prospective neighbor and adds the prospective neighbor as a neighbor. In decision block **2701**, if the prospective neighbor is already a neighbor, then the routine returns, else the routine continues at block **2702**. In block **2702**, the routine dials the prospective neighbor. In decision block **2703**, if the number of holes of this process is greater than zero, then the routine continues at block **2704**, else the routine continues at block **2706**. In block **2704**, the routine sends a port connection call external message (i.e., port_connection_call) to the prospective neighbor and receives its response (i.e., port_connection_resp). Assuming the response is successfully received, in block **2705**, the routine adds the prospective neighbor as a neighbor of this process by invoking the add neighbor routine. In block **2706**, the routine hangs up with the prospect and then returns.

FIG. **28** is a flow diagram illustrating the processing of the handle connection edge search call routine in one embodiment. This routine is passed a indication of the neighbor who sent the message and the message itself. This routine either forwards the message to a neighbor or proposes the edge between this process and the sending neighbor to the requesting process for edge pinning. In decision block **2801**, if this process is not the requesting process or the number of holes of the requesting process is still greater than or equal to two, then the routine continues at block **2802**, else the routine continues at block **2813**. In decision block **2802**, if the forwarding distance is greater than zero, then the random walk is not complete and the routine continues at block **2803**, else the routine continues at block **2804**. In block **2803**, the routine invokes the forward connection edge search routine passing the identification of the requesting process and the decremented forwarding distance. The routine then continues at block **2815**. In decision block **2804**, if the requesting process is a neighbor or the edge between this process and the sending neighbor is reserved because it has already been offered to a process, then the routine continues at block **2805**, else the routine continues at block **2806**. In block **2805**, the routine invokes the forward connection edge search routine passing an indication of the requesting party and a toggle indicator that alternatively indicates to continue the random walk for one or two more computers. The routine then continues at block **2815**. In block **2806**, the routine dials the requesting process via the call-in port. In block **2807**, the routine sends an edge proposal call external message (i.e., edge_proposal_call) and receives the response (i.e., edge_proposal_resp). Assuming that the response is successfully received, the routine continues at block **2808**. In decision block **2808**, if the response indicates that the edge is acceptable to the requesting process, then the routine continues at block **2809**, else the routine continues at block **2812**. In block **2809**, the routine reserves the edge between this process and the sending neighbor. In block **2810**, the routine adds the requesting process as a neighbor by invoking the add neighbor routine. In block **2811**, the routine removes the sending neighbor as a neighbor. In block **2812**, the routine hangs up the external port and continues at block **2815**. In decision block **2813**, if this process is the requesting process and the number of holes of this process equals one, then the routine continues at block **2814**, else the routine continues at block **2815**. In block **2814**, the routine invokes the fill hole routine. In block **2815**, the routine sends an connection edge search response message (i.e., connection_edge_search_response) to the sending neighbor indicating acknowledgement and then returns. The graphs are sensitive to parity. That is, all possible paths starting from a node and ending at that node will have an even length unless the graph has a cycle whose length is odd. The broadcaster component uses a toggle indicator to vary the random walk distance between even and odd distances.

FIG. **29** is a flow diagram illustrating the processing of the handle connection edge search response routine in one embodiment. This routine is passed as indication of the requesting process, the sending neighbor, and the message. In block **2901**, the routine notes that the connection edge search response (i.e., connection_edge_search_resp) has been received and if the forwarding distance is less than or equal to one unreserves the edge between this process and the sending neighbor. In decision block **2902**, if the requesting process indicates that the edge is acceptable as indicated in the message, then the routine continues at block **2903**, else the routine returns. In block **2903**, the routine reserves the edge between this process and the sending neighbor. In block **2904**, the routine removes the sending neighbor as a neighbor. In block **2905**, the routine invokes the court neighbor routine to connect to the requesting process. In decision block **2906**, if the invoked routine was unsuccessful, then the routine continues at block **2907**, else the routine returns. In decision block **2907**, if the number of holes of this process is greater than zero, then the routine continues at block **2908**, else the routine returns. In block **2908**, the routine invokes the fill hole routine and then returns.

FIG. **30** is a flow diagram illustrating the processing of the broadcast routine in one embodiment. This routine is invoked by the application program to broadcast a message on the broadcast channel. This routine is passed the message to be broadcast. In decision block **3001**, if this process has at least one neighbor, then the routine continues at block **3002**, else the routine returns since it is the only process connected to be broadcast channel. In block **3002**, the routine generates an internal message of the broadcast statement type (i.e., broadcast_stmt). In block **3003**, the routine sets the sequence number of the message. In block **3004**, the routine invokes the distribute internal message routine to broadcast the message on the broadcast channel. The routine returns.

FIG. **31** is a flow diagram illustrating the processing of the acquire message routine in one embodiment. The acquire message routine may be invoked by the application program or by a callback routine provided by the application program. This routine returns a message. In block **3101**, the routine pops the message from the message queue of the broadcast channel. In decision block **3102**, if a message was retrieved, then the routine returns an indication of success, else the routine returns indication of failure.

FIGS. **32**–**34** are flow diagrams illustrating the processing of messages associated with the neighbors with empty ports condition. FIG. **32** is a flow diagram illustrating processing of the handle condition check message in one embodiment. This message is sent by a neighbor process that has one hole and has received a request to connect to a hole of this process. In decision block **3201**, if the number of holes of this process is equal to one, then the routine continues at block **3202**, else the neighbors with empty ports condition does not exist any more and the routine returns. In decision block **3202**, if the sending neighbor and this process have the same set of neighbors, the routine continues at block **3203**, else the routine continues at block **3205**. In block **3203**, the routine initializes a condition double check message (i.e., condition_double_check) with the list of neighbors of this process. In block **3204**, the routine sends the message internally to a neighbor other than sending neighbor. The routine then returns. In block **3205**, the routine selects a neighbor of the sending process that is not also a neighbor of this process. In block **3206**, the routine sends a condition repair message (i.e., condition_repair_stmt) externally to the selected process. In block **3207**, the routine invokes the add neighbor routine to add the selected neighbor as a neighbor of this process and then returns.

FIG. **33** is a flow diagram illustrating processing of the handle condition repair statement routine in one embodiment. This routine removes an existing neighbor and connects to the process that sent the message. In decision block **3301**, if this process has no holcs, then the routine continues at block **3302**, else the routine continues at block **3304**. In block **3302**, the routine selects a neighbor that is not involved in the neighbors with empty ports condition. In block **3303**, the routine removes the selected neighbor as a neighbor of this process. Thus, this process that is executing the routine now has at least one hole. In block **3304**, the routine invokes the add neighbor routine to add the process that sent the message as a neighbor of this process. The routine then returns.

FIG. **34** is a flow diagram illustrating the processing of the handle condition double check routine. This routine determines whether the neighbors with empty ports condition really is a problem or whether the broadcast channel is in the small regime. In decision block **3401**, if this process has one hole, then the routine continues at block **3402**, else the routine continues at block **3403**. If this process does not have one hole, then the set of neighbors of this process is not the same as the set of neighbors of the sending process. In decision block **3402**, if this process and the sending process have the same set of neighbors, then the broadcast channel is not in the small regime and the routine continues at block **3403**, else the routine continues at block **3406**. In decision block **3403**, if this process has no holes, then the routine returns, else the routine continues at block **3404**. In block **3404**, the routine sets the estimated diameter for this process to one. In block **3405**, the routine broadcasts a diameter reset internal message (i.e., diameter_reset) indicating that the estimated diameter is one and then returns. In block **3406**, the routine creates a list of neighbors of this process. In block **3407**, the routine sends the condition check message (i.e., condition_check_stmt) with the list of neighbors to the neighbor who sent the condition double check message and then returns.

From the above description, it will be appreciated that although specific embodiments of the technology have been described, various modifications may be made without deviating from the spirit and scope of the invention. For example, the communications on the broadcast channel may be encrypted. Also, the channel instance or session identifier may be a very large number (e.g., 128 bits) to help prevent an unauthorized user to maliciously tap into a broadcast channel. The portal computer may also enforce security and not allow an unauthorized user to connect to the broadcast channel.

Accordingly, the invention is not limited except by the claims.

We claim:

1. A method of disconnecting a first computer from a second computer, the first computer and the second computer being connected to a broadcast channel, said broadcast channel forming an m-regular graph where m is at least 3, the method comprising:

when the first computer decides to disconnect from the second computer, the first computer sends a disconnect message to the second computer, said disconnect message including a list of neighbors of the first computer; and

when the second computer receives the disconnect message from the first computer, the second computer broadcasts a connection port search message on the broadcast channel to find a third computer to which it can connect in order to maintain an m-regular graph, said third computer being one of the neighbors on said list of neighbors.

2. The method of claim 1 wherein the second computer receives a port connection message indicating that the third computer is proposing that the third computer and the second computer connect.

3. The method of claim 1 wherein the first computer disconnects from the second computer after sending the disconnect message.

4. The method of claim 1 wherein the broadcast channel is implemented using the Internet.

5. The method of claim 1 wherein the first computer and second computer are connected via a TCP/IP connection.

6. A method for healing a disconnection of a first computer from a second computer, the computers being connected to a broadcast channel, said broadcast channel being an m-regular graph where m is at least 3, the method comprising:

attempting to send a message from the first computer to the second computer; and

when the attempt to send the message is unsuccessful, broadcasting from the first computer a connection port search message indicating that the first computer needs a connection; and

having a third computer not already connected to said first computer respond to said connection port search message in a manner as to maintain an m-regular graph.

7. The method of claim 6 including:

when a third computer receives the connection port search message and the third computer also needs a connection, sending a message from the third computer to the first computer proposing that the first computer and third computer connect.

8. The method of claim 7 including:

when the first computer receives the message proposing that the first computer and third computer connect, sending from the first computer to the third computer a message indicating that the first computer accepts the proposal to connect the first computer to the third computer.

9. The method of claim 6 wherein each computer connected to the broadcast channel is connected to at least three other computers.

10. The method of claim 6 wherein the broadcasting includes sending the message to each computer to which the first computer is connected.

11. A computer-readable medium containing instructions for controlling disconnecting of a computer from another computer, the computer and the other computer being connected to a broadcast channel, said broadcast channel being an m-regular graph where m is at least 3, comprising:

a component that, when the computer decides to disconnect from the other computer, the computer sends a disconnect message to the other computer, said disconnect message including a list of neighbors of the computer; and

a component that, when the computer receives a disconnect message from another computer, the computer broadcasts a connection port search message on the broadcast channel to find a computer to which it can connect in order to maintain an m-regular graph, said computer to which it can connect being one of the neighbors on said list of neighbors.

12. The computer-readable medium of claim 11 including:

a component that, when the computer receives a connection port search message and the computer needs to connect to another computer, sends to the computer that sent the connection port search message a port connection message indicating that the computer is proposing that the computer that sent the connection port search message connect to the computer.

13. The computer-readable medium of claim 12 including:

a component that, when the computer receives a port connection message, connecting to the computer that sent the port connection message.

14. The computer-readable medium of claim 11 wherein the computers are connected via a TCP/IP connection.

15. The computer-readable medium of claim 11 wherein the computers that are connected to the broadcast channel are peers.

16. The computer-readable medium of claim 11 wherein the broadcast channel is implemented using the Internet.

* * * * *

PATENT COOPERATION TREATY

From the INTERNATIONAL SEARCHING AUTHORITY

To:
PERKINS COIE LLP
Attn. Pirio, Maurice J.
P.O. Box 1247
Seattle, WA 98111-1247
UNITED STATES OF AMERICA

PCT

NOTIFICATION OF TRANSMITTAL OF THE INTERNATIONAL SEARCH REPORT OR THE DECLARATION

(PCT Rule 44.1)

Date of mailing (*day/month/year*) 05/06/2002

Applicant's or agent's file reference
030048001WO

FOR FURTHER ACTION See paragraphs 1 and 4 below

International application No.
PCT/US 01/24240

International filing date (*day/month/year*) 31/07/2001

Applicant
THE BOEING COMPANY

1. ☒ The applicant is hereby notified that the International Search Report has been established and is transmitted herewith.

Filing of amendments and statement under Article 19:
The applicant is entitled, if he so wishes, to amend the claims of the International Application (see Rule 46):

When? The time limit for filing such amendments is normally 2 months from the date of transmittal of the International Search Report; however, for more details, see the notes on the accompanying sheet.

Where? Directly to the International Bureau of WIPO
34, chemin des Colombettes
1211 Geneva 20, Switzerland
Fascimile No.: (41-22) 740.14.35

For more detailed instructions, see the notes on the accompanying sheet.

2. □ The applicant is hereby notified that no International Search Report will be established and that the declaration under Article 17(2)(a) to that effect is transmitted herewith.

3. □ **With regard to the protest** against payment of (an) additional fee(s) under Rule 40.2, the applicant is notified that:

□ the protest together with the decision thereon has been transmitted to the International Bureau together with the applicant's request to forward the texts of both the protest and the decision thereon to the designated Offices.

□ no decision has been made yet on the protest; the applicant will be notified as soon as a decision is made.

4. **Further action(s):** The applicant is reminded of the following:

Shortly after **18 months** from the priority date, the international application will be published by the International Bureau. If the applicant wishes to avoid or postpone publication, a notice of withdrawal of the international application, or of the priority claim, must reach the International Bureau as provided in Rules 90*bis*.1 and 90*bis*.3, respectively, before the completion of the technical preparations for international publication.

Within **19 months** from the priority date, a demand for international preliminary examination must be filed if the applicant wishes to postpone the entry into the national phase until 30 months from the priority date (in some Offices even later).

Within **20 months** from the priority date, the applicant must perform the prescribed acts for entry into the national phase before all designated Offices which have not been elected in the demand or in a later election within 19 months from the priority date or could not be elected because they are not bound by Chapter II.

Name and mailing address of the International Searching Authority
European Patent Office, P.B. 5818 Patentlaan 2
NL-2280 HV Rijswijk
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,
Fax: (+31-70) 340-3016

Authorized officer
Claude Berthon

NOTES TO FORM PCT/ISA/220

These Notes are intended to give the basic instructions concerning the filing of amendments under article 19. The Notes are based on the requirements of the Patent Cooperation Treaty, the Regulations and the Administrative Instructions under that Treaty. In case of discrepancy between these Notes and those requirements, the latter are applicable. For more detailed information, see also the PCT Applicant's Guide, a publication of WIPO.

In these Notes, "Article", "Rule", and "Section" refer to theprovisions of the PCT, the PCT Regulations and the PCT Administrative Instructions respectively.

INSTRUCTIONS CONCERNING AMENDMENTS UNDER ARTICLE 19

The applicant has, after having received the international search report, one opportunity to amend the claims of the international application. It should however be emphasized that, since all parts of the international application (claims, description and drawings) may be amended during the international preliminary examination procedure, there is usually no need to file amendments of the claims under Article 19 except where, e.g. the applicant wants the latter to be published for the purposes of provisional protection or has another reason for amending the claims before international pbulication. Furthermore, it should be emphasized that provisional protection is available in some States only.

What parts of the international application may be amended?

Under Article 19, only the claims may be amended.

During the international phase, the claims may also be amended (or further amended) under Article 34 before the International Preliminary Examining Authority. The description and drawings may only be amended under Article 34 before the International Examining Authority.

Upon entry into the national phase, all parts of the international application may be amended under Article 28 or, where applicable, Article 41.

When? Within 2 months from the date of transmittal of the international search report or 16 months from the priority date, whichever time limit expires later. It should be noted, however, that the amendments will be considered as having been received on time if they are received by the International Bureau after the expiration of the applicable time limit but before the completion of the technical preparations for international publication (Rule 46.1).

Where not to file the amendments?

The amendments may only be filed with the International Bureau and not with the receiving Office or the International Searching Authority (Rule 46.2).

Where a demand for international preliminary examination has been/is filed, see below.

How? Either by cancelling one or more entire claims, by adding one or more new claims or by amending the text of one or more of the claims as filed.

A replacement sheet must be submitted for each sheet of the claims which, on account of an amendment or amendments, differs from the sheet originally filed.

All the claims appearing on a replacement sheet must be numbered in Arabic numerals. Where a claim is cancelled, no renumbering of the other claims is required. In all cases where claims are renumbered, they must be renumbered consecutively (Administrative Instructions, Section 205(b)).

The amendments must be made in the language in which the international application is to be published.

What documents must/may accompany the amendments?

Letter (Section 205(b)):

The amendments must be submitted with a letter.

The letter will not be published with the international application and the amended claims. It should not be confused with the "Statement under Article 19(1)" (see below, under "Statement under Article 19(1)").

The letter must be in English or French, at the choice of the applicant. However, if the language of the international application is English, the letter must be in English; if the language of the international application is French, the letter must be in French.

NOTES TO FORM PCT/ISA/220 (continued)

The letter must indicate the differences between the claims as filed and the claims as amended. It must, in particular, indicate, in connection with each claim appearing in the international application (it being understood that identical indications concerning several claims may be grouped),whether

(i) the claim is unchanged;

(ii) the claim is cancelled;

(iii) the claim is new;

(iv) the claim replaces one or more claims as filed;

(v) the claim is the result of the division of a claim as filed.

The following examples illustrate the manner in which amendments must be explained in the accompanying letter:

1. [Where originally there were 48 claims and after amendment of some claims there are 51]: "Claims 1 to 29, 31, 32, 34, 35, 37 to 48 replaced by amended claims bearing the same numbers; claims 30, 33 and 36 unchanged; new claims 49 to 51 added."

2. [Where originally there were 15 claims and after amendment of all claims there are 11]: "Claims 1 to 15 replaced by amended claims 1 to 11."

3. [Where originally there were 14 claims and the amendments consist in cancelling some claims and in adding new claims]: "Claims 1 to 6 and 14 unchanged; claims 7 to 13 cancelled; new claims 15, 16 and 17 added." or "Claims 7 to 13 cancelled; new claims 15, 16 and 17 added; all other claims unchanged."

4. [Where various kinds of amendments are made]: "Claims 1-10 unchanged; claims 11 to 13, 18 and 19 cancelled; claims 14, 15 and 16 replaced by amended claim 14; claim 17 subdivided into amended claims 15, 16 and 17; new claims 20 and 21 added."

"Statement under article 19(1)" (Rule 46.4)

The amendments may be accompanied by a statement explaining the amendments and indicating any impact that such amendments might have on the description and the drawings (which cannot be amended under Article 19(1)).

The statement will be published with the international application and the amended claims.

It must be in the language in which the international appplication is to be published.

It must be brief, not exceeding 500 words if in English or if translated into English.

It should not be confused with and does not replace the letter indicating the differences between the claims as filed and as amended. It must be filed on a separate sheet and must be identified as such by a heading, preferably by using the words "Statement under Article 19(1)."

It may not contain any disparaging comments on the international search report or the relevance of citations contained in that report. Reference to citations, relevant to a given claim, contained in the international search report may be made only in connection with an amendment of that claim.

Consequence if a demand for international preliminary examination has already been filed

If, at the time of filing any amendments under Article 19, a demand for international preliminary examination has already been submitted, the applicant must preferably, at the same time of filing the amendments with the International Bureau, also file a copy of such amendments with the International Preliminary Examining Authority (see Rule 62.2(a), first sentence).

Consequence with regard to translation of the international application for entry into the national phase

The applicant's attention is drawn to the fact that, where upon entry into the national phase, a translation of the claims as amended under Article 19 may have to be furnished to the designated/elected Offices, instead of, or in addition to, the translation of the claims as filed.

For further details on the requirements of each designated/elected Office, see Volume II of the PCT Applicant's Guide.

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 01/24240

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 4912656 A	27-03-1990	NONE	
US 5056085 A	08-10-1991	NONE	

Form PCT/ISA/210 (patent family annex) (July 1992)

PATENT COOPERATION TREATY

PCT

INTERNATIONAL SEARCH REPORT

(PCT Article 18 and Rules 43 and 44)

Applicant's or agent's file reference 030048001WO	**FOR FURTHER ACTION** see Notification of Transmittal of International Search Report (Form PCT/ISA/220) as well as, where applicable, item 5 below.	
International application No. PCT/US 01/ 24240	International filing date *(day/month/year)* 31/07/2001	(Earliest) Priority Date *(day/month/year)* 31/07/2000
Applicant THE BOEING COMPANY		

This International Search Report has been prepared by this International Searching Authority and is transmitted to the applicant according to Article 18. A copy is being transmitted to the International Bureau.

This International Search Report consists of a total of 3 sheets.

☒ It is also accompanied by a copy of each prior art document cited in this report.

1. **Basis of the report**

a. With regard to the **language,** the international search was carried out on the basis of the international application in the language in which it was filed, unless otherwise indicated under this item.

☐ the international search was carried out on the basis of a translation of the international application furnished to this Authority (Rule 23.1(b)).

b. With regard to any **nucleotide and/or amino acid sequence** disclosed in the international application, the international search was carried out on the basis of the sequence listing :

☐ contained in the international application in written form.

☐ filed together with the international application in computer readable form.

☐ furnished subsequently to this Authority in written form.

☐ furnished subsequently to this Authority in computer readble form.

☐ the statement that the subsequently furnished written sequence listing does not go beyond the disclosure in the international application as filed has been furnished.

☐ the statement that the information recorded in computer readable form is identical to the written sequence listing has been furnished

2. ☐ **Certain claims were found unsearchable** (See Box I).

3. ☐ **Unity of invention is lacking** (see Box II).

4. With regard to the **title,**

☒ the text is approved as submitted by the applicant.

☐ the text has been established by this Authority to read as follows:

5. With regard to the **abstract,**

☒ the text is approved as submitted by the applicant.

☐ the text has been established, according to Rule 38.2(b), by this Authority as it appears in Box III. The applicant may, within one month from the date of mailing of this international search report, submit comments to this Authority.

6. The figure of the **drawings** to be published with the abstract is Figure No. 1

☐ as suggested by the applicant.

☐ None of the figures.

☐ because the applicant failed to suggest a figure.

☒ because this figure better characterizes the invention.

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 01/24240

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 H04L12/18 H04L12/56

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal, WPI Data, PAJ, IBM-TDB, INSPEC, COMPENDEX

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 4 912 656 A (CAIN JOSEPH B ET AL) 27 March 1990 (1990-03-27)	1-3, 7-10,12, 19,20, 22,24-27
	column 5, line 60 -column 7, line 20	
A		13-18, 21,28-33

Y	US 5 056 085 A (VU THU V) 8 October 1991 (1991-10-08)	1-3, 7-10,12, 19,20, 22,24-27
	column 2, line 49 -column 3, line 12 --- -/--	

☒ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search 27 May 2002	Date of mailing of the international search report 05/06/2002
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Authorized officer Ströbeck, A.

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 01/24240

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	ALAGAR S ET AL: "Reliable broadcast in mobile wireless networks" MILITARY COMMUNICATIONS CONFERENCE, 1995. MILCOM '95, CONFERENCE RECORD, IEEE SAN DIEGO, CA, USA 5-8 NOV. 1995, NEW YORK, NY, USA,IEEE, US, 5 November 1995 (1995-11-05), pages 236-240, XP010153965 ISBN: 0-7803-2489-7 page 237, left-hand column, line 43 -page 239, right-hand column, last line -----	1-33

UNITED STATES PATENT AND TRADEMARK OFFICE

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

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/629,577	07/31/2000	Fred B. Holt	030048003US	4317

25096 7590 11/05/2003

PERKINS COIE LLP
PATENT-SEA
P.O. BOX 1247
SEATTLE, WA 98111-1247

EXAMINER	
LAZARO, DAVID R	
ART UNIT	PAPER NUMBER
2155	6

DATE MAILED: 11/05/2003

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

PTO-90C (Rev. 10/03)

11

4

Office Action Summary	**Application No.** 09/629,577	**Applicant(s)** HOLT ET AL.	
	Examiner David Lazaro	**Art Unit** 2155	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
- Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *19 June 2002* .

2a)□ This action is **FINAL.** 2b)☒ This action is non-final.

3)□ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-30* is/are pending in the application.

4a) Of the above claim(s) *1-8* is/are withdrawn from consideration.

5)□ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *9-30* is/are rejected.

7)□ Claim(s) _____ is/are objected to.

8)□ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on _____ is/are: a)☒ accepted or b)□ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

11)□ The proposed drawing correction filed on _____ is: a)□ approved b)□ disapproved by the Examiner.

If approved, corrected drawings are required in reply to this Office action.

12)□ The oath or declaration is objected to by the Examiner.

Priority under 35 U.S.C. §§ 119 and 120

13)□ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)□ All b)□ Some * c)□ None of:

1.□ Certified copies of the priority documents have been received.

2.□ Certified copies of the priority documents have been received in Application No. _____ .

3.□ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

14)□ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application).

a) □ The translation of the foreign language provisional application has been received.

15)□ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) □ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☒ Information Disclosure Statement(s) (PTO-1449) Paper No(s) 4,5 .

4) □ Interview Summary (PTO-413) Paper No(s). _____ .

5) □ Notice of Informal Patent Application (PTO-152)

6) □ Other: .

DETAILED ACTION

Election/Restrictions

1. Restriction to one of the following inventions is required under 35 U.S.C. 121:

I. Claims 1-8, drawn to a method for determining a diameter of a broadcast channel, classified in class 709, subclass 238.

II. Claims 9-30, drawn to a method and computer-readable medium for disconnecting a first computer from a second computer with the first and second computer being connected to a broadcast channel, classified in class 709, subclass 227.

2. The inventions are distinct, each from the other because of the following reasons:

3. Inventions I and II are related as subcombinations disclosed as usable together in a single combination. The subcombinations are distinct from each other if they are shown to be separately usable. In the instant case, invention I has separate utility such as determining the broadcast channel diameter. Invention II has separate utility such as disconnecting a first computer from a second computer when they are connected to a broadcast channel. See MPEP § 806.05(d).

4. Because these inventions are distinct for the reasons given above and have acquired a separate status in the art as shown by their different classification, restriction for examination purposes as indicated is proper.

5. During a telephone conversation with Chun Ng of Perkins Coie LLP on 10/22/03 a provisional election was made without traverse to prosecute the invention of Group II, claims 9-30. Affirmation of this election must be made by applicant in replying to this Office action. Claims 1-8 are withdrawn from further consideration by the examiner, 37 CFR 1.142(b), as being drawn to a non-elected invention.

6. Applicant is reminded that upon the cancellation of claims to a non-elected invention, the inventorship must be amended in compliance with 37 CFR 1.48(b) if one or more of the currently named inventors is no longer an inventor of at least one claim remaining in the application. Any amendment of inventorship must be accompanied by a request under 37 CFR 1.48(b) and by the fee required under 37 CFR 1.17(i).

Papers Received

7. Oath/Declaration, basic filing fee, additional claim fee and late filing fee/oath or declaration surcharge were received on 11/06/00.

Information Disclosure Statement

8. IDS received on 4/23/02 has been considered by the examiner.

9. Supplemental IDS received on 6/19/02 has been considered by the examiner.

Specification

10. This application does not contain an abstract of the disclosure as required by 37 CFR 1.72(b). An abstract on a separate sheet is required.

Claim Rejections - 35 USC § 103

10. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

11. Claims 9, 10, 12, 15-18, 22-25 and 28-30 are rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent 6,618,752 Moore et al. (Moore) in view of U.S. Patent 6,353,599 by Bi et al (Bi).

12. With respect to Claim 9, Moore teaches a method of disconnecting a first computer from a second computer, the first computer being connected to a broadcast channel, the method comprising when the first computer decides to disconnect from the second computer, the first computer sends a disconnect message to the second computer (Col. 10 lines 35-40). Moore does not disclose the second computer broadcasting a connection port search message to find a third computer to which it can connect. However it is well known in the art that a connection port search message can be broadcast to find a computer to connect to as shown by Bi (Col. 2, line 14-20, Bi) (Claim 13 lines 48-51). It would have been obvious to one of ordinary skill in the art at the time the invention was made to take the method disclosed by Moore and modify as indicated by Bi with the step of when the second computer receives the disconnect message from the first computer, the second computer broadcasts a connection port search message to find a third computer to which it can connect. One would be motivated to do this as it provides

MM

an easy way to establish communications without previously knowing the specific port address (Col. 2 lines 34-37).

13. With respect to Claim 10, Moore in view of Bi further teaches the second computer receives a port connection message indicating the third computer is proposing that the third computer and the second computer connect (Col. 2 lines 51-55 of Bi).

14. With respect to Claim 12, Moore in view of Bi further teaches the broadcast channel is implemented using the Internet (Col. 4 lines 13-14 of Moore).

15. With respect to Claim 15, Moore in view of Bi further teaches the first computer and second computer are connect via a TCP/IP connection (Col. 4 lines 67 to Col. 5 line 2 of Moore).

16. With respect to Claim 16, Moore teaches a method for disconnecting a first computer from a second computer comprising connecting the first computer to a second computer, attempting to send a message from the first computer to the second computer and connecting to another computer when the communications fail (Col. 10 lines 48-52). Moore does not explicitly disclose broadcasting a connection port search message. However it is well known in the art that a connection port search message can be broadcast to find a computer to connect to as shown by Bi (Claim 13 lines 48-51), see also col. 2, lines 14-20 - Bi, WI. It would have been obvious to one of ordinary skill in the art at the time the invention was made to take the method disclosed by Moore and modify as indicated by Bi with the step of when the attempt to send the message is unsuccessful, broadcasting a connection port search message indicating that the first computer needs a connection. One would be motivated to do this as it provides an easy way to establish

communications without previously knowing the specific port address (Col. 2 lines 34-37).

17. With respect to Claim 17, Moore in view of Bi further teaches when a third computer receives the connection port search message and the third computer also needs a connection, sending a message from the third computer to the first computer proposing that the first computer and third computer connect (Col. 2 lines 51-55 of Bi).

18. With respect to Claim 18, Moore in view of Bi further teaches the first computer receives the message proposing that the first computer and third computer connect, sending from the first computer to the third computer a message indicating that the first computer accepts the proposal to connect to the first computer to the third computer (Col. 10 lines 42-47 of Moore).

19. With respect to Claim 22, Moore in view of Bi further teaches the broadcasting includes sending the message to each computer to which the first computer is connected (Col. 10 lines 4-7 of Moore).

20. With respect to Claim 23, Moore teaches a computer readable medium containing instructions for controlling disconnecting of a computer from another computer, the computer and the other computer being connected to a broadcast channel comprising a component that when the computer decides to disconnect from the other computer, the computer sends a disconnect message to the other computer (Col. 10 lines 35-40). Moore does not explicitly disclose a component that broadcasts a connection port search message to find a computer to which it can connect. However it is well known in the art that a connection port search message can be broadcast to find

a computer to connect to as shown by Bi (Claim 13 lines 48-51). It would have been obvious to one of ordinary skill in the art at the time the invention was made to take the computer-readable media containing instructions disclosed by Moore and modify it as indicated by Bi with a component when the computer receives a disconnect message from another computer, the computer broadcasts a connection port search message to find a computer to which it can connect. One would be motivated to do this as it provides an easy way to establish communications without previously knowing the specific port address (Col. 2 lines 34-37).

21. With respect to Claim 24, Moore in view of Bi further teaches a component that, when the computer receives a connection port search message and the computer needs to connect to another computer, sends to the computer that sent the connection port search message a port connection message indicating that the computer is proposing that the computer that sent the connection port search message connect to the computer (Col. 2 lines 51-55 or Bi).

22. With respect to Claim 25, Moore in view of Bi further teaches a component that, when the computer receives a port connection message, connecting to the computer that sent the port connection message (Col. 10 lines 4-7 of Moore).

23. With respect to Claim 28, Moore in view of Bi further teaches the computers are connected via a TCP/IP connection (Col. 4 lines 67 to Col. 5 line 2 of Moore).

24. With respect to Claim 29, Moore in view of Bi further teaches the computers that are connected to the broadcast channel are peers (Col. 3 line 52 to Col. 4 line 3 of Moore).

25. With respect to Claim 30, Moore in view of Bi further teaches the broadcast channel is implemented using the Internet (Col. 4 lines 13-14 of Moore).

26. Claims 13, 14, 19-21, 26 and 27 are rejected under 35 U.S.C. 103(a) as being unpatentable over Moore in view of Bi as applied to claims 9, 16 and 23 above, and further in view of "Graph Theory with Applications" by Bondy et al. (Bondy).

27. With respect to Claim 13, Moore in view of Bi teaches all the limitations of Claim 9 but does not explicitly disclose each computer connected to the broadcast channel network is connected to at least three other computers. However it is well known in the art that a communications network can be designed to be more reliable by increasing the connectivity and edge connectivity as shown in Bondy (Page 47-50 section 3.3). It would have been obvious to one of ordinary skill in the art at the time the invention was made to take the method disclosed by Moore in view of Bi and modify it as disclosed by Bondy with each computer connected to the broadcast channel is connected to at least three other computers. The larger amount of connectivity and edge connectivity, the more reliable the system becomes. Therefore the amount in terms of a reliable system is arbitrary. One would be motivated to have this as it provides reliability in a communications network where the failure of communications link can jeopardize communications of the system (Page 47-50 section 3.3).

28. With respect to Claim 14, Moore in view of Bi and in further view of Bondy further teach the computers and their connections form an m-regular graph for the same reason and motivation as described above in Claim 13.

29. With respect to Claim 19, Moore in view of Bi teaches all the limitations of Claim 16 but does not explicitly disclose each computer connected to the broadcast channel network is connected to at least three other computers. However it is well known in the art that a communications network can be designed to be more reliable by increasing the connectivity and edge connectivity as shown in Bondy (Page 47-50 section 3.3). It would have been obvious to one of ordinary skill in the art at the time the invention was made to take the method disclosed by Moore in view of Bi and modify it as disclosed by Bondy with each computer connected to the broadcast channel is connected to at least three other computers. The larger amount of connectivity and edge connectivity, the more reliable the system becomes. Therefore the amount in terms of a reliable system is arbitrary. One would be motivated to have this as it provides reliability in a communications network where the failure of communications link can jeopardize communications of the system (Page 47-50 section 3.3).

30. With respect to Claim 20, Moore in view of Bi and in further view of Bondy further teach the computers and their connections form an m-regular graph for the same reason and motivation as described above in Claim 19.

31. With respect to Claim 21, Moore in view of Bi and in further view of Bondy further teach the computers and their connections form an m-connected graph for the same reason and motivation as described above in Claim 19.

32. With respect to Claim 26, Moore in view of Bi teaches all the limitations of Claim 23 but does not explicitly disclose each computer connected to the broadcast channel network is connected to at least three other computers. However it is well known in the

art that a communications network can be designed to be more reliable by increasing the connectivity and edge connectivity as shown in Bondy (Page 47-50 section 3.3). It would have been obvious to one of ordinary skill in the art at the time the invention was made to take the computer-readable medium disclosed by Moore in view of Bi and modify it as disclosed by Bondy with each computer connected to the broadcast channel is connected to at least three other computers. The larger amount of connectivity and edge connectivity, the more reliable the system becomes. Therefore the amount in terms of a reliable system is arbitrary. One would be motivated to have this as it provides reliability in a communications network where the failure of communications link can jeopardize communications of the system (Page 47-50 section 3.3).

33. With respect to Claim 27, Moore in view of Bi and in further view of Bondy further teach the computers and their connections form an m-regular graph for the same reason and motivation as described above in Claim 19.

Conclusion

34. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

35. U.S. Patent 6,252,884 by Hunter "Dynamic Configuration of Wireless Networks" June 26, 2001.

36. U.S. Patent 6,073,177 by Hebel et al. “Dynamic Method for Connecting a Client to a Server Application” June 6, 2000.

37. U.S. Patent 5,946,316 by Chen et al. “Dynamic Distributed Mulitcast Routing Protocol” August 31, 1999.

38. Yavatkar et al. "A Reliable Dissemination Protocol for Interactive Collaborative Applications" Proc. ACM Multimedia, 1995 p.333-344

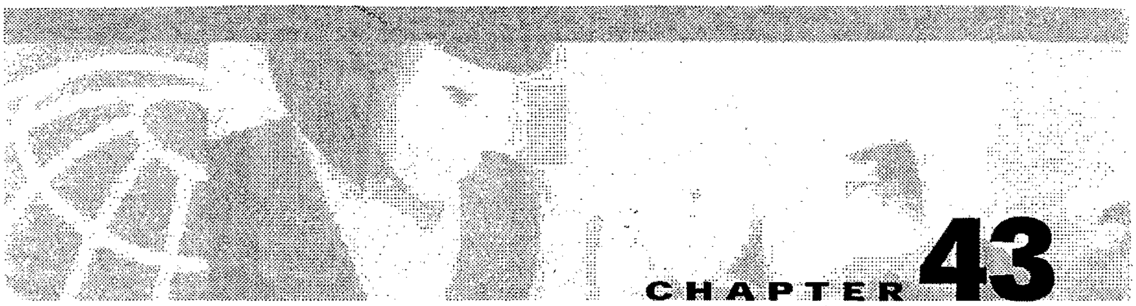
Any inquiry concerning this communication or earlier communications from the examiner should be directed to David Lazaro whose telephone number is 703-305-4868. The examiner can normally be reached on 8:30-5:00 M-F.

If attempts to reach the examiner by telephone are unsuccessful, the examiner’s supervisor, Hosain Alam can be reached on 703-308-6662. The fax phone number for the organization where this application or proceeding is assigned is (703) 872-9306.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is 703-305-3900.

David Lazaro
October 31, 2003

HAlam
HOSAIN ALAM
SUPERVISORY PATENT EXAMINER



CHAPTER 43

Chapter Goals

- Explain IP multicast addressing.
- Learn the basics of Internet Group Management Protocol (IGMP).
- Explain how multicast in Layer 2 switching works.
- Define multicast distribution trees.
- Learn how multicast forwarding works.
- Explain the basics of protocol-independent multicast (PIM).
- Define multiprotocol BGP.
- Learn how Multicast Source Discovery Protocol (MSDP) works.
- Explain reliable multicast: PGM.

Internet Protocol Multicast

Background

Internet Protocol (IP) multicast is a bandwidth-conserving technology that reduces traffic by simultaneously delivering a single stream of information to thousands of corporate recipients and homes. Applications that take advantage of multicast include videoconferencing, corporate communications, distance learning, and distribution of software, stock quotes, and news.

IP Multicast delivers source traffic to multiple receivers without adding any additional burden on the source or the receivers while using the least network bandwidth of any competing technology. Multicast packets are replicated in the network by Cisco routers enabled with Protocol Independent Multicast (PIM) and other supporting multicast protocols resulting in the most efficient delivery of data to multiple receivers possible. All alternatives require the source to send more than one copy of the data. Some even require the source to send an individual copy to each receiver. If there are thousands of receivers, even low-bandwidth applications benefit from using Cisco IP Multicast. High-bandwidth applications, such as MPEG video, may require a large portion of the available network bandwidth for a single stream. In these applications, the only way to send to more than one receiver simultaneously is by using IP Multicast. Figure 43-1 demonstrates how data from one source is delivered to several interested recipients using IP multicast.

Figure 43-1 Multicast Transmission Sends a Single Multicast Packet Addressed to All Intended Recipients

Multicast Group Concept

Multicast is based on the concept of a group. An arbitrary group of receivers expresses an interest in receiving a particular data stream. This group does not have any physical or geographical boundaries—the hosts can be located anywhere on the Internet. Hosts that are interested in receiving data flowing to a particular group must join the group using IGMP. Hosts must be a member of the group to receive the data stream.

IP Multicast Addresses

Multicast addresses specify an arbitrary group of IP hosts that have joined the group and want to receive traffic sent to this group.

IP Class D Addresses

The *Internet Assigned Numbers Authority (IANA)* controls the assignment of IP multicast addresses. It has assigned the old Class D address space to be used for IP multicast. This means that all IP multicast group addresses will fall in the range of 224.0.0.0 to 239.255.255.255.

Note This address range is only for the group address or destination address of IP multicast traffic. The source address for multicast datagrams is always the unicast source address.

Reserved Link Local Addresses

The IANA has reserved addresses in the 224.0.0.0 through 224.0.0.255 to be used by network protocols on a local network segment. Packets with these addresses should never be forwarded by a router; they remain local on a particular LAN segment. They are always transmitted with a time-to-live (TTL) of 1.

Network protocols use these addresses for automatic router discovery and to communicate important routing information. For example, OSPF uses 224.0.0.5 and 224.0.0.6 to exchange link state information. Table 43-1 lists some of the well-known addresses.

Table 43-1 Link Local Addresses

Address	Usage
224.0.0.1	All systems on this subnet
224.0.0.2	All routers on this subnet
224.0.0.5	OSPF routers
224.0.0.6	OSPF designated routers
224.0.0.12	DHCP server/relay agent

Globally Scoped Address

The range of addresses from 224.0.1.0 through 238.255.255.255 are called globally scoped addresses. They can be used to multicast data between organizations and across the Internet.

Some of these addresses have been reserved for use by multicast applications through IANA. For example, 224.0.1.1 has been reserved for Network Time Protocol (NTP).

More information about reserved multicast addresses can be found at http://www.isi.edu/in-notes/iana/assignments/multicast-addresses.

Limited Scope Addresses

The range of addresses from 239.0.0.0 through 239.255.255.255 contains limited scope addresses or administratively scoped addresses. These are defined by RFC 2365 to be constrained to a local group or organization. Routers are typically configured with filters to prevent multicast traffic in this address range from flowing outside an autonomous system (AS) or any user-defined domain. Within an autonomous system or domain, the limited scope address range can be further subdivided so those local multicast boundaries can be defined. This also allows for address reuse among these smaller domains.

Glop Addressing

RFC 2770 proposes that the 233.0.0.0/8 address range be reserved for statically defined addresses by organizations that already have an AS number reserved. The AS number of the domain is embedded into thc second and third octets of the 233.0.0.0/8 range.

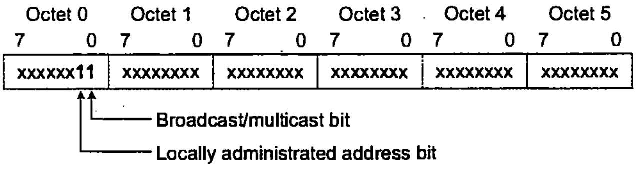
For example, the AS 62010 is written in hex as F23A. Separating out the two octets F2 and 3A, we get 242 and 58 in decimal. This would give us a subnet of 233.242.58.0 that would be globally reserved for AS 62010 to use.

Layer 2 Multicast Addresses

Normally, network interface cards (NICs) on a LAN segment will receive only packets destined for their burned-in MAC address or the broadcast MAC address. Some means had to be devised so that multiple hosts could receive the same packet and still be capable of differentiating among multicast groups.

Fortunately, the IEEE LAN specifications made provisions for the transmission of broadcast and/or multicast packets. In the 802.3 standard, bit 0 of the first octet is used to indicate a broadcast and/or multicast frame. Figure 43-2 shows the location of the broadcast/multicast bit in an Ethernet frame.

Figure 43-2 IEEE 802.3 MAC Address Format



This bit indicates that the frame is destined for an arbitrary group of hosts or all hosts on the network (in the case of the broadcast address, 0xFFFF.FFFF.FFFF).

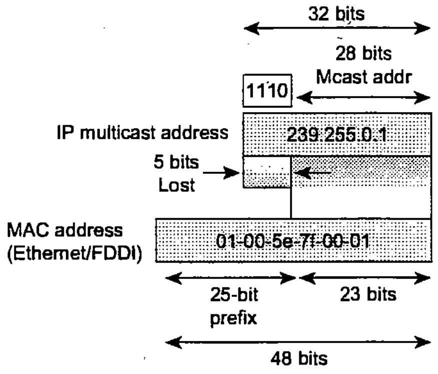
IP multicast makes use of this capability to transmit IP packets to a group of hosts on a LAN segment.

Ethernet MAC Address Mapping

The IANA owns a block of Ethernet MAC addresses that start with 01:00:5E in hexadecimal. Half of this block is allocated for multicast addresses. This creates the range of available Ethernet MAC addresses to be 0100.5e00.0000 through 0100.5e7f.ffff.

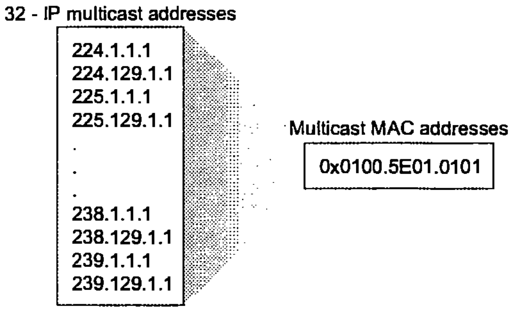
This allocation allows for 23 bits in the Ethernet address to correspond to the IP multicast group address. The mapping places the lower 23 bits of the IP multicast group address into these available 23 bits in the Ethernet address (shown in Figure 43-3).

Figure 43-3 Mapping of IP Multicast to Ethernet/FDDI MAC Address



Because the upper 5 bits of the IP multicast address are dropped in this mapping, the resulting address is not unique. In fact, 32 different multicast group IDs all map to the same Ethernet address (see Figure 43-4).

Figure 43-4 MAC Address Ambiguities



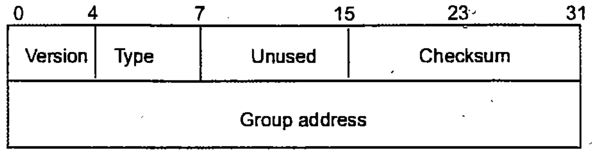
Internet Group Management Protocol

IGMP is used to dynamically register individual hosts in a multicast group on a particular LAN. Hosts identify group memberships by sending IGMP messages to their local multicast router. Under IGMP, routers listen to IGMP messages and periodically send out queries to discover which groups are active or inactive on a particular subnet.

IGMP Version 1

RFC 1112 defines the specification for IGMP Version 1. A diagram of the packet format is found in Figure 43-5.

Figure 43-5 IGMP Version 1 Packet Format



In Version 1, there are just two different types of IGMP messages:

- Membership query
- Membership report

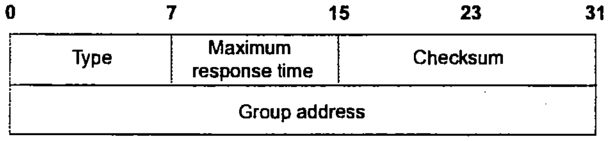
Hosts send out IGMP membership reports corresponding to a particular multicast group to indicate that they are interested in joining that group. The router periodically sends out an IGMP membership query to verify that at least one host on the subnet is still interested in receiving traffic directed to that group. When there is no reply to three consecutive IGMP membership queries, the router times out the group and stops forwarding traffic directed toward that group.

IGMP Version 2

RFC 2236 defines the specification for IGMP Version 2.

A diagram of the packet format follows in Figure 43-6.

Figure 43-6 IGMPv2 Message Format



In Version 2, there are four types of IGMP messages:

- Membership query
- Version 1 membership report
- Version 2 membership report
- Leave group

IGMP Version 2 works basically the samc as Version 1. The main difference is that there is a leave group message. The hosts now can actively communicate to the local multicast router their intention to leave the group. The router then sends out a group-specific query and determines whether there are any remaining hosts interested in receiving the traffic. If there are no replies, the router times out the group and stops forwarding the traffic. This can greatly reduce the leave latency compared to IGMP Version 1. Unwanted and unnecessary traffic can be stopped much sooner.

Multicast in the Layer 2 Switching Environment

The default behavior for a Layer 2 switch is to forward all multicast traffic to every port that belongs to the destination LAN on the switch. This would defeat the purpose of the switch, which is to limit traffic to the ports that need to receive the data.

Two methods exist by which to deal with multicast in a Layer 2 switching environment efficiently—Cisco Group Management Protocol (CGMP) and IGMP snooping.

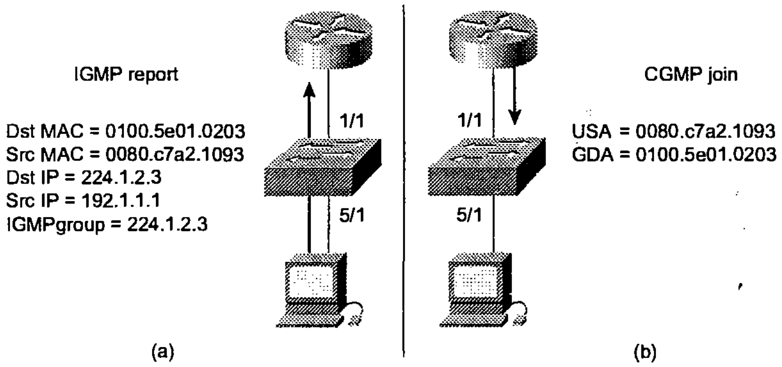
Cisco Group Management Protocol

CGMP is a Cisco-developed protocol that allows Catalyst switches to leverage IGMP information on Cisco routers to make Layer 2 forwarding decisions. CGMP must be configured both on the multicast routers and on the Layer 2 switches. The net result is that with CGMP, IP multicast traffic is delivered only to those Catalyst switch ports that are interested in the traffic. All other ports that have not explicitly requested the traffic will not receive it.

The basic concept of CGMP is shown in Figure 43-7. When a host joins a multicast group (part A), it multicasts an unsolicited IGMP membership report message to the target group (224.1.2.3, in this example). The IGMP report is passed through the switch to the router for the normal IGMP processing. The router (which must have CGMP enabled on this interface) receives this IGMP report and processes it as it normally would, but in addition it creates a CGMP join message and sends it to the switch.

The switch receives this CGMP join message and then adds the port to its content addressable memory (CAM) table for that multicast group. Subsequent traffic directed to this multicast group will be forwarded out the port for that host. The router port is also added to the entry for the multicast group. Multicast routers must listen to all multicast traffic for every group because the IGMP control messages are also sent as multicast traffic. With CGMP, the switch must listen only to CGMP join and CGMP leave messages from the router. The rest of the multicast traffic is forwarded using its CAM table exactly the way the switch was designed.

Figure 43-7 Basic CGMP Operation



IGMP Snooping

IGMP snooping requires the LAN switch to examine, or snoop, some Layer 3 information in the IGMP packets sent between the hosts and the router. When the switch hears the IGMP host report from a host for a particular multicast group, the switch adds the host's port number to the associated multicast table entry. When the switch hears the IGMP leave group message from a host, it removes the host's port from the table entry.

Because IGMP control messages are transmitted as multicast packets, they are indistinguishable from multicast data at Layer 2. A switch running IGMP snooping examine every multicast data packet to check whether it contains any pertinent IGMP must control information. If IGMP snooping has been implemented on a low-end switch with a slow CPU, this could have a severe performance impact when data is transmitted at high rates. The solution is to implement IGMP snooping on high-end switches with special ASICs that can perform the IGMP checks in hardware. CGMP is ideal for low-end switches without special hardware.

Multicast Distribution Trees

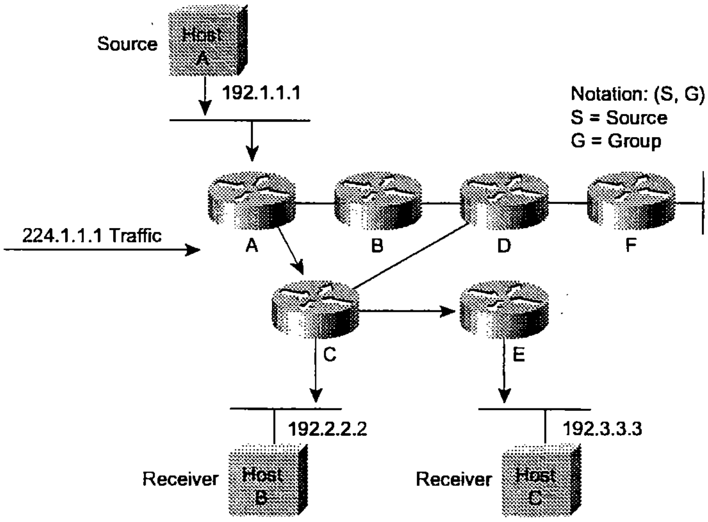
Multicast-capable routers create distribution trees that control the path that IP multicast traffic takes through the network to deliver traffic to all receivers. The two basic types of multicast distribution trees are source trees and shared trees.

Source Trees

The simplest form of a multicast distribution tree is a *source tree* whose root is the source of the multicast tree and whose branches form a spanning tree through the network to the receivers. Because this tree uses the shortest path through the network, it is also referred to as a shortest path tree (SPT).

Figure 43-8 shows an example of an SPT for group 224.1.1.1 rooted at the source, Host A, and connecting two receivers, hosts B and C.

Figure 43-8 Host A Shortest Path Tree



The special notation of (S,G), pronounced "S comma G," enumerates an SPT in which S is the IP address of the source and G is the multicast group address. Using this notation, the SPT for the example in Figure 43-7 would be (192.1.1.1, 224.1.1.1).

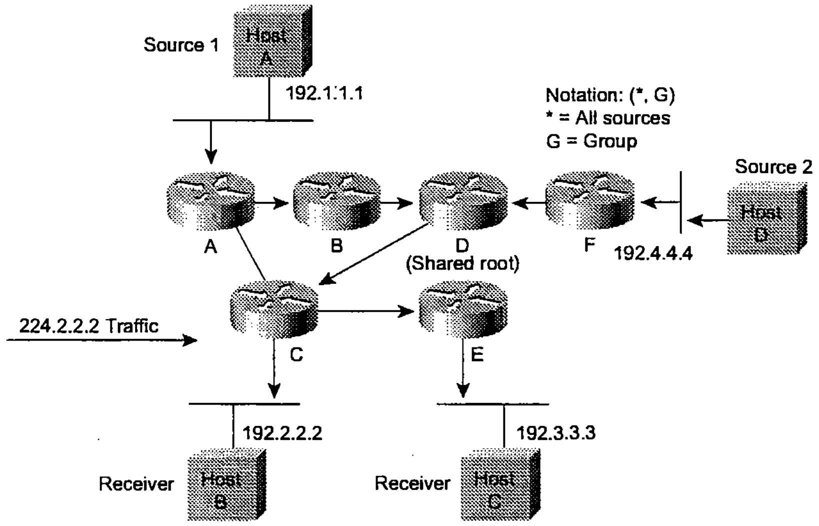
The (S,G) notation implies that a separate SPT exists for each individual source sending to each group, which is correct. For example, if Host B is also sending traffic to group 224.1.1.1 and hosts A and C are receivers, then a separate (S,G) SPT would exist with a notation of (192.2.2.2,224.1.1.1).

Shared Trees

Unlike source trees that have their root at the source, *shared trees* use a single common root placed at some chosen point in the network. This shared root is called the *rendezvous point (RP)*.

Figure 43-9 shows a shared tree for the group 224.2.2.2 with the root located at Router D. When using a shared tree, sources must send their traffic to the root, and then the traffic is forwarded down the shared tree to reach all receivers.

Figure 43-9 Shared Distribution Tree



In this example, multicast traffic from the source hosts A and D travels to the root (Router D) and then down the shared tree to the two receivers, hosts B and C. Because all sources in the multicast group use a common shared tree, a wildcard notation written as (*, G), pronounced "star comma G," represents the tree. In this case, * means all sources, and the G represents the multicast group. Therefore, the shared tree shown in Figure 43-8 would be written as (*, 224.2.2.2).

Both SPT and shared trees are loop-free. Messages are replicated only where the tree branches.

Members of multicast groups can join or leave at any time, so the distribution trees must be dynamically updated. When all the active receivers on a particular branch stop requesting the traffic for a particular multicast group, the routers prune that branch from the distribution tree and stop forwarding traffic down that branch. If one receiver on that branch becomes active and requests the multicast traffic, the router dynamically modifies the distribution tree and starts forwarding traffic again.

Shortest path trees have the advantage of creating the optimal path between the source and the receivers. This guarantees the minimum amount of network latency for forwarding multicast traffic. This optimization does come with a price, though: The routers must maintain path information for each source. In a network that has thousands of sources and thousands of groups, this can quickly become a resource issue on the routers. Memory consumption from the size of the multicast routing table is a factor that network designers must take into consideration.

Shared trees have the advantage of requiring the minimum amount of state in each router. This lowers the overall memory requirements for a network that allows only shared trees. The disadvantage of shared trees is that, under certain circumstances, the paths between the source and receivers might not be the optimal paths—which might introduce some latency in packet delivery. Network designers must carefully consider the placement of the RP when implementing an environment with only shared trees.

Multicast Forwarding

In unicast routing, traffic is routed through the network along a single path from the source to the destination host. A unicast router does not really care about the source address—it only cares about the destination address and how to forward the traffic towards that destination. The router scans through its routing table and then forwards a single copy of the unicast packet out the correct interface in the direction of the destination.

In multicast routing, the source is sending traffic to an arbitrary group of hosts represented by a multicast group address. The multicast router must determine which direction is upstream (toward the source) and which direction (or directions) is downstream. If there are multiple downstream paths, the router replicates the packet and forwards the traffic down the appropriate downstream paths—which is not necessarily all paths. This concept of forwarding multicast traffic away from the source, rather than to the receiver, is called *reverse path forwarding.*

Reverse Path Forwarding

Reverse path forwarding (RPF) is a fundamental concept in multicast routing that enables routers to correctly forward multicast traffic down the distribution tree. RPF makes use of the existing unicast routing table to determine the upstream and downstream neighbors. A router forwards a multicast packet only if it is received on the upstream interface. This RPF check helps to guarantee that the distribution tree will be loop-free.

RPF Check

When a multicast packet arrives at a router, the router performs an RPF check on the packet. If the RPF check is successful, the packet is forwarded. Otherwise, it is dropped.

For traffic flowing down a source tree, the RPF check procedure works as follows:

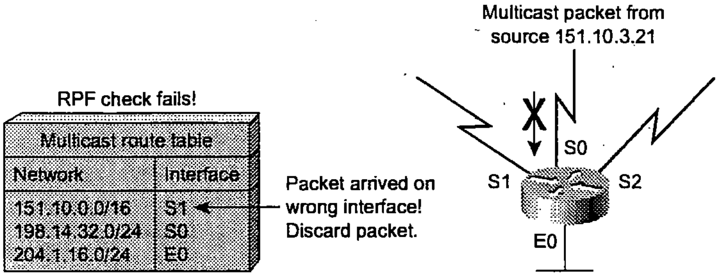
Step 1 Router looks up the source address in the unicast routing table to determine whether it has arrived on the interface that is on the reverse path back to the source.

Step 2 If packet has arrived on the interface leading back to the source, the RPF check is successful and the packet is forwarded.

Step 3 If the RPF check in Step 2 fails, the packet is dropped.

Figure 43-10 shows an example of an unsuccessful RPF check.

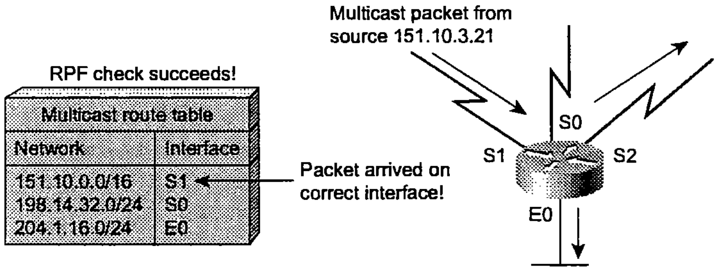
Figure 43-10 RPF Check Fails



A multicast packet from source 151.10.3.21 is received on interface S0. A check of the unicast route table shows that the interface that this router would use to forward unicast data to 151.10.3.21 is S1. Because the packet has arrived on S0, the packet will be discarded.

Figure 43-11 shows an example of a successful RPF check.

Figure 43-11 RPF Check Succeeds



This time the multicast packet has arrived on S1. The router checks the unicast routing table and finds that S1 is the correct interface. The RPF check passes and the packet is forwarded.

Protocol-Independent Multicast

Protocol-independent multicast (PIM) gets its name from the fact that it is IP routing protocol-independent. PIM can leverage whichever unicast routing protocols are used to populate the unicast routing table, including EIGRP, OSPF, BGP, or static routes. PIM uses this unicast routing information to perform the multicast forwarding function, so it is IP protocol-independent. Although PIM is called a multicast routing protocol, it actually uses the unicast routing table to perform the reverse path forwarding (RPF) check function instead of building up a completely independent multicast routing table. PIM does not send and receive multicast routing updates between routers like other routing protocols do.

PIM Dense Mode

PIM Dense Mode (PIM-DM) uses a push model to flood multicast traffic to every corner of the network. This is a brute-force method for delivering data to the receivers, but in certain applications, this might be an efficient mechanism if there are active receivers on every subnet in the network.

PIM-DM initially floods multicast traffic throughout the network. Routers that do not have any downstream neighbors prune back the unwanted traffic. This process repeats every 3 minutes.

The flood and prune mechanism is how the routers accumulate their state information—by receiving the data stream. These data streams contain the source and group information so that downstream routers can build up their multicast forwarding tables. PIM-DM can support only source trees—(S,G) entries. It cannot be used to build a shared distribution tree.

PIM Sparse Mode

PIM Sparse Mode (PIM-SM) uses a pull model to deliver multicast traffic. Only networks that have active receivers that have explicitly requested the data will be forwarded the traffic. PIM-SM is defined in RFC 2362.

PIM-SM uses a shared tree to distribute the information about active sources. Depending on the configuration options, the traffic can remain on the shared tree or switch over to an optimized source distribution tree. The latter is the default behavior for PIM-SM on Cisco routers. The traffic starts to flow down the shared tree, and then routers along the path determine whether there is a better path to the source. If a better, more direct path exists, the designated router (the router closest to the receiver) will send a join message toward the source and then reroute the traffic along this path.

PIM-SM has the concept of an RP, since it uses shared trees—at least initially. The RP must be administratively configured in the network. Sources register with the RP, and then data is forwarded down the shared tree to the receivers. If the shared tree is not an optimal path between the source and the receiver, the routers dynamically create a source tree and stop traffic from flowing down the shared tree. This is the default behavior in IOS. Network administrators can force traffic to stay on the shared tree by using a configuration option (lp pim spt-threshold infinity).

PIM-SM scales well to a network of any size, including those with WAN links. The explicit join mechanism prevents unwanted traffic from flooding the WAN links.

Sparse-Dense Mode

Cisco has implemented an alternative to choosing just dense mode or just sparse mode on a router interface new IP. This was necessitated by a change in the paradigm for forwarding multicast traffic via PIM that became apparent during its development. It turned out that it was more efficient to choose sparse or dense on a per group basis rather than a per router interface basis. Sparse-dense mode facilitates this ability.

Network administrators can also configure sparse-dense mode. This configuration option allows individual groups to be run in either sparse or dense mode, depending on whether RP information is available for that group. If the router learns RP information for a particular group, it will be treated as sparse mode; otherwise, that group will be treated as dense mode.

Multiprotocol Border Gateway Protocol

Multiprotocol Border Gateway Protocol (MBGP) gives a method for providers to distinguish which route prefixes they will use for performing multicast RPF checks. The RPF check is the fundamental mechanism that routers use to determine the paths that multicast forwarding trees will follow and successfully deliver multicast content from sources to receivers.

MBGP is described in RFC 2283, Multiprotocol Extensions for BGP-4. Since MBGP is an extension of BGP, it brings along all the administrative machinery that providers and customers like in their interdomain routing environment. Including all the inter-AS tools to filter and control routing (e.g., route maps). Therefore, by using MBGP, any network utilizing internal or external BGP can apply the multiple policy control knobs familiar in BGP to specify routing (and thereby forwarding) policy for multicast.

Two path attributes, MP_REACH_NLRI and MP_UNREACH_NLRI have been introduced in BGP4+. These new attributes create a simple way to carry two sets of routing information—one for unicast routing and one for multicast routing. The routes associated with multicast routing are used to build the multicast distribution trees.

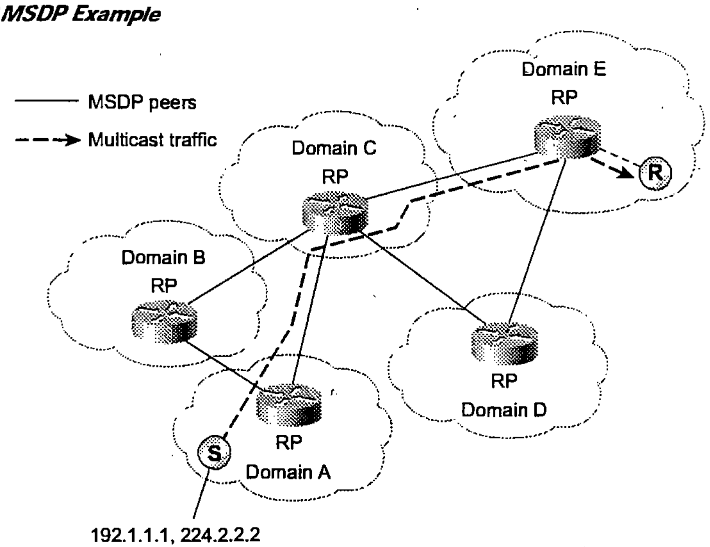
The main advantage of MBGP is that an internet can support noncongruent unicast and multicast topologies. When the unicast and multicast topologies are congruent, MBGP can support different policies for each. MBGP provides a scalable policy based interdomain routing protocol.

Multicast Source Discovery Protocol

In the PIM Sparse mode model, multicast sources and receivers must register with their local Rendezvous Point (RP). Actually, the closest router to the sources or receivers registers with the RP but the point is that the RP knows about all the sources and receivers for any particular group. RPs in other domains have no way of knowing about sources located in other domains. MSDP is an elegant way to solve this problem. MSDP is a mechanism that connects PIM-SM domains and allows RPs to share information about active sources. When RPs in remote domains know about active sources they can pass on that information to their local receivers and multicast data can be forwarded between the domains. A nice feature of MSDP is that it allows each domain to maintain an independent RP which does not rely on other domains, but it does enable RPs to forward traffic between domains.

The RP in each domain establishes an MSDP peering session using a TCP connection with the RPs in other domains or with border routers leading to the other domains. When the RP learns about a new multicast source within its own domain (through the normal PIM register mechanism), the RP encapsulates the first data packet in a Source Active (SA) message and sends the SA to all MSDP peers. The SA is forwarded by each receiving peer using a modified RPF check, until it reaches every MSDP router in the interconnected networks—theoretically the entire multicast internet. If the receiving MSDP peer is an RP, and the RP has a (*,G) entry for the group in the SA (there is an interested receiver), the RP will create (S,G) state for the source and join to the shortest path tree for the state of the source. The encapsulated data is decapsulated and forwarded down that RP's shared tree. When the packet is received by a receiver's last hop router, the last-hop may also join the shortest path tree to the source. The source's RP periodically sends SAs, which include all sources within that RP's own domain. Figure 43-12 shows how data would flow between a source in domain A to a receiver in domain E.

Figure 43-12 MSDP Example



MDSP was developed for peering between Internet Service Providers (ISPs). ISPs did not want to rely on an RP maintained by a competing ISP to service their customers. MSDP allows each ISP to have their own local RP and still forward and receive multicast traffic to the Internet.

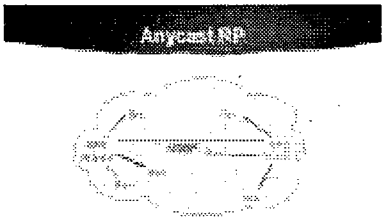
Anycast RP-Logical RP

A very useful application of MSDP is called anycast RP. This is a technique for configuring a multicast sparse-mode network to provide for fault tolerance and load sharing within a single multicast domain.

Two or more RPs are configured with the same IP address on loopback interfaces—say, 10.0.0.1, for example (refer to Figure 43-13). The loopback address should be configured as a 32 bit address. All the downstream routers are configured so that they know that their local RP's address is 10.0.0.1. IP routing automatically selects the topologically closest RP for each source and receiver. Because some sources might end up using one RP and some receivers a different RP, there needs to be some way for the RPs to exchange information about active sources. This is done with MSDP. All the RPs are configured to be MSDP peers of each other. Each RP will know about the active sources in the other RP's area. If any of the RPs fail, IP routing will converge and one of the RPs will become the active RP in both areas.

Note The Anycast RP example above uses IP addresses from RFC 1918. These IP addresses are normally blocked at interdomain borders and therefore are not accessible to other ISPs. You must use valid IP addresses if you want the RPs to be reachable from other domains.

Figure 43-13 Anycast RP



Note The RPs are used only to set up the initial connection between sources and receivers. After the last-hop routers join the shortest path tree, the RP is no longer necessary.

Multicast Address Dynamic Client Allocation Protocol

The *Multicast Address Dynamic Client Allocation Protocol (MADCAP)* is defined in RFC 2730 as a protocol that allows hosts to request a multicast address allocation dynamically from a MADCAP server. The concept is very similar to the way DHCP works today and is built on a client/server model.

Multicast-Scope Zone Announcement Protocol

Multicast-Scope Zone Announcement Protocol (MZAP) is defined in RFC 2776 as a protocol that allows networks to automatically discover administratively scoped zones relative to a particular location.

Reliable Multicast-Pragmatic General Multicast

Pragmatic General Multicast (PGM) is a reliable multicast transport protocol for applications that require ordered, duplicate-free, multicast data delivery from multiple sources to multiple receivers. PGM guarantees that a receiver in a multicast group either receives all data packets from transmissions and retransmissions, or can detect unrecoverable data packet loss.

The PGM Reliable Transport Protocol itself is implemented on the sources and the receivers. The source maintains a transmit window of outgoing data packets and retransmits individual packets when it receives a negative acknowledgment (NAK). The network elements (routers) assist in suppressing an implosion of NAKs (when a failure does occur) and aids in efficient forwarding of the retransmitted data just to the networks that need it.

PGM is intended as a solution for multicast applications with basic reliability requirements. The specification for PGM is network layer-independent. The Cisco implementation of PGM Router Assist supports PGM over IP.

Today, the specification for PGM is an Internet draft that can be found on the IETF web site (http://www.ietf.org) under the name "PGM Reliable Transport Protocol."

Review Questions

Q—*What is the range of available IP multicast addresses?*

A—224.0.0.0 to 239.255.255.255.

Q—*What is the purpose of IGMP?*

A—IGMP is used between the hosts and their local multicast router to join and leave multicast groups.

Q—*What is an advantage of IGMPv2 over IGMPv1?*

A—IGMPv2 has a leave group message that can greatly reduce the latency of unwanted traffic on a LAN.

Q—*What is a potential disadvantage of IGMP snooping over CGMP on a low-end Layer 2 switch?*

A—IGMP snooping requires the switch to examine every multicast packet for an IGMP control message. On a low-end switch, this might have a severe performance impact.

Q—*What is an advantage of shortest path (or source) trees compared to shared trees?*

A—Source trees guarantee an optimal path between each source and each receiver, which will minimize network latency.

Q—*What is an advantage of using shared trees?*

A—Shared trees require very little state to be kept in the routers, which requires less memory.

Q—*What information does the router use to do an RPF check?*

A—The unicast routing table.

Q—*Why is protocol-independent multicast called "independent"?*

A—PIM works with any underlying IP unicast routing protocol—RIP, EIGRP, OSPF, BGP or static routes.

Q—*What is the main advantage of MBGP?*

A—Providers can have noncongruent unicast and multicast routing topologies.

Q—*How do RPs learn about sources from other RPs with MSDP?*

A—RPs are configured to be MSDP peers with other RPs. Each RP forwards source active (SA) messages to each other.

Q—*What is the purpose of the anycast RP?*

A—Load balancing and fault tolerance.

For More Information

Williamson, Beau. *Developing IP Multicast Networks.* Indianapolis: Cisco Press, 2000.

Multicast Quick Start Configuration Guide (http://www.cisco.com/warp/customer/105/48.html)

By Patricia A. Murphy,
Manager Emerging Technology and Standards,
Rockwell Automation Control Systems.

The Next Generation Networking Paradigm: Producer/Consumer Model

As today's source/destination-based networks cannot offer the required functionality or accommodate increased traffic, system capabilities and productivity improvements are restricted. Consequently, a new network model - one that provides more functionality, makes more efficient use of bandwidth, and increases information flow, all while reducing traffic on the wire - is needed.

In a discussion of what is needed in the new network model regarding diagnostics, explicit and I/O messaging and throughput, the producer/consumer network model is revealed as the only model available today that can meet the control environment's demanding requirements and allow for future migration. The paper concludes with a discussion of the benefits of the producer/consumer network model, including Multicast and two one-way I/O trigger mechanisms: change-of-state and cyclic I/O production.

INTRODUCTION

If there's one thing we've all learned over the past decade, it's that users are demanding more from their control systems, and consequently, from the networks that tie the system together. Users want better diagnostics available over the network, less downtime, and reduced installation and maintenance costs. At the same time, they are demanding improved throughput.

With increased functionality comes more traffic and data on the wire. Today's networks, which are source/destination based, cannot offer the required functionality and accommodate increased traffic, thus restricting system capabilities and productivity improvements.

Increased demands on networks have forced the evolution of a new network model - one that provides more functionality, makes more efficient use of bandwidth, and increases information flow, all while reducing traffic on the wire.

Unfortunately, much of the discussion to date about networks has focused on baud rates, protocol efficiency, and physical characteristics (i.e. type of wire used). In reality, it's more complex than that. Available diagnostics, messaging types and throughput must all be considered when evaluating a network.

The most important factor affecting these capabilities in the control environment is the network model.

The source/destination model used for the past two decades can no longer meet today's network needs. The only model available today that can meet these demanding requirements and allows for future migration is the producer/consumer network model. Here's why:

DIAGNOSTICS

Networks provide a convenient way to retrieve diagnostics from devices. Device-specific information, such as detection of a photoelectric sensor's low margin due to a dirty lens can be communicated over the network to the control system during run time. The network delivers the diagnostic to the system operator interface, alerting plant personnel to the problem. The lens can be cleaned at a convenient time before there is a glitch in the process. Trouble-shooting a device, reading its fault codes, updating data logs -- all while not impacting the remote I/O control data exchange among other nodes -- is a must.

EXPLICIT AND I/O MESSAGING

Explicit messages, used for device configuration and diagnostics, are extremely flexible, with the data field carrying protocol information and instructions for service performance. For example, a message would be able to write new presets to five timers in a controller, or to execute a self-test. Explicit messages are used for uploading and downing programs, modifying device configurations, and data logging, trending and diagnostic functions. Nodes must interpret each message, perform the requested task, and generate responses. These types of messages are highly variable in both size and frequency.

I/O messages on the other hand are implicit in nature. The data field contains no protocol information, only real-time I/O control data. The meaning of the data has been predefined and processing time in the node is minimized. An example of an I/O message is a controller sending output data to an I/O block, and the I/O block responding with its input data. Such messages are low overhead, short, vary frequently and require high performance.

In the past, manufacturers have had separate networks to deal with the very different requirements of these two messaging types. A network used for I/O control cannot tolerate the variability introduced by explicit messaging. Allen-Bradley's blue-hose duo, DH+/RIO and Siemens' Profibus FMS/ Profibus DP are examples of this situation.

Today's users are demanding both functions on the same wire. And today's smarter devices need the functionality provided by both messaging types. Yesterday's source/destination networks cannot deal with these modern demands.

THROUGHPUT

Ultimately it's the throughput required by the application that determines what type of network model is required. Throughput is the rate at which input data from devices can be delivered to all nodes that need it and the resulting output data (decisions) can be delivered to all the devices that need it. Nodes include sensors, operator interfaces, controllers, data loggers, alarm monitors, actuators, etc. It is determined by baud rate, protocol efficiency, and most important of all, the network model, or delivery method. Let's briefly discuss each.

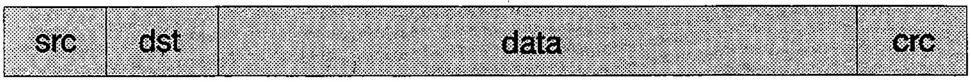


Figure 1.

Baud rate is raw speed. It's unfortunate that this is often the most used measure of performance because it's the most misleading. Not only that, but with today's new networks, it's the least important of the three throughput factors.

Protocol efficiency - data bytes (the payload) versus the total bytes in the packet - typically expressed as a percent, is a measure of the network protocol overhead. While important, it is not nearly as significant as the data delivery and exchange method (network model) used. If a particular information exchange takes two or more packets on the wire as compared to one, the fact the one protocol has 25 percent greater efficiency becomes meaningless.

Network model. Every control vendor has its own favorite networks, whether it be Data Highway Plus, Remote I/O, Profibus FMS, Profibus DP, Interbus-S, ASI, Modbus Plus, GeniusLan, or Lonworks. All these networking options have the same thing in common. They follow the legacy source/destination network model. A typical packet is shown below.

In master/slave implementations of this model, the source field is usually not present, as the master is the only source and all responses from slaves are for the master. This master/slave polling is inherently a one-to-one data exchange. It is typically used for the exchange of real time control data (I/O messaging). When used for I/O exchange, such networks are typically limited to that function alone to obtain the necessary repeatability and throughput for control.

Peer-to-peer networking goes beyond master/slave, providing considerably more flexibility. But as a result most networks that support peer-to-peer use explicit messaging.

PC-based programming and configuration of controllers uses explicit peer-to-peer messaging. PC-based MMI also use explicit peer-to-peer messaging. As additional MMI units are added, the network load increases dramatically as each unit typically will read all the same variables out of each controller as does the prior units so an operator can get the same alarms, trends, and graphics from multiple locations.

To keep nodes from dominating the wire, most peer-to-peer networks use some sort of token rotation algorithm. While these algorithms have been enhanced over the years to be more "fair, the basic flexibility that makes it attractive makes its use for peer-to-peer interlocking between controllers very problematic. Response times vary considerably for any given message, depending on load and on how "far away" one is from the token holder when there is a need to speak.

Frequently low-end electronic operator interface (EOI) units will be found on I/O networks, basically replacing simple push button, pilot light and meters. But as each EOI device is added, an additional load of typically the same data new node with a different destination address is added to the network. While variability isn't a factor because of the fixed nature of such loads, the increase in data load slows response time for all nodes, including the real I/O. It's not just EOI that's causing excess network loading. As I/O devices get smarter, the extra diagnostic and configuration data can absorb considerable bandwidth.

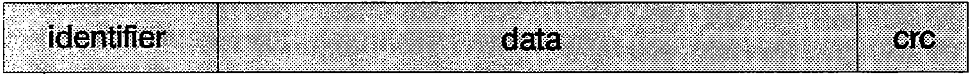


Figure 2.

Whether master/slave or peer-to-peer, destination-oriented networks waste considerable bandwidth sending the same data set to multiple nodes. Trying to do coordinated control like sending a new setpoint to different drives in a synchronized manner is very difficult, as data arrives at each drive at a different time.

The new network paradigm: The producer/consumer network model

To manage the growing need for data, smarter devices and better control, new networks that simply increase the baud rate or number of nodes only postpones the inevitable. What is needed is a whole new network model that is designed to manage today's control issues.

That new model is producer/consumer. With producer/consumer networks, messages are identified by content. If a node needs data, it will "accept" that identifier and consume it.

Multicast. Because data is identified by its content; if a node needs that data, multiple nodes can consume the same data at the same time from a single producer. Nodes may be synchronized more precisely while achieving more efficient bandwidth usage. The source of data has to produce the information only once. Additional EOI and MMI can be added without increasing network traffic, since they can consume these same messages. And nodes can produce more than one data set, each using a unique identifier.

Multicast is inherently impossible with source/destination networks, although attempts have been made. Some have added a third field for a group destination and then reserved node numbers for group destination. Others allow a node to carry more than one node number. But these are all band-aid approaches, desperately trying to extend the exhausted legacy source/destination model.

Producer/consumer also allows for two new powerful I/O triggers, in addition to traditional polling. Polling is born out of the source/destination model, and is inherently a two message bi-directional transaction (originator sends output data, and receiving node responds with input data). This transaction is repeated as rapidly as possible to minimize latency from when an input occurs and is delivered to the controller. Most polling cycles are filled with the same outputs and inputs, wasting bandwidth.

With the producer/consumer model, two more efficient and effective one-way I/O trigger mechanisms are available: change-of-state and cyclic I/O production.

Change-of-state (event-based) data production. Nodes produce data only when that data changes. There is no "network polling cycle delay," and, as a result, the data is delivered to all consumers when it changes. A background heartbeat is produced cyclically so that consumers can tell if a device hasn't changed from one that is no longer online. Change-of-state can dramatically reduce network traffic and the load-on typically needed to repeatedly receive, process and generate the same data.

Cyclic (time-based) data production. Cyclic data production involves nodes producing data at a user-configured rate. Data is updated at a rate appropriate to the node and the application. Data can be sampled and produced by sensors at precise intervals for better PID control. Controllers can collect a stock of data for operator interfaces and produce it a couple times a second, plenty fast for human consumption; thereby preserving bandwidth for nodes with rapidly changing I/O.

Both peer-to-peer and controller-to-device exchanges can be handled more efficiently with cyclic and change-of-state data production of producer/consumer networks. Operator interface needs can be layered on top of I/O traffic with minimal increases to network load.

At the same time, producer/consumer networks can accommodate the flexible explicit messaging, point-to-point needs for device configuration and programming. Certain identifiers are typically specified for such traffic and nodes know they contain destination and other protocol information. These identifiers, coupled with the network access method, combine to insure that explicit messages, with their assorted larger overhead are much lower priority on the network than the I/O messages. Large uploads and downloads adjustments to configurations parameters, and diagnostic activities by users with their S/W tools are relegated to background traffic, fitted between the higher priority I/O messages.

No need for users to run both an I/O network and explicit message network through the plant. And no need for vendors to put an I/O port and an explicit message network port on devices. Is it any wonder that the newest open control networks -- DeviceNet, ControlNet, and FOUNDATION Fieldbus -- are all based on the producer/consumer model?

Will the source/destination model disappear?

Source/destination is a "hand me down" from the computer and data processing industry. While limited, source/destination systems are still well-suited for a variety of applications which do not require complex coordination and sharing of data. The flexibility and efficiency of the producer/consumer network model will allow for the expanded functionality demanded by today's applications and is well suited for tomorrow's smarter devices. In this day in age where users are demanding more (functionality, diagnostics) with less (one wire, not two), both users and vendors need a control networking strategy that works smarter -- and, consequently, a network model that works smarter and accomodates the future

Patricia A. Murphy is manager for Emerging Technology and Standards at Rockwell Automation Control Systems.

Murphy's responsibilities include:

- *Integrating advanced technology into Rockwell Automation Control System's Communications Business.*
- *Coordinating advanced technology projects.*
- *Participating in industry consortia such as Fieldbus Foundation, ODVA and ControlNet International.*
- *Leading business standards activity and participating in international standards committees as appropriate.*

Murphy has many years of marketing and product management experience in technology driven industries including automation, with Rockwell Automation Control Systems, and telecommunications, with Ericsson and GE.

The Gamer's Guide

the Guide | the Glossary | Who's Who

First-Person Shooters

General

There are many different varieties of computer games on the market today. From flight simulators to real-time strategy, all of these varieties have created niche markets within the huge population of computer gamers. One of the fastest growing game-types is the First-Person Shooter. This subcatagory of action games, also called over-the-shoulder shooters and other names, is based upon a relatively simple premise: You see the world through the eyes of your character. The goal of such games also tend to be very simple: You run down darkened hallways and kill anything that gets in your way. However, reducing the genre to simplistic elements fails to appreciate the reasons that these games are so popular. First-person shooters have lead the industry in implementing the latest technology, providing the greatest support for multi-player gaming, and generating the greatest amount of praise and condemnation.

History

The birth of the first-person shooter can be traced back to id Software's Wolfenstein 3-D, released in shareware on May 5, 1992. Using a very strong graphics engine developed in-house, this game allowed players to see an amazing landscape through the eyes of the main character. The graphics may appear dated now after less than a decade, but the ability to move along the x and y axis in a graphically rendered hallway was a revelation. So was the feeling that the perspective gave the game. Players who were used to seeing small, animated characters jump up and down in side-scrolling games were amazed at the immersive feel of Wolfenstein. You could see the weapon that you were carrying and could see it fire. Life-like Nazis and vicious dogs attacked you from every corner.

If Wolfenstein is the father of the first-person shooter, then Doom would be his most popular descendent. Doom, also by id Software, moved the action from the darkened hallways of a German castle during World War II to futuristic, although still darkened, hallways with a score of alien monsters. The game was an immediate success. This can be attributed to both the technical and aesthetic improvements in the game as well as the timing of its release. Doom, released on December 10, 1993, entered the scene just as modem speeds and awareness of the internet were increasing and the corporate and academic worlds were embracing the Local Area Network (LAN) . Its ability to support convincing multi-player games catapulted it into the forefront of the gaming community.

Other software companies were quick to recognize the appeal of Doom and the first-person shooters. Many licensed the graphics engine from id and began creating their own games. These "Doom clones" enjoyed great success. Games such as Rise of the Triad and LucasArt's Dark Forces created worlds that rivaled the world of Doom. This competition encouraged companies to find new ways to exploit the Doom engine. One result was beautifully rendered games with innovative ideas, like Raven Software's Heretic. The other result was increasingly graphic violence and a move towards parental ratings. Apogee, and its subsidiary 3D Realms, have ironically been at the forefront of both controversial content and moves to inform parents of the content.

Doom was followed by the highly successful Doom II. This game is viewed by many as the ultimate development of the Doom engine. However, the game still lacked true 3D. id Software then announced the development of Quake, a true 3D game. Development of Quake lasted 18 months, giving competitors an opportunity to develop competing technology. 3D Realms took this opportunity. Using the Build graphics engine, created by the amazing Ken Silverman, 3D Realms developed Duke Nukem 3D. This game wasn't really a 3D game, but the graphics engine was versatile enough to make the illusion of 3D almost seamless. Duke was released several months before Quake and quickly garnered a huge following. Many of the innovations contained in this game are currently being used by other software companies.

The release of Quake marked the true beginning of the 3D age of first-person shooters, at least in the traditional sense that we use the terms 3D and first-person shooter. Descent was the first title to actually use 3D successfully, with impressive and popular results, but Quake was the first title that used 3D in the traditional first-person shooter. With its superb graphics engine, quality levels and monsters, and multi-player support for 16 players, Quake allowed id Software to regain its crown as the king of the first-person shooter. Amazingly, Quake remained the technology leader for a year and a half. This has as much to do with the appeal of the game as with id's decision to release enough tools and information to allow users to modify Quake. Using a language called Quake C, programmers could alter virtually every element of the game. The creation of powerful level editors and a host of utilities, all by the growing "Quake community," paved the way for spectacular new games, new levels, and new ideas.

LucasArts' Jedi Knight was a revolutionary game on many levels. Until this title was released in mid 1997, it was laughable to talk about the "storyline" in a first-shooter. Jedi Knight took advantage of the rich Star Wars history to create a unique and entertaining game. Through the use of cutscenes and a non-linear story, LucasArts provided a game that compelled a player to finish.

The release of Quake II in December 1997 pushed the technology envelop even further. Many of the features that had been pioneered during Quake's ongoing development, such as OpenGL support, transparent water, and the client prediction of QuakeWorld, were included in this new game. Quake II also added nifty visual effects, like 16 bit textures in GL mode and colored lighting, but was also the first id title to have a strong storyline. Granted, the storyline still involved you running around killing everything, but the player was provided with specific tasks for each level.

Just as Duke Nukem 3D and Jedi Knight had done previously, the release of Unreal has had a profound impact on a community that had been entirely focused upon titles from id. Unreal's strength lies not in its storyline, which tended to disappear in the middle of the game, but in the power of its graphics engine. The engine adds many visual improvements, allowed for incredibly large levels, and is still very easy to modify. A wide range of titles, from hunting games to adventure games, are currently being developed with the Unreal engine. Many have seen this as a real threat to the empire of id.

Fans of the Quake series are looking forward to the release of Quake III Arena. This game has gone through a few profound changes since it was first proposed, but the many new visual effects and gameplay elements that have been proposed for the game are very impressive.

1999 will undoubtedly see a continued evolution of the first-person shooter. Several companies are currently developing games utilizing the graphics engine from id Software's Quake 2. Several other companies are using the popular engine from Unreal to design their games. Still others have created, or are creating, their own graphics engine. It is impossible to determine how successful these efforts will be in an increasingly crowded field. Regardless, it is clear that the first-person shooter will remain a large and powerful genre in computer gaming.

Technology

As a group, first-person shooters tend to be driving forces in accelerating the development of computer software and hardware. Other types of games are now trying to keep up, but FPSs have a clear lead in the technology race. The rise of the 3D graphics accelerator cannot be accredited to business applications; it is rooted firmly in computer games. It could be argued that 3Dfx, the company that created the Voodoo chipset used by several graphics cards, owes much of its success to GL Quake.

First-person shooters have also pioneered the mutli-player gaming experience. Artificial intelligence with games continue to increase, but many players find monsters a poor substitute for playing against real people.

Controversy

Just as FPSs lead the way in technology, they also have cornered the market in controversy. Very few games contain as much graphic violence as a first-person shooter. Many first-person shooters have also received criticism for their sexist portrayal of women, satanic overtones, and racial stereotypes. Many voices have been raised in defense, and in condemnation, of these controversial games. The course of first-person shooters in this area remains uncertain. id Software removed all Satanic references from Quake 2, but the program was banned for sale in Germany for its violent content. 3D Realms created a "parental lock" for Duke Nukem 3D, an effort to allow parents to monitor a child's use of the game, but their is little doubt that the strippers and prostitutes will return for Duke Nukem Forever. Some software companies appear to be concerned about reducing controversy. Others appear to go out of their way to generate controversy. This division will likely remain as long segments of the gaming community are drawn towards each approach. In the end, the actual playability of a game will generate more interest, and more sales, than controversy alone.

[Basics] [Servers] [Beyond basics] [Other versions of Quake] [Sources of information] [Citizenship] [Quake C] [Level editing] [Mods] [Quake derivatives] [Operating Systems] [Guide News]

Last update: October 20, 1998
This page is maintained by Darren L. Tabor,
aka Dakota

Published on **The O'Reilly Network** (http://www.oreillynet.com/)
http://www.oreillynet.com/pub/a/p2p/2001/01/25/truelove0101.html
See this if you're having trouble printing code examples

Gnutella: Alive, Well, and Changing Fast

01/25/2001

Gnutella, an open peer-to-peer search system primarily used for file sharing, was released in March. Within four months, developer activity had substantially diminished, although usage continued to surge due to Napster-driven media attention on peer-to-peer file-sharing systems. After five months, the strain of an increasing number of users on a weak technical infrastructure resulted in a quasi-collapse of the Gnutella network. Late in the year, however, a second wave of more sophisticated development began to emerge, informed by experience. Defying reports of its demise, Gnutella is evolving and usage is growing in response, although significant technical challenges remain.

What problems have been overcome and how? What problems remain to be solved, and how can they be addressed? Clip2's Distributed Search Solutions initiative has continuously gathered data on the Gnutella network and closely followed related application development. Here, we cover some representative issues to provide insight into Gnutella's evolution.

Related Articles:

In Praise of Freeloaders

P2P Directory

Mojo Nation Responds

Open-Source Roundtable: Free Riding on Gnutella 🔊

Gnutella and Freenet Represent True Technological Innovation

Remaking the Peer-to-Peer Meme

More from OpenP2P.com ▶

The origins and technical significance of Gnutella have been described elsewhere. Some notable points:

- Gnutella's creators released an executable application and published neither its source code nor the communications protocol. Extant protocol publications made by third parties trace their primary sources to reverse-engineerings of the original application.
- It is generally acknowledged that Gnutella was not designed to support an unlimited user population, but instead a few hundred to perhaps a few thousand users.
- The Gnutella protocol defines five message types, the data carried by each type, the transmission rules for each type and the mechanics of connection between hosts.
- Pings and queries used to discover hosts and files, respectively, are broadcast; other message types, including responses, are routed. Messages are supposed to be dropped after a predefined number of relays.
- Gnutella is not a file-transfer protocol. The protocol is designed for finding hosts and their files. File transfer is handled directly between serving and requesting hosts via HTTP. Gnutella applications that serve files contain mini Web servers.
- The protocol does not specify how many connections a given host may initiate, accept or simultaneously maintain. It does not dictate conditions under which a host should maintain or drop a given connection.
- Many independent developers have produced a number of Gnutella-speaking applications.

It is not hard to imagine from the foregoing that Gnutella is susceptible to a number of problems.

Non-compliance

Non-compliant implementations are problematic not just for their users, who may not be able to effectively communicate with others, but they are also trouble for the network at large. Because Gnutella messages are relayed from host to host, the impact of a non-compliant application can easily extend beyond its installed base and be magnified out of proportion.

But, what does "non-compliant" mean for a protocol without a blessed standard? In the open world of Gnutella, free from central authority, compliance means being able to effectively communicate with the bulk of the installed base. It is not unlike the situation with languages such as English that have no formal codification. Protocol specification documents in this environment then become analogous to dictionaries that reflect popular usage instead of dictating usage.

Of course, non-compliance can arise out of the purposeful invention of new words or simply out of poor grammar and pronunciation. Among the many ways an application can go wrong on the latter front: It can malform messages it originates, it can corrupt messages it forwards and it can improperly route messages. Proper handling of the routed message types by creating and maintaining a routing table is a feature that, when short-shrifted by a developer, results in substantial costs to users, including increased traffic and lost responses. The low barriers to entry to Gnutella programming have encouraged less experienced developers to try their hands, often exacerbating matters.

Non-compliant implementations have been kept in check by, among other things, the availability of quality protocol specification documents, and the strict filtering implemented in popular applications in order to not propagate deviant messages. They represent a continued problem.

Connectivity

Connectivity was a big headache for users. Just as a Web browser needs a start page, a Gnutella application needs a start host. Unfortunately, early programs did not come preset with one because host addresses generally have short shelf lives. This sent users searching across Web sites, message boards and chat rooms for active host addresses. Developer Bob Schmidt came to the rescue with gnuCache, an open-source application that automatically began doling out addresses from several enthusiast-run servers. Not long after, Clip2 began reliably serving lists of well connected, verified active hosts through a service that could be accessed at Gnutellahosts.com via Gnutella and the Web. By fall, developers had begun providing an "auto-connect" feature in Gnutella applications that relied upon host list services for start hosts, relieving users of the need to bother with this matter. Technically, these services are sufficiently uniform in the way they operate so as to be interchangeable to the developer, although the quality of addresses returned varies. The connectivity problem has thus been addressed in a manner that is not susceptible to a single point of failure.

Lack of Search Results

A lack of search results was a substantial issue following the quasi-collapse of the network in August. As the traffic carried by an average host grew, it eventually exceeded the capacity of hosts on the slowest physical links -- dial-up modems. These hosts became bottlenecks in the network, effectively severing communication lines running through them. Fragmentation into smaller sub-networks effectively resulted, with the upshot that users saw fewer search results.

Responses to the issue followed a common theme: move users on slower connections to the edge of the network.

In October, Clip2 introduced the Reflector, a special Gnutella server designed to run on a high-speed connection and act as a proxy for users on slower links. In so doing it conserves the user's bandwidth and situates slower hosts at the edge of the network. Via a Reflector, a network of users can use Gnutella with far less aggregate bandwidth than would otherwise be required. Most Reflectors are run on behalf of a particular user population and not publicly advertised, although a handful of public-access ones are available at any given time.

November and December saw the introduction of two significant new Gnutella applications. First, Lime Wire LLC introduced LimeWire, then Free Peers, Inc. released BearShare. Both programs apply connection-preferencing rules that decide whether a given connection will be maintained. One common example: connections to unresponsive hosts are dropped. The consistent repeated application of this simple rule to a series of connections will tend to drive slower hosts to have fewer connections and sit at the edge of the network, a bit like a poor conversationalist might find himself marginalized at a party.

Coincident with these developments and the uptake in adoption of these applications, Clip2 has seen a steady increase in the number of responsive hosts active at any given time on the network, rising from a typical figure of 500 in October to more than 1500 in early January 2001. The quantity of search results has increased as well. According to Clip2 estimates, the number of Gnutella users per day has risen from 10,000 to 30,000 in November to between 20,000 and 50,000 in January.

Download Failure

Download failure looms as one of the most serious problems according to many. Although attractive search results may come back, they are useless if the associated files cannot be downloaded. Quantitative study of the problem is complicated since users have preferences in the files they download and upload. Since all files are not equal, there is much room for inaccuracy in the results of any test that assumes otherwise. Nonetheless, there is a preponderance of perception that downloads fail too often, particularly relative to other peer-to-peer file-sharing systems.

Spurred by an August 2000 paper by Eytan Adar and Bernardo Huberman of Xerox PARC, there is belief that "freeloading" - users downloading much more than they upload - is a major source of the download failure problem, although the critical ratio of supply and demand is anyone's guess. The response to commentator Clay Shirky's counterpoint that "bandwidth over time is infinite" is that the server bandwidth available to users who want to download a file right now is too finite.

Developers are taking two major actions:

1. removing as much friction as possible from the upload process, such as defaulting a user's upload directory to be his download directory; and
2. blocking uploads to users who are not themselves uploading.

Web sites such as Gnute and Gnutella.it allow users to search and download from the Gnutella network without providing a direct means of contributing files back into the network. Seizing on this asymmetry, recent versions of LimeWire and BearShare have taken the offensive of denying download requests from Web site users. Instead of the file, the user finds a suggestion to download LimeWire or BearShare. The next step in this evolution may be to prioritize download requests even among users of these applications based on how much they have uploaded or made available for upload. The situation begins to resemble the model of Mojo Nation, in which downloading has a cost that is payable by providing resources back into the community. An alternative approach that might potentially be effective would be to not advertise a file -- not respond to a query for it -- so long as there is no bandwidth available to serve the file.

"Busy signals" are not the only possible cause for download failure. Hosts may be unreachable due to firewalls or intervening network address translation devices, applications may be buggy or incompatible, hosts may go offline or change their content between the point of advertising a file and the point of receiving a download request, and so on. A mechanism that enabled hosts to verify each other's ability to upload any file would address some of these issues.

What Next?

"What next?" is a fitting conclusion, for it is a problem that looms over Gnutella's future. Non-compliant implementations, connectivity, a lack of search results, download failure - these are all nuts-and-bolts problems with Gnutella. Sorting them out is necessary for Gnutella to meet commonly held basic expectations of it as a usable, public, decentralized file-sharing system. What happens when these core issues are sufficiently resolved?

The answer is that users spur developers to push on to new features. But which features? The trouble of "What's next?" is the contentious issue of agreeing on what problems need to be solved. Some aspire to see Gnutella be more scalable or more secure. Some want the system to be more anonymous, some want it to be less. Some hope it becomes a more generalized distributed search medium and grow beyond its file-sharing origins. Some imagine other applications riding upon it, even commerce. It seems there is no end to the expectations.

Unfortunately, Gnutella has a history of aborted, failed or poorly supported attempts to unite developers; the analogy of herding cats has rarely been so apt. One of the most notable efforts -- Gnutella Next Generation -- never significantly advanced beyond the proposal stage. Media reports have confused a spin-off effort known as gPulp as a Gnutella organization, but as the principal behind it has recently stated, "We are not a working group on Gnutella."

As of this writing, then, there is no clear leader in terms of a working group or other form of organization. There is, however, one arbiter of innovations: the market. Gnutella developers who have experimented with "improvements" that run counter to, outsid, or in between the lines of the de facto protocol have been kept in check by the fact that their applications must be able to communicate with those produced by other developers.

When the developer of an application known as Gnotella wanted to place more information in search-response messages than existing protocol specifications called for, he made sure he did it in a way that could still be passed on by the original Gnutella application, which was dominant in terms of user base at the time. Some other applications regarded Gnotella's search responses as noncompliant and dropped or otherwise "mishandled" the messages. The ability of Gnotella users to respond to queries was impaired, but the degree of impairment depended on the popularity of applications that regarded Gnotella as noncompliant. This story is being repeated with BearShare, which has recently released a version that also places extra information in search responses.

Will this market-driven pattern continue, so that Gnutella evolves in a competitive, Darwinian, decentralized and bottom-up manner? Or will it "grow up" and follow the trajectory of many other protocols, evolving through top-down committee processes? Only time will tell.

Kelly Truelove is the founder and CEO of Clip2, where he has led the company's efforts on P2P systems, distributed search, and Gnutella. He is a speaker at the upcoming Peer to Peer and Web Services Conference.

Related Articles:

In Praise of Freeloaders

P2P Directory

Mojo Nation Responds

Open-Source Roundtable: Free Riding on Gnutella 🔊

Gnutella and Freenet Represent True Technological Innovation

Remaking the Peer-to-Peer Meme

Discuss this article in the O'Reilly Network General Forum.

Return to the P2P DevCenter.

3.3.5 Find, for all $\nu \geq 5$, a 2-connected graph G of diameter two with $\varepsilon = 2\nu - 5$.

(Murty, 1969 has shown that every such graph has at least this number of edges.)

REFERENCES

Halin, R. (1969). A theorem on n-connected graphs. *J. Combinatorial Theory*, **7**, 150–54

Harary, F. (1962). The maximum connectivity of a graph. *Proc. Nat. Acad. Sci. U.S.A.*, **48**, 1142–46

Murty, U. S. R. (1969). Extremal nonseparable graphs of diameter 2, in *Proof Techniques in Graph Theory* (ed. F. Harary), Academic Press, New York, pp. 111–18

Whitney, H. (1932). Non-separable and planar graphs. *Trans. Amer. Math. Soc.*, **34**, 339–62

4 Euler Tours and Hamilton Cycles

4.1 EULER TOURS

A trail that traverses every edge of G is called an *Euler trail* of G because Euler was the first to investigate the existence of such trails in graphs. In the earliest known paper on graph theory (Euler, 1736), he showed that it was impossible to cross each of the seven bridges of Königsberg once and only once during a walk through the town. A plan of Königsberg and the river Pregel is shown in figure 4.1*a*. As can be seen, proving that such a walk is impossible amounts to showing that the graph of figure 4.1*b* contains no Euler trail.

A *tour* of G is a closed walk that traverses each edge of G at least once. An *Euler tour* is a tour which traverses each edge exactly once (in other words, a closed Euler trail). A graph is *eulerian* if it contains an Euler tour.

Theorem 4.1 A nonempty connected graph is eulerian if and only if it has no vertices of odd degree.

Proof Let G be eulerian, and let C be an Euler tour of G with origin (and terminus) u. Each time a vertex v occurs as an internal vertex of C, two of the edges incident with v are accounted for. Since an Euler tour contains

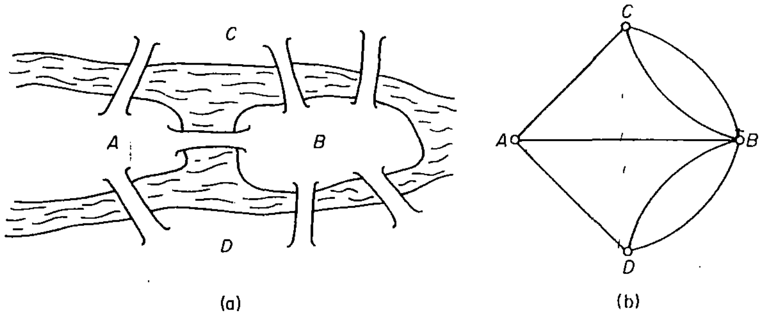


Figure 4.1. The bridges of Königsberg and their graph

A Reliable Dissemination Protocol for Interactive Collaborative Applications

Rajendra Yavatkar, James Griffioen, and Madhu Sudan
Department of Computer Science
University of Kentucky
Lexington, KY 40506
{raj,griff,madhu}@dcs.uky.edu
(606) 257-3961

ABSTRACT

The widespread availability of networked multimedia workstations and PCs has caused a significant interest in the use of collaborative multimedia applications. Examples of such applications include distributed shared whiteboards, group editors, and distributed games or simulations. Such applications often involve many participants and typically require a specific form of multicast communication called *dissemination* in which a single sender must reliably transmit data to multiple receivers in a timely fashion. This paper describes the design and implementation of a reliable multicast transport protocol called *TMTP* (Tree-based Multicast Transport Protocol). TMTP exploits the efficient best-effort delivery mechanism of IP multicast for packet routing and delivery. However, for the purpose of scalable flow and error control, it dynamically organizes the participants into a hierarchical control tree. The control tree hierarchy employs *restricted nacks with suppression* and an *expanding ring search* to distribute the functions of state management and error recovery among many members, thereby allowing scalability to large numbers of receivers. An Mbone-based implementation of TMTP spanning the United States and Europe has been tested and experimental results are presented.

KEYWORDS

Reliable Multicast, Transport Protocols, Mbone, Interactive Multipoint Services, Collaboration

INTRODUCTION

Widespread availability of IP multicast [6, 2] has substantially increased the geographic span and portability of collaborative multimedia applications. Example applications include distributed shared whiteboards [15], group editors [7, 14], and distributed games or simulations. Such applications often involve a large number of participants and are interactive in nature with participants dynamically joining and leaving the applications. For example, a large-scale conferencing application (e.g., an IETF presentation) may involve hundreds of people who listen for a short time and then leave the conference. These applications typically require a specific form of multicast delivery called *dissemination*. Dissemination involves 1xN communication in which a single sender must reliably multicast a significant amount of data to multiple receivers. IP multicast provides scalable and efficient routing and delivery of IP packets to multiple receivers. However, it does not provide the reliability needed by these types of collaborative applications.

Our goal is to exploit the highly efficient best-effort delivery mechanisms of IP multicast to construct a scalable and efficient protocol for reliable dissemination. Reliable dissemination on the scale of tens or hundreds of participants scattered across the Internet requires carefully designed flow and error control algorithms that avoid the many potential bottlenecks. Potential bottlenecks include host processing capacity [18] and network resources. Host processing capacity becomes a bottleneck when the sender must maintain state information and process incoming acknowledgements and retransmission requests from a large number of receivers. Network resources become a bottleneck unless the frequency and scope of retransmissions is limited. For instance, loss of packets due to congestion in a small portion of the IP multicast tree should not lead to retransmission of packets to all the receivers. Frequent multicast retransmissions of packets also wastes valuable network bandwidth.

This paper describes the design and implementation of a reliable dissemination protocol called *TMTP* (Tree-based Multicast Transport Protocol) that includes the following features:

1. TMTP takes advantage of IP multicast for efficient

packet routing and delivery.

2. TMTP uses an *expanding ring search* to dynamically organize the dissemination group members into a *hierarchical control tree* as members join and leave a group.

3. TMTP achieves scalable reliable dissemination via the hierarchical control tree used for flow and error control. The control tree takes the flow and error control duties normally placed at the sender and distributes them across several nodes. This distribution of control also allows error recovery to proceed independently and concurrently in different portions of the network.

4. Error recovery is primarily driven by receivers who use a combination of *restricted negative acknowledgements with nack suppression* and periodic positive acknowledgements. In addition, the tree structure is exploited to restrict the scope of retransmissions to the region where packet loss occurs; thereby insulating the rest of the network from additional traffic.

We have completed a user-level implementation of TMTP based on IP/UDP multicast and have used it for a systematic performance evaluation of reliable dissemination across the current Internet Mbone. Our experiments involved as many as thirty group members located at several sites in the US and Europe. The results are impressive; TMTP meets our objective of scalability by significantly reducing the sender's processing load, the total number of retransmissions that occur, and the end-to-end latency as the number of receivers is increased.

Background

A considerable amount of research has been reported in the area of group communication. Several systems such as the ISIS system [1], the V kernel [4], Amoeba, the Psynch protocol [17], and various others have proposed group communication primitives for constructing distributed applications. However, all of these systems support a general group communication model (NxN communication) designed to provide reliable delivery with support for atomicity and/or causality or to simply support an unreliable, unordered multicast delivery. Similarly, transport protocols specifically designed to support group communication have also been designed before [13, 5, 3, 19, 9]. These protocols mainly concentrated on providing reliable broadcast over local area networks or broadcast links. Flow and error control mechanisms employed in networks with physical layer multicast capability are simple and do not necessarily scale well to a wide area network with unreliable packet delivery.

Earlier multicast protocols used conventional flow and error control mechanisms based on a *sender-initiated* approach in which the sender disseminates packets and uses either a *Go-Back-N* or a *selective repeat* mechanism for error recovery. If used for reliable dissemination of information to a large number of receivers, this approach has several limitations. First, the sender must maintain and process a large amount of state information associated with each receiver. Second, the approach can lead to a *packet implosion* problem where a large number of ACKs or NACKs must be received and processed by the sender over a short interval. Overall, this can lead to severe bottlenecks at a sender resulting in an overall decrease in throughput [18].

An alternate approach based on *receiver-initiated* methods [19, 15] shifts the burden of reliable delivery to the receivers. Each receiver maintains state information and explicitly requests retransmission of lost packets by sending negative acknowledgements (NACKs). Under this approach, the receiver uses two kinds of timers. The first timer is used to detect lost packets when no new data is received for some time. The second timer is used to delay transmission of NACKs in the hope that some other receiver might generate a NACK (called *nack suppression*).

It has been shown that the receiver-initiated approach reduces the bottleneck at the sender and provides substantially better performance [18]. However, the receiver-initiated approach has some major drawbacks. First, the sender does not receive positive confirmation of reception of data from all the receivers and, therefore, must continue to buffer data for long periods of time. The second and most important drawback is that the end-to-end delay in delivery can be arbitrarily large as error recovery solely depends on the timeouts at the receiver unless the sender periodically polls the receivers to detect errors [19]. If the sender sends a train of packets and if the last few packets in the train are lost, receivers take a long time to recover causing unnecessary increases in end-to-end delay. Periodic polling of all receivers is not an efficient and practical solution in a wide area network. Third, the approach requires that a NACK must be multicast to all the receivers to allow suppression of NACKs at other receivers and, similarly, all the retransmissions must be multicast to all the receivers. However, this can result in unnecessary propagation of multicast traffic over a large geographic area even if the packet losses and recovery problems are restricted to a distant but small geographic area[1]. Thus, the approach may unnecessarily waste valuable bandwidth.

In this paper we present an alternative approach that achieves scalable reliable dissemination by reducing the processing bottlenecks of sender-initiated approaches

[1] Assume that only a distant portion of the Internet is congested resulting in packet loss in the area. One or more receivers in this region may multicast repeated NACKS that must be processed by all the receivers and the resulting retransmissions must also be forwarded to and processed by all the receivers.

and avoiding the long recovery times of receiver-initiated approaches.

OVERVIEW OF OUR APPROACH

Under the TMTP dissemination model, a single sender multicasts a stream of information to a *dissemination group*. A *dissemination group* consists of processes scattered throughout the Internet, all interested in receiving the same data feed. A session directory service (similar to the session directory *sd* from LBL [12]) advertizes all active dissemination groups.

Before a transmitting process can begin to send its stream of information, the process must create a dissemination group. Once the dissemination group has been formed, interested processes can dynamically join the group to receive the data feed. The dissemination protocol does not provide any mechanism to insure that all receivers are present and listening before transmission begins. Although such a mechanism may be applicable in certain situations, we envision a highly dynamic dissemination system in which receiver processes usually join a data feed already in progress and/or leave a data feed prior to its termination. Consequently, the protocol makes no effort to coordinate the sender and receivers, and an application must rely on an external synchronization method when such coordination is necessary.

For the purposes of flow and error control, TMTP organizes the group participants into a hierarchy of subnets or *domains*. Typically, all the group members in the same subnet belong to a domain and a single *domain manger* acts as a representative on behalf of the domain for that particular group. The domain manager is responsible for recovering from errors and handling local retransmissions if one or more of the group members within its domain do not receive some packets.

In addition to handling error recovery for the local domain, each domain manager may also provide error recovery for other domain managers in its vicinity. For this purpose, the domain managers are organized into a *control tree* as shown in Figure 1. The sender in a dissemination group serves as the root of the tree and has at most K domain managers as children. Similarly, each domain manager will accept at most K other domain managers as children, resulting in a tree with maximum degree K. The value of K is chosen at the time of group creation and registration and does *not* include local group members in a domain (or subnet). The degree of the tree (K) limits the processing load on the sender and the internal nodes of the control tree. Consequently, the protocol overhead grows slowly, proportional to the Log_K(Number_Of_Receivers).

Packet transmission in TMTP proceeds as follows. When a sender wishes to send data, TMTP uses IP multicast to transmit packets to the entire group. The transmission rate is controlled using a sliding window based protocol described later. The control tree ensures reliable delivery to each member. Each node of the control tree (including the root) is only responsible for handling the errors that arise in its immediate K children. Likewise, children only send periodic, positive acknowledgments to their immediate parent. When a child detects a missing packet, the child multicasts a NACK in combination with nack suppression. On the receipt of the NACK, its parent in the control tree multicasts the missing packet. To limit the scope of the multicast NACK and the ensuing multicast retransmission, TMTP uses the *Time-To-Live* (TTL) field to restrict the transmission radius of the message. As a result, error recovery is completely localized. Thus, a dissemination application such as a world-wide IETF conference would organize each geographic domain (e.g., the receivers in California vs. all the receivers in Australia) into separate subtrees so that error recovery in a region can proceed independently without causing additional traffic in other regions. TMTP's hierarchical structure also reduces the end-to-end delay because the retransmission requests need not propagate all the way back to the original sender. In addition, locally retransmitted packets will be received quickly by the affected receivers.

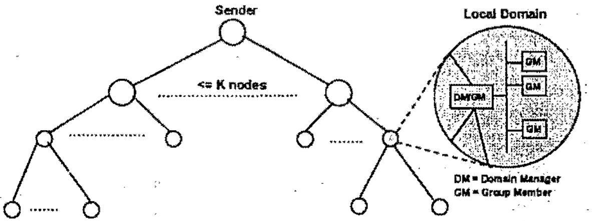


Figure 1: An example control tree with the maximum degree of each node restricted to K. Local group members within a domain are indicated by GM. There is no restriction on the number of local group members within a domain.

The control tree is self-organizing and does not rely on any centralized coordinator, being built dynamically as members join and leave the group. A new domain manager attaches to the control tree after discovering the closest node in the tree using an *expanded ring search*. Note that ***the control tree is built solely at the transport layer and thus does not require any explicit support from, or modification to, the IP multicast infrastructure inside the routers.***

The following sections describe the details of the TMTP protocol.

GROUP MANAGEMENT

The session directory provides the following group management primitives:

CreateGroup(GName,CommType): A sender creates a new group (with identifier *GName*) using the CreateGroup routine. *CommType* specifies the type of communication pattern desired and may be either *dissemination* or *concast*[2]. If successful, CreateGroup returns an IP multicast address and a port number to use when transmitting the data.

JoinGroup(Gname): Processes that want to receive the data feed represented by *GName* call JoinGroup to become a member of the group. Join returns the transport level address (IP multicast address and port number) for the group which the new process uses to listen to the data feed.

LeaveGroup(Gname): Removes the caller from the dissemination group *GName*.

DeleteGroup(GName): When the transmission is complete, the sending process issues a DeleteGroup request to remove the group *GName* from the system. DeleteGroup also informs all participants, and domain managers that the group is no longer active.

CONTROL TREE MANAGEMENT

Each dissemination group has an associated control tree consisting of domain managers. Over the lifetime of the dissemination group, the control tree grows and shrinks dynamically in response to additions and deletions to and from the dissemination group membership. Specifically, the tree grows whenever the first process in a domain joins the group (i.e., a domain manager is created) and shrinks whenever the last process left in a domain leaves the group (i.e., a domain manager terminates).

There are only two operations associated with control tree management: *JoinTree* and *LeaveTree*. When a new domain manager is created, it executes the JoinTree protocol to become a member of the control tree. Likewise, domain managers that no longer have any local processes to support may choose to execute the LeaveTree protocol.

Figures 2 and 3 outline the protocols for joining and leaving the control tree. The join algorithm employs an *expanding ring search* to locate potential connection points into the control tree. A new domain manager begins an expanding ring search by multicasting a SEARCH_FOR_PARENT request message with a small time-to-live value (TTL). The small TTL value restricts the scope of the search to nearby control nodes by limiting the propagation of the multicast message. If the manager does not receive a response within some fixed timeout period, the manager resends the SEARCH_FOR_PARENT message using a larger TTL value. This process repeats until the manager receives a WILLING_TO_BE_PARENT message from one or more domain managers in the control tree. All existing domain managers that receive the SEARCH_FOR_PARENT message will respond with a

```
While (NotDone) {
   Multicast a SEARCH_FOR_PARENT msg
   Collect responses
   If (no responses)
      Increment TTL /* try again */
   Else
      Select closest respondent as parent
      Send JOIN_REQUEST to parent
      Wait for JOIN_CONFIRM reply
      If (JOIN_CONFIRM received)
          NotDone = False
      Else       /* try again */
}
```

(A) New Domain Manger Algorithm

```
Receive request message
If (request is SEARCH_FOR_PARENT)
    If (MAX_CHILDREN not exceeded)
       Send WILLING_TO_BE_PARENT msg
    Else
       /* Do not respond */
Else If (request is JOIN_REQUEST)
    Add child to the tree
    Send JOIN_CONFIRM msg
```

(B) Existing Domain Manger Algorithm

Figure 2: The protocol used by domain managers to join the control tree. A new domain manager performs algorithm (A) while all other existing managers execute algorithm (B).

[2] Although this paper focuses on dissemination, TMTP also supports efficient concast style communication[10].

```
If (I_am_a_leaf_manager)
   Send LEAVE_TREE request
    to parent
   Receive LEAVE_CONFIRM
   Terminate
Else /* I am an internal manager */
   Fulfill all pending obligations
   Send FIND_NEW_PARENT message to children
   Receive FIND_NEW_PARENT reply from all children
   Send LEAVE_TREE request to parent
   Receive LEAVE_CONFIRM
   Terminate
```

Figure 3: The algorithm used to leave the control tree after the last local group member terminates.

WILLING_TO_BE_PARENT message unless they already support the maximum number of children. The new domain manager then selects the closest domain manager (based on the TTL values) and directly contacts the selected manager to become its child. For each domain, its manager maintains a *multicast radius* for the domain, which is the TTL distance to the farthest child within the domain. The domain manager keeps the children informed of the current multicast radius. As described later in the description of the error control part of TMTP, both parent and its children in a domain use the current multicast radius to restrict the scope of their multicast transmissions.

Before describing the LeaveTree protocol, note that a domain manager typically has two types of children. First, a domain manager supports the group members that reside within its local domain. Second, a domain manager may also act as a parent to one or more children domain managers. We say a manager is an *internal manager* of the tree if it has other domain managers as children. We say a manager is a *leaf manager* if it only supports group members from its local domain.

A domain manager may only leave the tree after its last local member leaves the group. At this point, the domain manager begins executing the LeaveTree protocol shown in Figure 3. The algorithm for leaf managers is straightforward. However, the algorithm for internal managers is complicated by the fact that internal managers are a crucial link in the control tree, continuously servicing flow and error control messages from other managers, even when there are no local domain members left. In short, a departing internal node must discontinue service at some point and possibly coordinate children with the rest of the tree to allow seamless reintegration of children into the tree. Several alternative algorithms can be devised to determine when and how service will be cutoff and children reintegrated. The level of service provided by these algorithms could range from "unrecoverable interrupted service" to "temporarily interrupted service" to "uninterrupted service". Our current implementation provides "probably uninterrupted service" which means children of the departing manager continue to receive the feed while they reintegrate themselves into the tree. However, errors that arise during the brief reintegration time might not be correctable. We are still investigating alternatives to this approach.

After a departing manager has fulfilled all obligations to its children and parent, the departing manager instructs its children to find a new parent. The children then begin the process of joining the tree all over again. Although we investigated several other possible algorithms, we chose the above algorithm for its simplicity. Other, more static algorithms, such as requiring orphaned children to attach themselves to their grandparents, often result in poorly constructed control trees. Forcing the children to restart the join procedure ensures that children will select the closest possible connection point. Other more complex dynamic methods can be used to speed up the selection of the closest connection point but, in our experience, the performance of our simple algorithm has been acceptable.

DELIVERY MANAGEMENT

TMTP couples its packet transmission strategy with a unique tree-based error and flow control protocol to provide efficient and reliable dissemination. Conventional flow and error control algorithms employ a sender- or receiver-initiated approach. However, using the control tree, TMTP is able to combine the advantages of each approach while avoiding their disadvantages. Logically, TMTP's delivery management protocol can be partitioned into three components: data transmission, error handling, and flow control. The following sections address each of these aspects.

The Transmission Protocol

The basic transmission protocol is quite simple and is best described via a simple example. Assume a sender process S has established a dissemination group X and wants to multicast data to group X. S begins by multicasting data to the $(IP\_multicast\_addr, port\_no)$ representing group X. The multicast packets travel directly to all group members via standard IP multicast. In addition, all the domain managers in the control tree listen and receive the packets directly.

As in the sender-initiated approach, the root S expects to receive positive acknowledgments in order to reclaim buffer space and implement flow control. However, to avoid the *ack implosion* problem of the sender initiated approach, the sender does not receive acknowledgments directly from all the group members and, instead, receives ACKs only from its K immediate children. Once a domain manager receives a multicast packet from the sender, it can send an acknowledgment for the packet to its parent because the branch of the tree the manager represents has successfully received the packet (even though the individual members may not have received the packet). That is, a domain manager does not need to wait for ACKs from its children in order to send an ACK to the parent. In addition, each domain manager only periodically sends such ACKs to its parent. This feature substantially reduces ACK processing at the sender (and each domain manager).

Error Control

Before describing the details of TMTP's error control mechanism we must define an important concept called *limited scope multicast* messages. A limited scope multicast restricts the scope of a multicast message by setting the TTL value in the IP header to some small value which we call the multicast radius. The appropriate multicast radius to use is obtained from the expanding ring search that domain managers use to join the tree. Limited scope multicast messages prevent messages targeted to a particular region of the tree from propagating throughout the entire Internet.

TMTP employs error control techniques from both sender and receiver initiated approaches. Like the sender initiated approach, a TMTP traffic source (sender) requires periodic (unicast) positive acknowledgements and uses timeouts and (limited scope multicast) retransmissions to ensure reliable delivery to all its immediate children (domain managers). However, in addition to the sender, the domain managers in the control tree are also responsible for error control after they receive packets from the sender. Although the sender initially multicasts packets to the entire group, it is the domain manager's responsibility to ensure reliable delivery. Each domain manager also relies on periodic positive ACKs (from its immediate children), timeouts, and retransmissions to ensure reliable delivery to its children. When a retransmission timeout occurs, the sender (or domain manager) assumes the packet was lost and retransmits it using IP multicast (with a small TTL equal to the multicast radius for the local domain so that it only goes to its children).

In addition to the sender initiated approach, TMTP uses *restricted NACKs with NACK suppression* to respond quickly to packet losses. When a receiver notices a missing packet, the receiver generates a negative acknowledgment that is multicast to the parent and siblings using a restricted (small) TTL value. To avoid multiple receivers generating a NACK for the same packet, each receiver delays a random amount of time before transmitting its NACK. If the receiver hears a NACK from another sibling during the delay period, it suppresses its own NACK. This technique substantially reduces the load imposed by NACKs. When a domain manager receives a NACK, it immediately responds by multicasting the missing packet to the local domain using a limited scope multicast message.

Flow Control

TMTP achieves flow control by using a combination of rate-based and window-based techniques. The rate-based component of the protocol prohibits senders from transmitting data faster than some predefined maximum transmission rate. The maximum rate is set when the group is created and never changes. Despite its static nature, a fixed rate helps avoid congestion arising from bursty traffic and packet loss at rate-dependent receivers while still providing the necessary quality-of-service without excessive overhead.

TMTP's primary means of flow control consists of a window-based approach used for both dissemination from the sender and retransmission from domain managers. Within a window, senders transmit at a fixed rate.

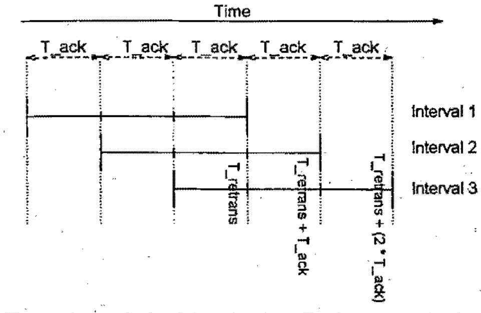
TMTP's window-based flow control differs slightly from conventional point-to-point window-based flow control. Note that retransmissions are very expensive because they are multicast. In addition, transient traffic conditions or congestion in one part of the network can put backpressure on the sender causing it to slow the data flow. To oversimplify, TMTP avoids both of these problems by partitioning the window and delaying retransmissions as long as possible. This increases the chance of a positive acknowledgement being received and it also allows domain managers to rectify transient behavior before it begins to cause backpressure.

TMTP uses two different timers to control the window size and the rate at which the window advances. $T_{retrans}$ defines a timeout period that begins when the first packet in a window is sent. Since the transfer rate is fixed, $T_{retrans}$ also defines the window size. A second timer, T_{ack}, defines the periodic interval at which each receiver is expected to unicast a positive ACK to its parent.

The sender specifies the value of T_{ack} based on the RTT to its farthest child. $T_{retrans}$ is chosen such that $T_{retrans} = n \times T_{ack}$, where n is an integer, $n \geq 2$. Both $T_{retrans}$ and T_{ack} are fixed at the beginning of transmission and do not change. A sender must allocate enough buffer space to hold packets that are transmitted over the $T_{retrans}$ period.

Figure 4 illustrates the windowing algorithm graphically. The sender starts a timer and begins transmitting data (at a fixed rate). Consider the packets transmitted during the first T_{ack} interval. Although the sender should see a positive ACK at time T_{ack}, the sender does not require one until time $T_{retrans}$. Instead, the sender continues to send packets during the second and third interval. After $T_{retrans}$ amount of time, the timer expires. At this point, the sender retransmits all unACK'd packets that were sent during the first T_{ack} interval. Retransmissions continue until all packets in the T_{ack} interval are acknowledged at which point the window is advanced by T_{ack}. On the receiving end, packets continue to arrive without being acknowledged until T_{ack} amount of time has expired[3].

[3] However, a receiver may generate a *restricted NACK* as soon as it detects a missing packet.



Three retransmission intervals where T_retrans = 3 * T_ack

At the end of the first interval, packets sent during the first T_ack period are retransmitted. At the end of the second interval, packets sent during the second T_ack period are retransmitted. At the end of the third interval, packets sent during the third T_ack period are retransmitted.

Figure 4: Different Stages in Sending Data

A domain manager must continue to hold packets in its buffer until all of its children have acknowledged them. If the children fail to acknowledge packets, the domain manager's window will not advance and its buffers will eventually fill up. As a result, the domain manager will drop and not acknowledge any new data from the sender, thereby causing backpressure to propagate up the tree which ultimately slows the flow of data.

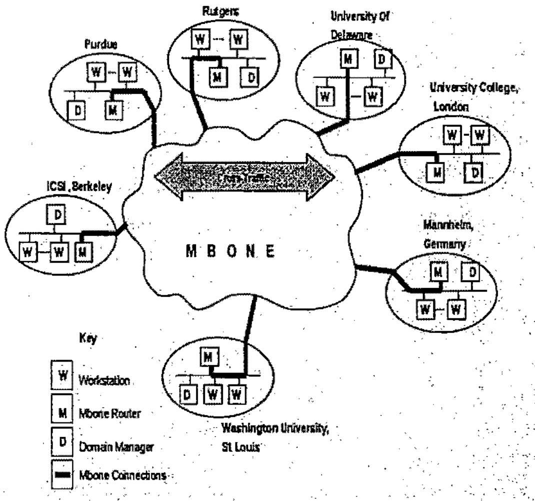
There are three reasons for using multiple T_{ack} intervals during a retransmission timeout interval ($T_{retrans}$). First, by requiring more than one positive ACK during the retransmission interval, TMTP protects itself from spurious retransmissions arising from lost ACKs. First, by requiring more than one positive ACK during the retransmission interval, TMTP protects itself from spurious retransmissions arising from lost ACKs. Second, a larger retransmission interval gives receivers sufficient time to recover missing packets using receiver-initiated recovery when only one (or a few) packets in a window are lost. This avoids unnecessary multicast retransmissions of a window full of data. Third, multiple T_{ack} intervals during the retransmission interval provide sufficient opportunity for a domain manager to recover from transient network load in its part of the subtree without unnecessarily applying backpressure to the sender.

We have chosen the value of the multiplying factor n to be 3 based on empirical evidence; the appropriate value depends on several factors including expected error rates, variance in RTT, and expected length of the intervals with transient, localized congestion. Further study is necessary to determine whether value of n should be chosen dynamically using an adaptive algorithm.

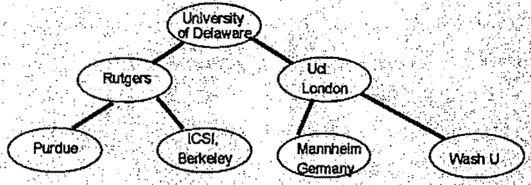
RESULTS

The Test Environment

Figure 5a illustrates the environment in which the experiments were run. Our tests involved seven geographically distinct Internet Mbone sites across the United States and Europe: Washington University in St. Louis, Purdue University, the International Computer Science Institute at Berkeley, Rutgers University, the University of Delaware, University College at London, and the University of Mannheim in Germany. All of our experiments were conducted using standard IP multicast across the Internet Mbone and thus experienced real Internet delays, congestion, and packet loss.



(5a) The Internet Mbone Used



(5b) The Control Tree

Figure 5: Figure 5a shows the test environment consisting of seven geographically distant sites connected by the Mbone. Figure 5b shows the corresponding control tree configuration used in the experiments.

As a point of comparison, we implemented a standard sender-initiated reliable multicast transport protocol both with and without window-base flow control (called *WIN_BASEP* and *BURST_BASEP* respectively). Under both protocols, the sender maintains state information for all receivers, expects positive ACKs from each receiver, and uses timeouts and global multicast retransmissions to recover from missing acknowledgments. The two BASEP protocols illustrate the performance bottlenecks related to processor load and end-to-end latency. All three protocols used the same packet size (1 Kbytes). TMTP and WIN_BASEP used a window size of 5. TMTP uses a transmission rate of 10 packets per second, while both BASEP protocols transmit packets as fast as possible (up to the window size in the case of WIN_BASEP). Both BASEP protocols set the retransmission timeout period to be twice the RTT to the farthest site (approx. 2 seconds in our tests). TMTP uses a retransmission period of $T_{retrans} = N \times T_{ack}$. T_{ack} is dynamically set based on the RTT to the farthest group member (approximately 1.1 seconds for our tests). After some preliminary evaluation of different setting for N, our empirical results indicated that $N = 3$ provides sufficient time for local domains to recover without delaying acks unnecessarily or consuming too much buffer space. Consequently, $T_{retrans}$ was approximately 3.3 seconds in our tests. The following sections describe the performance measures used and detail the actual experiments performed.

Performance Measures

To evaluate the performance of our protocol, we identified two important measures of performance: *end-to-end delay* and *processing load*. In addition, we monitored the total number of retransmissions to estimate the amount of network traffic generated by TMTP.

From the application's perspective, the primary concern is the delay in reliably delivering the entire data feed (e.g., video, audio, or file data) to the multiple recipients of the group. To measure the end-to-end delay, we required that each receiving application send back a single positive acknowledgment (a GOT_IT message) to the sending application when the entire data transmission was complete. The sending application then calculated the end-to-end delay as the time between the beginning of the transmission and the time at which the last group member's final GOT_IT message is received.

From the network's perspective, the primary concern is network load and scalability of the algorithm. If the protocol provides low end-to-end delay but consumes large amounts of network resources, the protocol will not scale well, congesting the Internet by consuming shared resources required by other Internet users. There are two aspects to network load: processing load and bandwidth consumption. To measure the processing load at the sender, receivers, and domain managers, we monitored the following processing activities:

- receiving and processing a selective positive acknowledgment
- receiving and processing a negative acknowledgment
- handling a timer event (such as a retransmission timeout)
- performing a retransmission

Because it is hard to measure the amount of processing time needed for each of the events listed above (and highly dependent on the operating system and architecture), we have chosen to simply count the total number of such events at the sender to estimate the processing load generated by a protocol.

The second important measure of network load is bandwidth consumption. The precise amount of bandwidth consumed by each protocol is much harder to quantify since we were unable to collect traces of traffic across the Mbone to determine the number of links traversed and the amount of bandwidth consumed over each link. However, our results indicate that TMTP generated far fewer retransmissions than the BASEP protocols, and most TMTP retransmissions are local to a particular domain. For example, under the BASEP protocols most timeouts/retransmissions occurred as a result of dropped ACKs. TMTP's hierarchy substantially reduced the number of lost ACKs, experiencing only 6 local retransmissions totaled across all domain managers (four occurring concurrently) as opposed to 9 global retransmission for BURST_BASEP (out of thirty 1K messages).

Experiments Performed

Each of our experiments measured the performance of a single dissemination group consisting of many processes evenly distributed across the seven sites pictured in Figure 5a. The total number of processes acting as receivers was varied between five and thirty processes. The five process case used only five domains while all other cases used seven domains. In each experiment, a sending process created a dissemination group, waited for the receiving processes to join the group and organize their domains into a control tree. Multiple tree configurations are possible depending on when, and in what order, domain mangers join the tree. However to ensure consistency across tests, we held the tree configuration constant across all tests (see Figure 5b). After all receivers joined the group, the sender disseminated a data file to the group, and then waited for the final GOT_IT message from all receivers. The values reported for each test are averaged over at least five runs taken during weekdays at roughly the same time so that the observed Internet traffic conditions remain similar across tests.

To gauge the scalability of the protocol, we monitored the changes in processing load at the senders, receivers, and managers. To measure the effective throughput, we measured the changes in end-to-end delay as perceived by the sender. Both processing load and end-to-end delay were recorded under a variety of workloads. In the first set of tests, the sender transmitted a 30 Kbyte file to a varying number of receivers. The dissemination was considered complete when all the receivers correctly receive the entire file. In the second set of tests, the number of processes was fixed at 30 and we incrementally increased the file size from 3K to 30 Kbytes. The end-to-end delay is measured as the time between the beginning of the file transfer and the time at which the last group member's final GOT_IT message is received.

To measure the processing load, we counted the total

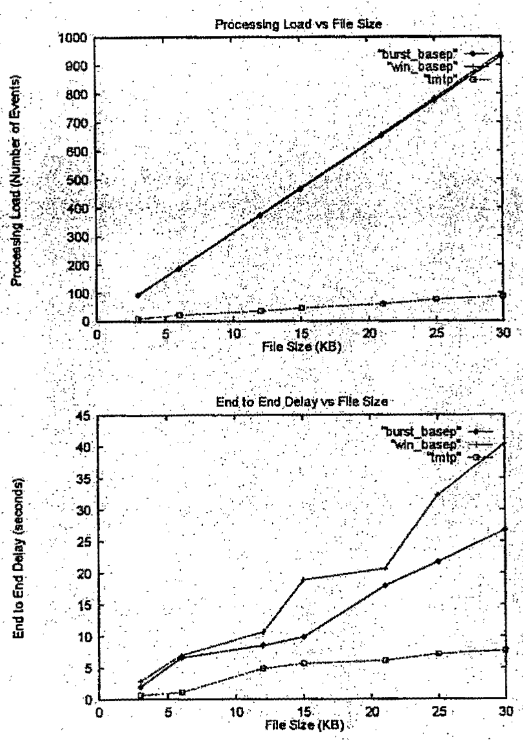


Figure 6: (a) Effect of the amount of data transmitted on the processing load. (b) Effect of the amount of data transmitted on the end-to-end delay. Figure b shows the time for the file transfer to complete at all the receivers. All measurements were taken with a dissemination group of size 30.

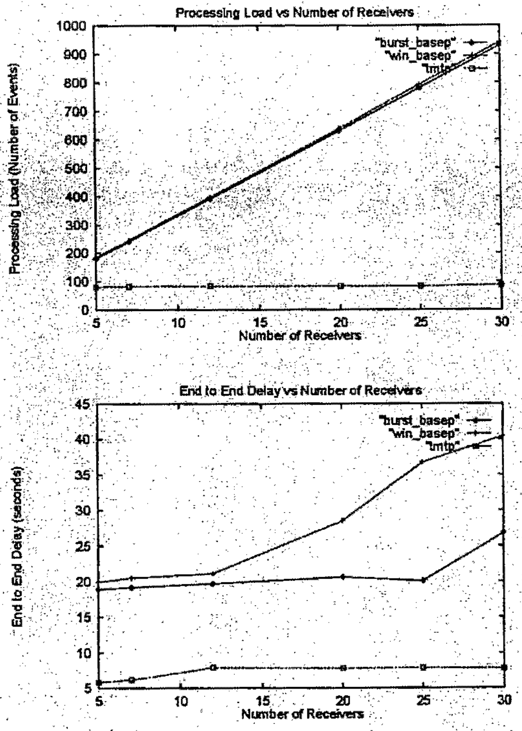


Figure 7: (a) Impact of group size (no. of receivers) on the processing load. (b) Impact of group size (no. of receivers) on the end-to-end delay. Figure b shows the time for the file transfer to complete at all the receivers. All measurements were taken for a dissemination of a 30 KB file.

number of events at the sender that contribute to the processing load. Similarly, we recorded the number of events at each domain manager. Figure 6 only shows the number of events processed at the sender. However, the balanced nature of our control tree meant the event processing load was spread equally among the sender and all domain managers. Consequently, the number of events processed at each domain manager is approximately the same as the number of events processed at the sender. Variations occurred based on the number of NACKs received.

Figures 6 and 7 show the results for each of the experiments performed. From these results we draw the following observations:

Impact of the Data Size

Figures 6a and 6b show how the file size affects the processing load and end-to-end delay. As the file size increases, the number of packets transmitted increases, thereby increasing the number of events (such as ACK/NACK processing or timer events) that affect the processing load at the sender (or a domain manager). Similarly, end-to-end delay is likely to increase due to time needed to deliver all the packets and due to increased probability of packet loss.

As the plots show, both the versions of the BASEP benchmark protocol show a significant increase in the processing load at the sender and the end-to-end delay. Note that the delay for WIN_BASEP (with flow control) is actually higher than BURST_BASEP (no flow control). This occurs because the WIN_BASEP sender expects acknowledgments from all its receivers before advancing the flow control window.

In the case of TMTP, the processing load shows only a small increase because the work is distributed among many nodes in the control tree. Consequently, the sender does not have to process acknowledgments or retransmission requests from all the receivers. TMTP's end-to-end delay is substantially lower than that of the BASEP protocols for all file sizes. Although all three protocols experience an increase in end-to-end delay resulting from larger data transmissions, packet losses, and retransmissions, TMTP's end-to-end delay rises at a significantly lower rate than that of the BASEP protocols. This occurs because error recovery in TMTP proceeds concurrently in different parts of the control tree rather than sequentially as in the BASEP cases.

Impact of the Group Size

Figures 7a and 7b show how the number of receivers (group size) affects the processing load and end-to-end delay.

Again, as the plots show, two versions of BASEP protocol show sharp increases in processing load with increase in number of receivers because the sender solely shoulders the responsibility for processing acknowledgments and retransmission requests (or timeouts) from each receiver. In the case of TMTP, the processing load at the sender (and each domain manger) is limited by the maximum number of immediate children in the control tree and, therefore, shows almost no increase as the number of receivers is increased. This results from the fact that the number of domains remains at seven for more than seven receivers. An increase in the number of domains participating in the dissemination group would cause a slight load increase on domain managers who adopt the new children.

Figure 7a shows that the end-to-end delay of both BASEP protocols is significantly higher than that of TMTP. The primary reason for this difference stems from TMTP's receiver-initiated capabilities that respond to and correct errors quickly. In contrast, the BASEP protocols will not correct an error until a retransmission timeout occurs.

In the case of TMTP end-to-end delays increases gradually because error recovery proceeds concurrently and independently in different parts of the control tree as explained earlier. Figure 7b shows that the end-to-end delay stabilizes to almost a constant value beyond a point. That is, to a small extent, an artifact of our tests in which we did not add any new domains to the control tree, but rather only added new processes to the existing tree. However, in other experiments involving varying number of domains, we have observed a similar trend of gradual increase in end-to-end delays with increasing number of receivers at additional domains.

RELATED WORK

A considerable amount of work has been reported in the literature regarding reliable multicast [13, 5, 3, 18, 12, 1, 4, 19, 15, 8, 11, 16]. Most of the earlier approaches achieve reliable delivery using a *sender-initiated* approach which is not suitable for large-scale, delay-sensitive, reliable dissemination.

Pingali and others[18] recently analyzed and compared both sender- and receiver-initiated approaches to demonstrate the limitations of the sender-initiated approach for large-scale dissemination. Our work is also motivated by similar observations, but combines the elements of both the approaches to achieve fast, local error recovery.

The reliable multicast protocol used in LBL's whiteboard tool (*wb*) [15, 8] and the log-based reliable multicast protocol [11] are two recent examples of the receiver-initiated approach for reliable delivery. Unlike TMTP, these protocols do not combine sender-initiated with receiver-initiated approaches and differ significantly in flow control mechanisms and buffering mechanisms. Our work is related to the *wb* work in that the *wb* protocol also uses a *NACKs with NACK suppression* mechanism. The *wb* protocol reduces state management overhead and achieves high degree of fault tolerance by relying solely on the receiver to recover from a packet loss. However, the protocol incurs the overhead of global (sometimes redundant) multicasts; a receiver multicasts a *repair request* to the entire group and one or more receivers in the group who have missing data (irrespective of their proximity to the complaining receiver) will multicast the missing packet(s) to the entire group even though the loss (or congestion) is restricted to a small region of the group topology. TMTP restricts the scope of multicast NACKs and retransmissions to the local domain to avoid generating redundant multicast transmissions over a wider region. Similar to TMTP, receivers using the wb protocol delay their NACKs to suppress duplicate NACKs in case another receiver multicasts a NACK. However, in the wb protocol, each receiver delays its NACK (and the response) by a random amount that depends on the RTT to the original sender. This can result in higher latency in recovering from packet losses. TMTP, on the other hand, uses localized recovery and, thus, the amount of random delay is bounded by the largest RTT between the local domain manager and one of the receivers in the domain. In addition, TMTP allows recovery from different errors to proceed concurrently in different domains to allow faster and efficient recovery.

Cheriton et. al.[8] have recently proposed a collection of strategies (called log-based receiver-reliable multicast or LRBM) for achieving large-scale, reliable multicast delivery. Some elements of LRBM are similar to TMTP's mechanisms to some extent. LRBM uses a hierarchy of logging servers with a primary log server responsible for sending positive acknowledgments to the multicast source. The primary log server stores the packets as long as an application desires and the receivers must recover from errors by contacting a logging server. A secondary server at each site may log received packets and satisfy local retransmission requests to reduce load on the primary server. Deployment of LRBM in the Internet is necessary to evaluate its performance in achieving reliable delivery in a wide area network environment.

Recently Paul et. al. [16] have proposed and are examining three multicast alternatives with features similar to those of TMTP. In contrast to these protocols, TMTP uses a multi-level hierarchical control tree and a dynamic group management protocol, as opposed to a static two-level hierarchy, to evenly distribute the protocol processing load and allow finer grained independent and concurrent error recovery. TMTP targets a best-effort multicast system such as IP multicast rather than an ATM-like network with allocated resources. TMTP imposes no additional load on network-level routers and requires no modification to the network-level routers, but yet incorporates both local retransmissions and combined acknowledgments. Furthermore, TMTP employs receiver-initiated recovery techniques (*restricted negative acknowledgments with nack suppression* combined with periodic positive acknowledgments) and a unique flow control mechanism that can provide quick recovery from transient congestion and lost acknowledgments.

CONCLUSION

Based on our experimental results, we believe that TMTP can scale well to provide reliable delivery on a large scale without sacrificing end-to-end latency. Under TMTP, the network processing load increases very gradually, indicating that the protocol will scale well as the number of receivers increases. Moreover, TMTP provides significantly better application-level throughput because of the concurrency resulting from local retransmissions as shown by the end-to-end measurements.

References

[1] Ken Birman and Thomas Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):47–76, Feb 1987.

[2] S. Casner and S. Deering. First IETF Internet Audiocast. *ACM Computer Communication Review*, 22(3):92–97, July 1992.

[3] J. Chang and N. Maxemchuck. Reliable Broadcast Protocols. *ACM Transactions on Computer Systems*, 2(3):251–273, August 1984.

[4] David. R. Cheriton and W. Zwaenepoel. Distributed process groups in the V kernel. *ACM Transactions on Computer Systems*, 3(2):77–107, May 1985.

[5] J. Crowcroft and K. Paliwoda. A Multicast Transport Protocol. In *Proceedings of ACM SIGCOMM '88*, pages 247–256, August 1988.

[6] Stephen E. Deering and David R. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.

[7] Prasun Dewan. A Guide to Suite: Version 1.0. Technical Report SERC-TR-60-P, Software Engineering Research Center, Purdue University, West Lafayette, IN, February 1990.

[8] S. Floyd, V. Jacobsen, S. McCanne, C-G Liu, and L. Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. In *sigcomm95*, 1995. to appear.

[9] I. Gopal and J. Jaffe. Point-to-multipoint Communication over Broadcast Links. *IEEE Transactions on Communications*, 32, September 1984.

[10] James Griffioen and Rajendra Yavatkar. Clique: A Toolkit for Group Communication using IP Multicast. In *Proceedings of the Workshop on Services in Distributed and Networked Environments*, June 1994.

[11] H.W. Holbrook, S.K. Singhal, and D.R. Cheriton. Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation. In *sigcomm95*, 1995. to appear.

[12] Van Jacobson. *SD: Session Directory*. Lawrence Berkeley Laboratory, March 1993.

[13] M Frans Kaashoek, A.S. Tanenbaum, S.F. Hummel, and H.E. Bal. An Efficient Reliable Broadcast Protocol. *ACM Operating Systems Review*, 23(4), October 1989.

[14] Amit Mathur and Atul Prakash. Protocols for integrated audio and shared windows in collaborative systems. In *Proceedings of ACM Multimedia '94*, October 1994.

[15] Steven McCanne. A Distributed Whiteboard for Network Conferencing. Technical report, Real Time Systems Group, Lawrence Berkeley Laboratory, Berkeley, CA, September 1992. unpublished report.

[16] S. Paul, K. Sabnani, and D. Kristol. Multicast Transport Protocols for High Speed Networks. In *IEEE Int. Conf. on Network Protocols*, 1994 Oct.

[17] L. Peterson, N. Buchholz, and R.D. Schlichting. Preserving and using context information in interprocess communication. *ACM Transactions on Computer Systems*, 7(3):217–246, August 1989.

[18] Sridhar Pingali, Don Towsley, and James F. Kurose. A comparision of sender-initiated and receiver-initiated reliable multicast protocols. In *Proceddings of ACM SIGMETRICS '94*, volume 14, pages 221–230, 1994.

[19] S. Ramakrishnan and B.N. Jain. A Negative Acknowledgement Protocol with Periodic Polling Protocol for Multicast over Lans. In *Proccedings of IEEE INFOCOMM '87*, pages 502–511, March-April 1987.

GRAPH THEORY WITH APPLICATIONS

J. A. Bondy and U. S. R. Murty

Department of Combinatorics and Optimization,
University of Waterloo,
Ontario, Canada

AMERICAN ELSEVIER PUBLISHING CO., INC.

First published in Great Britain 1976 by
The Macmillan Press Ltd

Reprinted 1977

First published in the U.S.A. 1976 by
AMERICAN ELSEVIER PUBLISHING CO., INC.
52 Vanderbilt Avenue
New York, N.Y. 10017

ISBN 0-444-19451-7
LCCCN 75-29826

Printed in Great Britain

To our parents

Contents

1 Graphs and Subgraphs

1.1 GRAPHS AND SIMPLE GRAPHS

Many real-world situations can conveniently be described by means of a diagram consisting of a set of points together with lines joining certain pairs of these points. For example, the points could represent people, with lines joining pairs of friends; or the points might be communication centres, with lines representing communication links. Notice that in such diagrams one is mainly interested in whether or not two given points are joined by a line; the manner in which they are joined is immaterial. A mathematical abstraction of situations of this type gives rise to the concept of a graph.

A *graph* G is an ordered triple $(V(G), E(G), \psi_G)$ consisting of a nonempty set $V(G)$ of *vertices*, a set $E(G)$, disjoint from $V(G)$, of *edges*, and an *incidence function* ψ_G that associates with each edge of G an unordered pair of (not necessarily distinct) vertices of G. If e is an edge and u and v are vertices such that $\psi_G(e) = uv$, then e is said to *join* u and v; the vertices u and v are called the *ends* of e.

Two examples of graphs should serve to clarify the definition.

Example 1

$$G = (V(G), E(G), \psi_G)$$

where

$$V(G) = \{v_1, v_2, v_3, v_4, v_5\}$$
$$E(G) = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$$

and ψ_G is defined by

$$\psi_G(e_1) = v_1v_2,\ \psi_G(e_2) = v_2v_3,\ \psi_G(e_3) = v_3v_3,\ \psi_G(e_4) = v_3v_4$$
$$\psi_G(e_5) = v_2v_4,\ \psi_G(e_6) = v_4v_5,\ \psi_G(e_7) = v_2v_5,\ \psi_G(e_8) = v_2v_5$$

Example 2

$$H = (V(H), E(H), \psi_H)$$

where

$$V(H) = \{u, v, w, x, y\}$$
$$E(H) = \{a, b, c, d, e, f, g, h\}$$

and ψ_H is defined by

$$\psi_H(a) = uv, \quad \psi_H(b) = uu, \quad \psi_H(c) = vw, \quad \psi_H(d) = wx$$
$$\psi_H(e) = vx, \quad \psi_H(f) = wx, \quad \psi_H(g) = ux, \quad \psi_H(h) = xy$$

1 Graphs and Subgraphs

1.1 GRAPHS AND SIMPLE GRAPHS

Many real-world situations can conveniently be described by means of a diagram consisting of a set of points together with lines joining certain pairs of these points. For example, the points could represent people, with lines joining pairs of friends; or the points might be communication centres, with lines representing communication links. Notice that in such diagrams one is mainly interested in whether or not two given points are joined by a line; the manner in which they are joined is immaterial. A mathematical abstraction of situations of this type gives rise to the concept of a graph.

A *graph* G is an ordered triple $(V(G), E(G), \psi_G)$ consisting of a nonempty set $V(G)$ of *vertices*, a set $E(G)$, disjoint from $V(G)$, of *edges*, and an *incidence function* ψ_G that associates with each edge of G an unordered pair of (not necessarily distinct) vertices of G. If e is an edge and u and v are vertices such that $\psi_G(e) = uv$, then e is said to *join* u and v; the vertices u and v are called the *ends* of e.

Two examples of graphs should serve to clarify the definition.

Example 1

$$G = (V(G), E(G), \psi_G)$$

where

$$V(G) = \{v_1, v_2, v_3, v_4, v_5\}$$
$$E(G) = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$$

and ψ_G is defined by

$$\psi_G(e_1) = v_1v_2,\ \psi_G(e_2) = v_2v_3,\ \psi_G(e_3) = v_3v_3,\ \psi_G(e_4) = v_3v_4$$
$$\psi_G(e_5) = v_2v_4,\ \psi_G(e_6) = v_4v_5,\ \psi_G(e_7) = v_2v_5,\ \psi_G(e_8) = v_2v_5$$

Example 2

$$H = (V(H), E(H), \psi_H)$$

where

$$V(H) = \{u, v, w, x, y\}$$
$$E(H) = \{a, b, c, d, e, f, g, h\}$$

and ψ_H is defined by

$$\psi_H(a) = uv, \quad \psi_H(b) = uu, \quad \psi_H(c) = vw, \quad \psi_H(d) = wx$$
$$\psi_H(e) = vx, \quad \psi_H(f) = wx, \quad \psi_H(g) = ux, \quad \psi_H(h) = xy$$

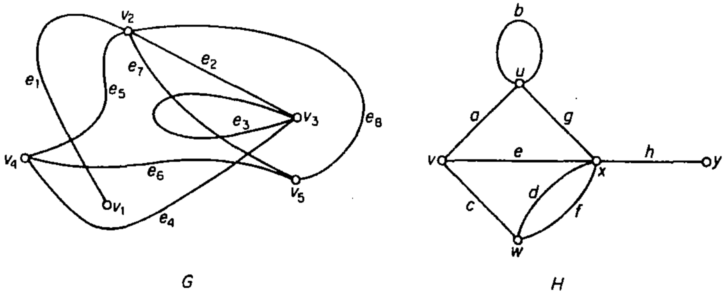


Figure 1.1. Diagrams of graphs G and H

Graphs are so named because they can be represented graphically, and it is this graphical representation which helps us understand many of their properties. Each vertex is indicated by a point, and each edge by a line joining the points which represent its ends.† Diagrams of G and H are shown in figure 1.1. (For clarity, vertices are depicted here as small circles.)

There is no unique way of drawing a graph; the relative positions of points representing vertices and lines representing edges have no significance. Another diagram of G, for example, is given in figure 1.2. A diagram of a graph merely depicts the incidence relation holding between its vertices and edges. We shall, however, often draw a diagram of a graph and refer to it as the graph itself; in the same spirit, we shall call its points 'vertices' and its lines 'edges'.

Note that two edges in a diagram of a graph may intersect at a point that is not a vertex (for example e_1 and e_6 of graph G in figure 1.1). Those graphs that have a diagram whose edges intersect only at their ends are called *planar*, since such graphs can be represented in the plane in a simple manner. The graph of figure 1.3*a* is planar, even though this is not immediately clear from the particular representation shown (see exercise 1.1.2). The graph of figure 1.3*b*, on the other hand, is nonplanar. (This will be proved in chapter 9.)

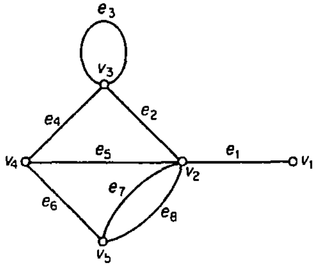
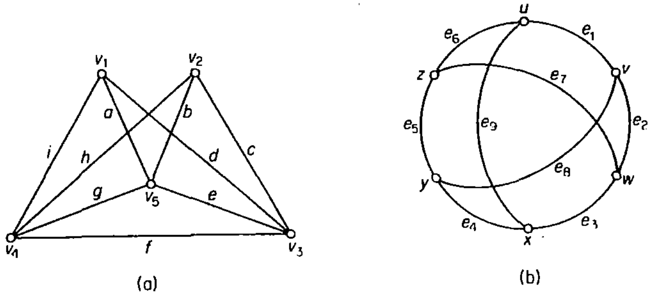


Figure 1.2. Another diagram of G

† In such a drawing it is understood that no line intersects itself or passes through a point representing a vertex which is not an end of the corresponding edge—this is clearly always possible.

Most of the definitions and concepts in graph theory are suggested by the graphical representation. The ends of an edge are said to be *incident* with the edge, and vice versa. Two vertices which are incident with a common edge are *adjacent*, as are two edges which are incident with a common vertex. An edge with identical ends is called a *loop*, and an edge with distinct ends a *link*. For example, the edge e_3 of G (figure 1.2) is a loop; all other edges of G are links.



(a)

(b)

Figure 1.3. Planar and nonplanar graphs

A graph is *finite* if both its vertex set and edge set are finite. In this book we study only finite graphs, and so the term 'graph' always means 'finite graph'. We call a graph with just one vertex *trivial* and all other graphs *nontrivial*.

A graph is *simple* if it has no loops and no two of its links join the same pair of vertices. The graphs of figure 1.1 are not simple, whereas the graphs of figure 1.3 are. Much of graph theory is concerned with the study of simple graphs.

We use the symbols $\nu(G)$ and $\varepsilon(G)$ to denote the numbers of vertices and edges in graph G. Throughout the book the letter G denotes a graph. Moreover, when just one graph is under discussion, we usually denote this graph by G. We then omit the letter G from graph-theoretic symbols and write, for instance, V, E, ν and ε instead of $V(G)$, $E(G)$, $\nu(G)$ and $\varepsilon(G)$.

Exercises

1.1.1 List five situations from everyday life in which graphs arise naturally.

1.1.2 Draw a different diagram of the graph of figure 1.3*a* to show that it is indeed planar.

1.1.3 Show that if G is simple, then $\varepsilon \leq \binom{\nu}{2}$.

1.2 GRAPH ISOMORPHISM

Two graphs G and H are *identical* (written $G = H$) if $V(G) = V(H)$, $E(G) = E(H)$, and $\psi_G = \psi_H$. If two graphs are identical then they can clearly be represented by identical diagrams. However, it is also possible for graphs that are not identical to have essentially the same diagram. For example, the diagrams of G in figure 1.2 and H in figure 1.1 look exactly the same, with the exception that their vertices and edges have different labels. The graphs G and H are not identical, but isomorphic. In general, two graphs G and H are said to be *isomorphic* (written $G \cong H$) if there are bijections $\theta: V(G) \to V(H)$ and $\phi: E(G) \to E(H)$ such that $\psi_G(e) = uv$ if and only if $\psi_H(\phi(e)) = \theta(u)\theta(v)$; such a pair (θ, ϕ) of mappings is called an *isomorphism* between G and H.

To show that two graphs are isomorphic, one must indicate an isomorphism between them. The pair of mappings (θ, ϕ) defined by

$$\theta(v_1) = y, \quad \theta(v_2) = x, \quad \theta(v_3) = u, \quad \theta(v_4) = v, \quad \theta(v_5) = w$$

and

$$\phi(e_1) = h, \quad \phi(e_2) = g, \quad \phi(e_3) = b, \quad \phi(e_4) = a$$
$$\phi(e_5) = e, \quad \phi(e_6) = c, \quad \phi(e_7) = d, \quad \phi(e_8) = f$$

is an isomorphism between the graphs G and H of examples 1 and 2; G and H clearly have the same structure, and differ only in the names of vertices and edges. Since it is in structural properties that we shall primarily be interested, we shall often omit labels when drawing graphs; an unlabelled graph can be thought of as a representative of an equivalence class of isomorphic graphs. We assign labels to vertices and edges in a graph mainly for the purpose of referring to them. For instance, when dealing with simple graphs, it is often convenient to refer to the edge with ends u and v as 'the edge uv'. (This convention results in no ambiguity since, in a simple graph, at most one edge joins any pair of vertices.)

We conclude this section by introducing some special classes of graphs. A simple graph in which each pair of distinct vertices is joined by an edge is called a *complete graph*. Up to isomorphism, there is just one complete graph on n vertices; it is denoted by K_n. A drawing of K_5 is shown in figure 1.4*a*. An *empty graph*, on the other hand, is one with no edges. A *bipartite graph* is one whose vertex set can be partitioned into two subsets X and Y, so that each edge has one end in X and one end in Y; such a partition (X, Y) is called a *bipartition* of the graph. A *complete bipartite graph* is a simple bipartite graph with bipartition (X, Y) in which each vertex of X is joined to each vertex of Y; if $|X| = m$ and $|Y| = n$, such a graph is denoted by $K_{m,n}$. The graph defined by the vertices and edges of a cube (figure 1.4*b*) is bipartite; the graph in figure 1.4*c* is the complete bipartite graph $K_{3,3}$.

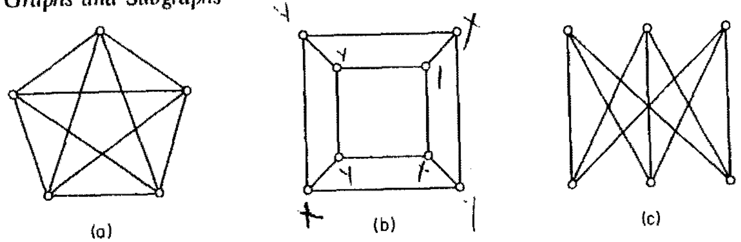


Figure 1.4. (*a*) K_5; (*b*) the cube; (*c*) $K_{3,3}$

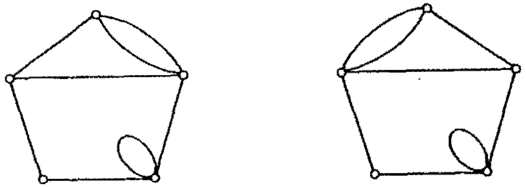
There are many other graphs whose structures are of special interest. Appendix III includes a selection of such graphs.

Exercises

1.2.1 Find an isomorphism between the graphs G and H of examples 1 and 2 different from the one given.

1.2.2 (*a*) Show that if $G \cong H$, then $\nu(G) = \nu(H)$ and $\varepsilon(G) = \varepsilon(H)$.
(*b*) Give an example to show that the converse is false.

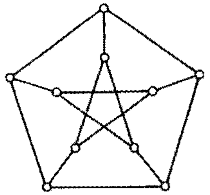
1.2.3 Show that the following graphs are not isomorphic:

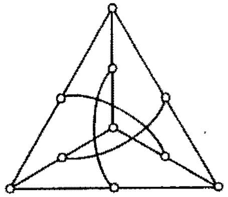


1.2.4 Show that there are eleven nonisomorphic simple graphs on four vertices.

1.2.5 Show that two simple graphs G and H are isomorphic if and only if there is a bijection $\theta: V(G) \to V(H)$ such that $uv \in E(G)$ if and only if $\theta(u)\theta(v) \in E(H)$.

1.2.6 Show that the following graphs are isomorphic:





1.2.7 Let G be simple. Show that $\varepsilon = \binom{\nu}{2}$ if and only if G is complete.

1.2.8 Show that

(*a*) $\varepsilon(K_{m,n}) = mn$;

(*b*) if G is simple and bipartite, then $\varepsilon \le \nu^2/4$.

1.2.9 A *k-partite graph* is one whose vertex set can be partitioned into k subsets so that no edge has both ends in any one subset; a *complete k-partite graph* is one that is simple and in which each vertex is joined to every vertex that is not in the same subset. The complete m-partite graph on n vertices in which each part has either $[n/m]$ or $\{n/m\}$ vertices is denoted by $T_{m,n}$. Show that

(*a*) $\varepsilon(T_{m,n}) = \binom{n-k}{2} + (m-1)\binom{k+1}{2}$, where $k = [n/m]$;

(*b*)* if G is a complete m-partite graph on n vertices, then $\varepsilon(G) \le \varepsilon(T_{m,n})$, with equality only if $G \cong T_{m,n}$.

1.2.10 The *k-cube* is the graph whose vertices are the ordered k-tuples of 0's and 1's, two vertices being joined if and only if they differ in exactly one coordinate. (The graph shown in figure 1.4*b* is just the 3-cube.) Show that the k-cube has 2^k vertices, $k2^{k-1}$ edges and is bipartite.

1.2.11 (*a*) The *complement* G^c of a simple graph G is the simple graph with vertex set V, two vertices being adjacent in G^c if and only if they are not adjacent in G. Describe the graphs K_n^c and $K_{m,n}^c$.

(*b*) A simple graph G is *self-complementary* if $G \cong G^c$. Show that if G is self-complementary, then $\nu \equiv 0, 1 \pmod 4$.

1.2.12 An *automorphism* of a graph is an isomorphism of the graph onto itself.

(*a*) Show, using exercise 1.2.5, that an automorphism of a simple graph G can be regarded as a permutation on V which preserves adjacency, and that the set of such permutations form a group $\Gamma(G)$ (the *automorphism group* of G) under the usual operation of composition.

(*b*) Find $\Gamma(K_n)$ and $\Gamma(K_{m,n})$.

(*c*) Find a nontrivial simple graph whose automorphism group is the identity.

(*d*) Show that for any simple graph G, $\Gamma(G) = \Gamma(G^c)$.

(*e*) Consider the permutation group A with elements (1)(2)(3), (1, 2, 3) and (1, 3, 2). Show that there is no simple graph G with vertex set {1, 2, 3} such that $\Gamma(G) = A$.

(*f*) Find a simple graph G such that $\Gamma(G) \cong A$. (Frucht, 1939 has shown that every abstract group is isomorphic to the automorphism group of some graph.)

1.2.13 A simple graph G is *vertex-transitive* if, for any two vertices u and v, there is an element g in $\Gamma(G)$ such that $g(u) = g(v)$; G is *edge-transitive* if, for any two edges u_1v_1 and u_2v_2, there is an element h in $\Gamma(G)$ such that $h(\{u_1, v_1\}) = \{u_2, v_2\}$. Find

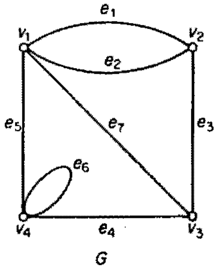
(*a*) a graph which is vertex-transitive but not edge-transitive;

(*b*) a graph which is edge-transitive but not vertex-transitive.

1.3 THE INCIDENCE AND ADJACENCY MATRICES

To any graph G there corresponds a $\nu \times \varepsilon$ matrix called the incidence matrix of G. Let us denote the vertices of G by $v_1, v_2, \ldots, v_\nu$ and the edges by $e_1, e_2, \ldots, e_\varepsilon$. Then the *incidence matrix* of G is the matrix $\mathbf{M}(G) = [m_{ij}]$, where m_{ij} is the number of times (0, 1 or 2) that v_i and e_j are incident. The incidence matrix of a graph is just a different way of specifying the graph.

Another matrix associated with G is the *adjacency matrix*; this is the $\nu \times \nu$ matrix $\mathbf{A}(G) = [a_{ij}]$, in which a_{ij} is the number of edges joining v_i and v_j. A graph, its incidence matrix, and its adjacency matrix are shown in figure 1.5.



$$\begin{array}{c|ccccccc} & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \\ \hline v_1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ v_2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ v_3 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ v_4 & 0 & 0 & 0 & 1 & 1 & 2 & 0 \end{array}$$

$\mathbf{M}(G)$

$$\begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ \hline v_1 & 0 & 2 & 1 & 1 \\ v_2 & 2 & 0 & 1 & 0 \\ v_3 & 1 & 1 & 0 & 1 \\ v_4 & 1 & 0 & 1 & 1 \end{array}$$

$\mathbf{A}(G)$

Figure 1.5

The adjacency matrix of a graph is generally considerably smaller than its incidence matrix, and it is in this form that graphs are commonly stored in computers.

Exercises

1.3.1 Let **M** be the incidence matrix and **A** the adjacency matrix of a graph G.

(*a*) Show that every column sum of **M** is 2.

(*b*) What are the column sums of **A**?

1.3.2 Let G be bipartite. Show that the vertices of G can be enumerated so that the adjacency matrix of G has the form

$$\left[\begin{array}{c|c} \mathbf{0} & \mathbf{A}_{12} \\ \hline \mathbf{A}_{21} & \mathbf{0} \end{array}\right]$$

where $\mathbf{A}_{21}$ is the transpose of $\mathbf{A}_{12}$.

1.3.3* Show that if G is simple and the eigenvalues of **A** are distinct, then the automorphism group of G is abelian

1.4 SUBGRAPHS

A graph H is a *subgraph* of G (written $H \subseteq G$) if $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$, and ψ_H is the restriction of ψ_G to $E(H)$. When $H \subseteq G$ but $H \neq G$, we write $H \subset G$ and call H a *proper subgraph* of G. If H is a subgraph of G, G is a *supergraph* of H. A *spanning subgraph* (or *spanning supergraph*) of G is a subgraph (or supergraph) H with $V(H) = V(G)$.

By deleting from G all loops and, for every pair of adjacent vertices, all but one link joining them, we obtain a simple spanning subgraph of G, called the *underlying simple graph* of G. Figure 1.6 shows a graph and its underlying simple graph.

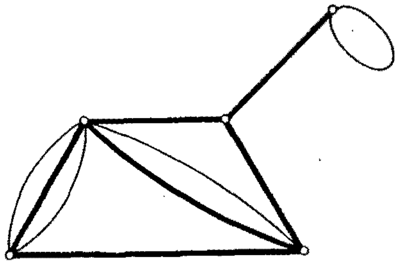
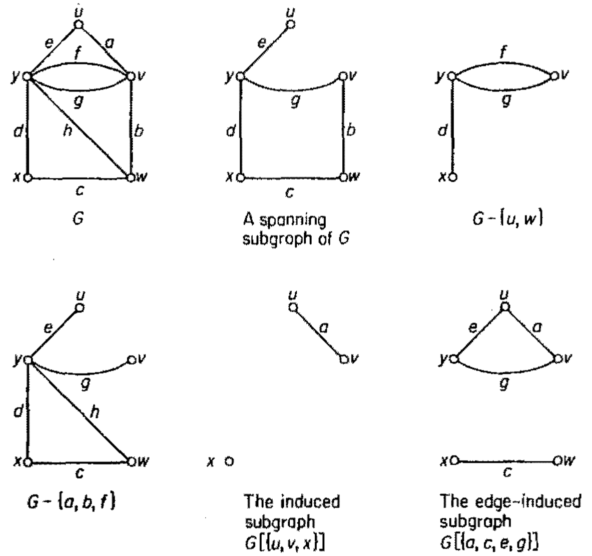


Figure 1.6. A graph and its underlying simple graph



G

A spanning subgraph of G

G - {u, w}

G - {a, b, f}

The induced subgraph G[{u, v, x}]

The edge-induced subgraph G[{a, c, e, g}]

Figure 1.7

Suppose that V' is a nonempty subset of V. The subgraph of G whose vertex set is V' and whose edge set is the set of those edges of G that have both ends in V' is called the subgraph of G *induced* by V' and is denoted by $G[V']$; we say that $G[V']$ is an *induced subgraph* of G. The induced subgraph $G[V \backslash V']$ is denoted by $G - V'$; it is the subgraph obtained from G by deleting the vertices in V' together with their incident edges. If $V' = \{v\}$ we write $G - v$ for $G - \{v\}$.

Now suppose that E' is a nonempty subset of E. The subgraph of G whose vertex set is the set of ends of edges in E' and whose edge set is E' is called the subgraph of G *induced* by E' and is denoted by $G[E']$; $G[E']$ is an *edge-induced subgraph* of G. The spanning subgraph of G with edge set $E \backslash E'$ is written simply as $G - E'$; it is the subgraph obtained from G by deleting the edges in E'. Similarly, the graph obtained from G by adding a set of edges E' is denoted by $G + E'$. If $E' = \{e\}$ we write $G - e$ and $G + e$ instead of $G - \{e\}$ and $G + \{e\}$.

Subgraphs of these various types are depicted in figure 1.7.

Let G_1 and G_2 be subgraphs of G. We say that G_1 and G_2 are *disjoint* if they have no vertex in common, and *edge-disjoint* if they have no edge in common. The *union* $G_1 \cup G_2$ of G_1 and G_2 is the subgraph with vertex set

$V(G_1) \cup V(G_2)$ and edge set $E(G_1) \cup E(G_2)$; if G_1 and G_2 are disjoint, we sometimes denote their union by $G_1 + G_2$. The *intersection* $G_1 \cap G_2$ of G_1 and G_2 is defined similarly, but in this case G_1 and G_2 must have at least one vertex in common.

Exercises

1.4.1 Show that every simple graph on n vertices is isomorphic to a subgraph of K_n.

1.4.2 Show that

(*a*) every induced subgraph of a complete graph is complete;

(*b*) every subgraph of a bipartite graph is bipartite.

1.4.3 Describe how $\mathbf{M}(G-E')$ and $\mathbf{M}(G-V')$ can be obtained from $\mathbf{M}(G)$, and how $\mathbf{A}(G-V')$ can be obtained from $\mathbf{A}(G)$.

1.4.4 Find a bipartite graph that is not isomorphic to a subgraph of any k-cube.

1.4.5* Let G be simple and let n be an integer with $1 < n < \nu - 1$. Show that if $\nu \geq 4$ and all induced subgraphs of G on n vertices have the same number of edges, then either $G \cong K_\nu$ or $G \cong K_\nu^c$.

1.5 VERTEX DEGREES

The *degree* $d_G(v)$ of a vertex v in G is the number of edges of G incident with v, each loop counting as two edges. We denote by $\delta(G)$ and $\Delta(G)$ the minimum and maximum degrees, respectively, of vertices of G.

Theorem 1.1

$$\sum_{v \in V} d(v) = 2\varepsilon$$

Proof Consider the incidence matrix $\mathbf{M}$. The sum of the entries in the row corresponding to vertex v is precisely $d(v)$, and therefore $\sum_{v \in V} d(v)$ is just the sum of all entries in $\mathbf{M}$. But this sum is also 2ε, since (exercise 1.3.1*a*) each of the ε column sums of $\mathbf{M}$ is 2 □

Corollary 1.1 In any graph, the number of vertices of odd degree is even.

Proof Let V_1 and V_2 be the sets of vertices of odd and even degree in G, respectively. Then

$$\sum_{v \in V_1} d(v) + \sum_{v \in V_2} d(v) = \sum_{v \in V} d(v)$$

is even, by theorem 1.1. Since $\sum_{v \in V_2} d(v)$ is also even, it follows that $\sum_{v \in V_1} d(v)$ is even. Thus $|V_1|$ is even □

all nodes with degree k

A graph G is *k-regular* if $d(v) = k$ for all $v \in V$; a *regular graph* is one that is k-regular for some k. Complete graphs and complete bipartite graphs $K_{n,n}$ are regular; so, also, are the k-cubes.

Exercises

1.5.1 Show that $\delta \leq 2\varepsilon/\nu \leq \Delta$.

1.5.2 Show that if G is simple, the entries on the diagonals of both $\mathbf{MM}'$ and $\mathbf{A}^2$ are the degrees of the vertices of G.

1.5.3 Show that if a k-regular bipartite graph with $k > 0$ has bipartition (X, Y), then $|X| = |Y|$.

1.5.4 Show that, in any group of two or more people, there are always two with exactly the same number of friends inside the group.

1.5.5 If G has vertices $v_1, v_2, \ldots, v_\nu$, the sequence $(d(v_1), d(v_2), \ldots, d(v_\nu))$ is called a *degree sequence* of G. Show that a sequence $(d_1, d_2, \ldots, d_n)$ of non-negative integers is a degree sequence of some graph if and only if $\sum_{i=1}^{n} d_i$ is even.

1.5.6 A sequence $\mathbf{d} = (d_1, d_2, \ldots, d_n)$ is *graphic* if there is a simple graph with degree sequence $\mathbf{d}$. Show that

(*a*) the sequences (7, 6, 5, 4, 3, 3, 2) and (6, 6, 5, 4, 3, 3, 1) are not graphic;

(*b*) if $\mathbf{d}$ is graphic and $d_1 \geq d_2 \geq \ldots \geq d_n$, then $\sum_{i=1}^{n} d_i$ is even and

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min\{k, d_i\} \quad \text{for} \quad 1 \leq k \leq n$$

(Erdös and Gallai, 1960 have shown that this necessary condition is also sufficient for $\mathbf{d}$ to be graphic.)

1.5.7 Let $\mathbf{d} = (d_1, d_2, \ldots, d_n)$ be a nonincreasing sequence of non-negative integers, and denote the sequence $(d_2 - 1, d_3 - 1, \ldots, d_{d_1+1} - 1, d_{d_1+2}, \ldots, d_n)$ by $\mathbf{d}'$.

(*a*)* Show that $\mathbf{d}$ is graphic if and only if $\mathbf{d}'$ is graphic.

(*b*) Using (*a*), describe an algorithm for constructing a simple graph with degree sequence $\mathbf{d}$, if such a graph exists.

(V. Havel, S. Hakimi)

1.5.8* Show that a loopless graph G contains a bipartite spanning subgraph H such that $d_H(v) \geq \frac{1}{2} d_G(v)$ for all $v \in V$.

1.5.9* Let $S = \{x_1, x_2, \ldots, x_n\}$ be a set of points in the plane such that the distance between any two points is at least one. Show that there are at most $3n$ pairs of points at distance exactly one.

1.5.10 The *edge graph* of a graph G is the graph with vertex set $E(G)$ in which two vertices are joined if and only if they are adjacent edges in

G. Show that, if G is simple

(*a*) the edge graph of G has $\varepsilon(G)$ vertices and $\sum_{v\in V(G)}\binom{d_G(v)}{2}$ edges;

(*b*) the edge graph of K_5 is isomorphic to the complement of the graph featured in exercise 1.2.6.

1.6 PATHS AND CONNECTION

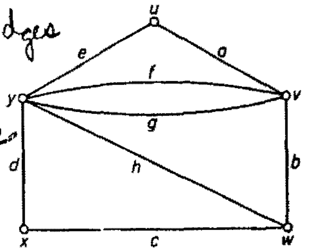
A *walk* in G is a finite non-null sequence $W = v_0e_1v_1e_2v_2 \dots e_kv_k$, whose terms are alternately vertices and edges, such that, for $1 \le i \le k$, the ends of e_i are v_{i-1} and v_i. We say that W is a walk *from* v_0 *to* v_k, or a (v_0, v_k)-*walk*. The vertices v_0 and v_k are called the *origin* and *terminus* of W, respectively, and $v_1, v_2, \dots, v_{k-1}$ its *internal vertices*. The integer k is the *length* of W.

If $W = v_0e_1v_1 \dots e_kv_k$ and $W' = v_ke_{k+1}v_{k+1} \dots e_lv_l$ are walks, the walk $v_ke_kv_{k-1} \dots e_1v_0$, obtained by reversing W, is denoted by W^{-1} and the walk $v_0e_1v_1 \dots e_lv_l$, obtained by concatenating W and W' at v_k, is denoted by WW'. A *section* of a walk $W = v_0e_1v_1 \dots e_kv_k$ is a walk that is a subsequence $v_ie_{i+1}v_{i+1} \dots e_jv_j$ of consecutive terms of W; we refer to this subsequence as the (v_i, v_j)-*section* of W.

In a simple graph, a walk $v_0e_1v_1 \dots e_kv_k$ is determined by the sequence $v_0v_1 \dots v_k$ of its vertices; hence a walk in a simple graph can be specified simply by its vertex sequence. Moreover, even in graphs that are not simple, we shall sometimes refer to a sequence of vertices in which consecutive terms are adjacent as a 'walk'. In such cases it should be understood that the discussion is valid for every walk with that vertex sequence.

If the edges $e_1, e_2, \dots, e_k$ of a walk W are distinct, W is called a *trail*; in this case the length of W is just $\varepsilon(W)$. If, in addition, the vertices $v_0, v_1, \dots, v_k$ are distinct, W is called a *path*. Figure 1.8 illustrates a walk, a trail and a path in a graph. We shall also use the word 'path' to denote a graph or subgraph whose vertices and edges are the terms of a path.

trail distinct edges
path distinct
vertices, edge



Walk: *uavfyfvgyhwbv*
Trail: *wcxdyhwbvgy*
Path: *xcwhyeuav*

Figure 1.8

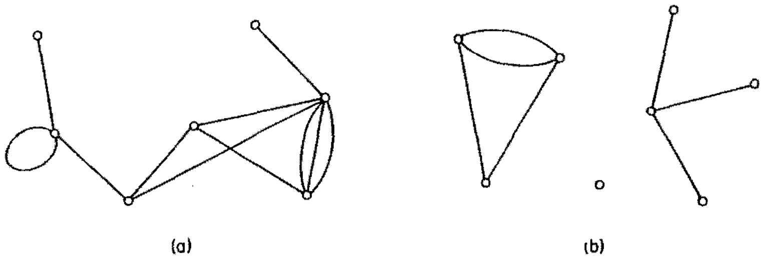


Figure 1.9. (*a*) A connected graph; (*b*) a disconnected graph with three components

Two vertices u and v of G are said to be *connected* if there is a (u, v)-path in G. Connection is an equivalence relation on the vertex set V. Thus there is a partition of V into nonempty subsets $V_1, V_2, \dots, V_\omega$ such that two vertices u and v are connected if and only if both u and v belong to the same set V_i. The subgraphs $G[V_1], G[V_2], \dots, G[V_\omega]$ are called the *components* of G. If G has exactly one component, G is *connected*; otherwise G is *disconnected*. We denote the number of components of G by $\omega(G)$. Connected and disconnected graphs are depicted in figure 1.9.

Exercises

1.6.1 Show that if there is a (u, v)-walk in G, then there is also a (u, v)-path in G.

1.6.2 Show that the number of (v_i, v_j)-walks of length k in G is the (i, j)th entry of $\mathbf{A}^k$.

1.6.3 Show that if G is simple and $\delta \ge k$, then G has a path of length k.

1.6.4 Show that G is connected if and only if, for every partition of V into two nonempty sets V_1 and V_2, there is an edge with one end in V_1 and one end in V_2.

1.6.5 (*a*) Show that if G is simple and $\varepsilon > \binom{\nu-1}{2}$, then G is connected.
(*b*) For $\nu > 1$, find a disconnected simple graph G with $\varepsilon = \binom{\nu-1}{2}$.

1.6.6 (*a*) Show that if G is simple and $\delta > [\nu/2] - 1$, then G is connected.
(*b*) Find a disconnected $([\nu/2]-1)$-regular simple graph for ν even.

1.6.7 Show that if G is disconnected, then G^c is connected.

1.6.8 (*a*) Show that if $e \in E$, then $\omega(G) \le \omega(G-e) \le \omega(G)+1$.
(*b*) Let $v \in V$. Show that $G-e$ cannot, in general, be replaced by $G-v$ in the above inequality.

1.6.9 Show that if G is connected and each degree in G is even, then, for any $v \in V$, $\omega(G-v) \le \frac{1}{2}d(v)$.

1.6.10 Show that any two longest paths in a connected graph have a vertex in common.

1.6.11 If vertices u and v are connected in G, the *distance* between u and v in G, denoted by $d_G(u, v)$, is the length of a shortest (u, v)-path in G; if there is no path connecting u and v we define $d_G(u, v)$ to be infinite. Show that, for any three vertices u, v and w, $d(u, v)+d(v, w) \geq d(u, w)$.

1.6.12 The *diameter* of G is the maximum distance between two vertices of G. Show that if G has diameter greater than three, then G^c has diameter less than three.

1.6.13 Show that if G is simple with diameter two and $\Delta = \nu - 2$, then $\varepsilon \geq 2\nu - 4$.

1.6.14 Show that if G is simple and connected but not complete, then G has three vertices u, v and w such that $uv, vw \in E$ and $uw \notin E$.

1.7 CYCLES

ot

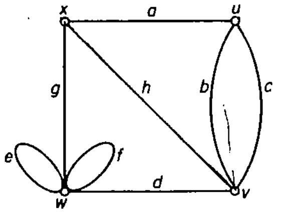
A walk is *closed* if it has positive length and its origin and terminus are the same. A closed trail whose origin and internal vertices are distinct is a *cycle*. Just as with paths we sometimes use the term 'cycle' to denote a graph corresponding to a cycle. A cycle of length k is called a *k-cycle*; a k-cycle is *odd* or *even* according as k is odd or even. A 3-cycle is often called a *triangle*. Examples of a closed trail and a cycle are given in figure 1.10.

Using the concept of a cycle, we can now present a characterisation of bipartite graphs.

Theorem 1.2 A graph is bipartite if and only if it contains no odd cycle.

Proof Suppose that G is bipartite with bipartition (X, Y), and let $C = v_0v_1 \ldots v_kv_0$ be a cycle of G. Without loss of generality we may assume that $v_0 \in X$. Then, since $v_0v_1 \in E$ and G is bipartite, $v_1 \in Y$. Similarly $v_2 \in X$ and, in general, $v_{2i} \in X$ and $v_{2i+1} \in Y$. Since $v_0 \in X$, $v_k \in Y$. Thus $k = 2i+1$, for some i, and it follows that C is even.

[illegible] to X



Closed trail: *ucvhxgwfwdvbu*
Cycle: *xaubvhx*

Figure 1.10

It clearly suffices to prove the converse for connected graphs. Let G be a connected graph that contains no odd cycles. We choose an arbitrary vertex u and define a partition (X, Y) of V by setting

distance

$$X = \{x \in V \mid d(u, x) \text{ is even}\}$$
$$Y = \{y \in V \mid d(u, y) \text{ is odd}\}$$

We shall show that (X, Y) is a bipartition of G. Suppose that v and w are two vertices of X. Let P be a shortest (u, v)-path and Q be a shortest (u, w)-path. Denote by u_1 the last vertex common to P and Q. Since P and Q are shortest paths, the (u, u_1)-sections of both P and Q are shortest (u, u_1)-paths and, therefore, have the same length. Now, since the lengths of both P and Q are even, the lengths of the (u_1, v)-section P_1 of P and the (u_1, w)-section Q_1 of Q must have the same parity. It follows that the (v, w)-path $P_1{}^{-1}Q_1$ is of even length. If v were joined to w, $P_1{}^{-1}Q_1wv$ would be a cycle of odd length, contrary to the hypothesis. Therefore no two vertices in X are adjacent; similarly, no two vertices in Y are adjacent □

Exercises

1.7.1 Show that if an edge e is in a closed trail of G, then e is in a cycle of G.

1.7.2 Show that if $\delta \geq 2$, then G contains a cycle.

1.7.3* Show that if G is simple and $\delta \geq 2$, then G contains a cycle of length at least $\delta + 1$.

1.7.4 The *girth* of G is the length of a shortest cycle in G; if G has no cycles we define the girth of G to be infinite. Show that

(*a*) a k-regular graph of girth four has at least $2k$ vertices, and (up to isomorphism) there exists exactly one such graph on $2k$ vertices;

(*b*) a k-regular graph of girth five has at least k^2+1 vertices.

1.7.5 Show that a k-regular graph of girth five and diameter two has exactly k^2+1 vertices, and find such a graph for $k = 2, 3$. (Hoffman and Singleton, 1960 have shown that such a graph can exist only if $k = 2, 3, 7$ and, possibly, 57.)

1.7.6 Show that

(*a*) if $\varepsilon \geq \nu$, G contains a cycle;

(*b*)* if $\varepsilon \geq \nu + 4$, G contains two edge-disjoint cycles. (L. Pósa)

APPLICATIONS

1.8 THE SHORTEST PATH PROBLEM

With each edge e of G let there be associated a real number $w(e)$, called its *weight*. Then G, together with these weights on its edges, is called a *weighted*

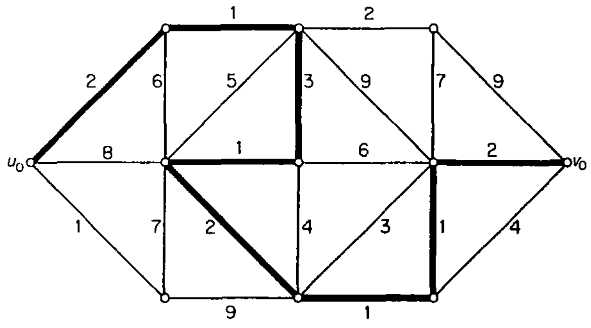


Figure 1.11. A (u_0, v_0)-path of minimum weight

graph. Weighted graphs occur frequently in applications of graph theory. In the friendship graph, for example, weights might indicate intensity of friendship; in the communications graph, they could represent the construction or maintenance costs of the various communication links.

If H is a subgraph of a weighted graph, the *weight* $w(H)$ of H is the sum of the weights $\sum_{e \in E(H)} w(e)$ on its edges. Many optimisation problems amount to finding, in a weighted graph, a subgraph of a certain type with minimum (or maximum) weight. One such is the *shortest path problem:* given a railway network connecting various towns, determine a shortest route between two specified towns in the network.

Here one must find, in a weighted graph, a path of minimum weight connecting two specified vertices u_0 and v_0; the weights represent distances by rail between directly-linked towns, and are therefore non-negative. The path indicated in the graph of figure 1.11 is a (u_0, v_0)-path of minimum weight (exercise 1.8.1).

We now present an algorithm for solving the shortest path problem. For clarity of exposition, we shall refer to the weight of a path in a weighted graph as its *length*; similarly the minimum weight of a (u, v)-path will be called the *distance* between u and v and denoted by $d(u, v)$. These definitions coincide with the usual notions of length and distance, as defined in section 1.6, when all the weights are equal to one.

It clearly suffices to deal with the shortest path problem for simple graphs; so we shall assume here that G is simple. We shall also assume that all the weights are positive. This, again, is not a serious restriction because, if the weight of an edge is zero, then its ends can be identified. We adopt the convention that $w(uv) = \infty$ if $uv \notin E$.

The algorithm to be described was discovered by Dijkstra (1959) and, independently, by Whiting and Hillier (1960). It finds not only a shortest (u_0, v_0)-path, but shortest paths from u_0 to all other vertices of G. The basic idea is as follows.

Suppose that S is a proper subset of V such that $u_0 \in S$, and let $\bar{S}$ denote $V \backslash S$. If $P = u_0 \ldots \bar{u}\bar{v}$ is a shortest path from u_0 to $\bar{S}$ then clearly $\bar{u} \in S$ and the $(u_0, \bar{u})$-section of P must be a shortest $(u_0, \bar{u})$-path. Therefore

> to an element in $\bar{S}$

$$d(u_0, \bar{v}) = d(u_0, \bar{u}) + w(\bar{u}\bar{v})$$

and the distance from u_0 to $\bar{S}$ is given by the formula

$$d(u_0, \bar{S}) = \min_{\substack{u \in S \\ v \in \bar{S}}} \{d(u_0, u) + w(uv)\} \tag{1.1}$$

This formula is the basis of Dijkstra's algorithm. Starting with the set $S_0 = \{u_0\}$, an increasing sequence $S_0, S_1, \ldots, S_{\nu-1}$ of subsets of V is constructed, in such a way that, at the end of stage i, shortest paths from u_0 to all vertices in S_i are known.

The first step is to determine a vertex nearest to u_0. This is achieved by computing $d(u_0, \bar{S}_0)$ and selecting a vertex $u_1 \in \bar{S}_0$ such that $d(u_0, u_1) = d(u_0, \bar{S}_0)$; by (1.1)

$$d(u_0, \bar{S}_0) = \min_{\substack{u \in S_0 \\ v \in \bar{S}_0}} \{d(u_0, u) + w(uv)\} = \min_{v \in \bar{S}_0} \{w(u_0 v)\}$$

and so $d(u_0, \bar{S}_0)$ is easily computed. We now set $S_1 = \{u_0, u_1\}$ and let P_1 denote the path u_0u_1; this is clearly a shortest (u_0, u_1)-path. In general, if the set $S_k = \{u_0, u_1, \ldots, u_k\}$ and corresponding shortest paths $P_1, P_2, \ldots, P_k$ have already been determined, we compute $d(u_0, \bar{S}_k)$ using (1.1) and select a vertex $u_{k+1} \in \bar{S}_k$ such that $d(u_0, u_{k+1}) = d(u_0, \bar{S}_k)$. By (1.1), $d(u_0, u_{k+1}) = d(u_0, u_j) + w(u_j u_{k+1})$ for some $j \leq k$; we get a shortest (u_0, u_{k+1})-path by adjoining the edge $u_j u_{k+1}$ to the path P_j.

We illustrate this procedure by considering the weighted graph depicted in figure 1.12*a*. Shortest paths from u_0 to the remaining vertices are determined in seven stages. At each stage, the vertices to which shortest paths have been found are indicated by solid dots, and each is labelled by its distance from u_0; initially u_0 is labelled 0. The actual shortest paths are indicated by solid lines. Notice that, at each stage, these shortest paths together form a connected graph without cycles; such a graph is called a *tree*, and we can think of the algorithm as a 'tree-growing' procedure. The final tree, in figure 1.12*h*, has the property that, for each vertex v, the path connecting u_0 and v is a shortest (u_0, v)-path.

Dijkstra's algorithm is a refinement of the above procedure. This refinement is motivated by the consideration that, if the minimum in (1.1) were to be computed from scratch at each stage, many comparisons would be

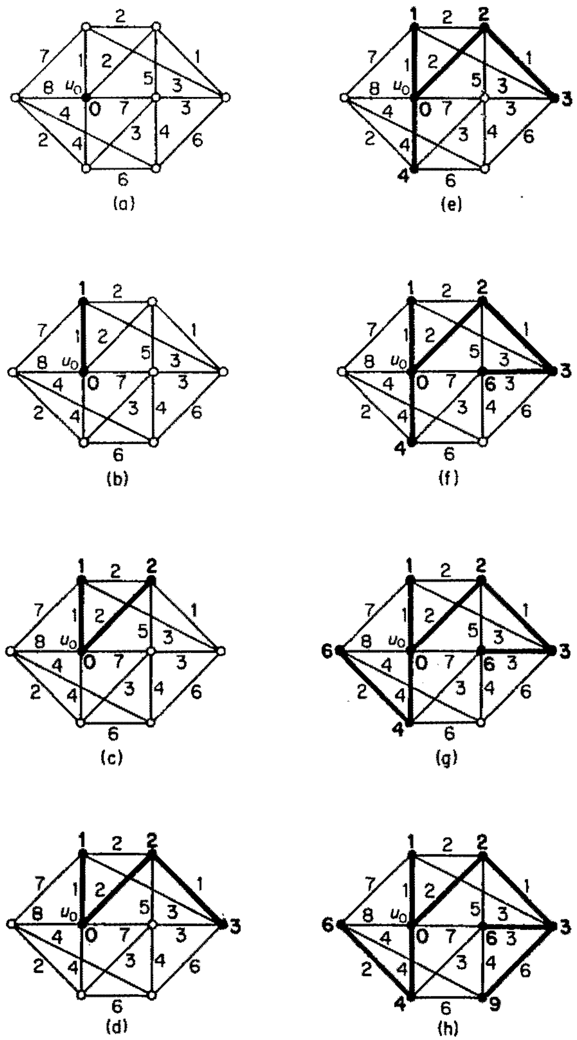


Figure 1.12. Shortest path algorithm

repeated unnecessarily. To avoid such repetitions, and to retain computational information from one stage to the next, we adopt the following labelling procedure. Throughout the algorithm, each vertex v carries a label $l(v)$ which is an upper bound on $d(u_0, v)$. Initially $l(u_0) = 0$ and $l(v) = \infty$ for $v \neq u_0$. (In actual computations ∞ is replaced by any sufficiently large number.) As the algorithm proceeds, these labels are modified so that, at the end of stage i,

$$l(u) = d(u_0, u) \quad \text{for} \quad u \in S_i$$

and

$$l(v) = \min_{u \in S_{i-1}}\{d(u_0, u) + w(uv)\} \quad \text{for} \quad v \in \bar{S}_i$$

Dijkstra's Algorithm

1. Set $l(u_0) = 0$, $l(v) = \infty$ for $v \neq u_0$, $S_0 = \{u_0\}$ and $i = 0$.
2. For each $v \in \bar{S}_i$, replace $l(v)$ by $\min\{l(v), l(u_i) + w(u_iv)\}$. Compute $\min_{v \in \bar{S}_i}\{l(v)\}$ and let u_{i+1} denote a vertex for which this minimum is attained. Set $S_{i+1} = S_i \cup \{u_{i+1}\}$.
3. If $i = \nu - 1$, stop. If $i < \nu - 1$, replace i by $i + 1$ and go to step 2.

When the algorithm terminates, the distance from u_0 to v is given by the final value of the label $l(v)$. (If our interest is in determining the distance to one specific vertex v_0, we stop as soon as some u_j equals v_0.) A flow diagram summarising this algorithm is shown in figure 1.13.

As described above, Dijkstra's algorithm determines only the distances from u_0 to all the other vertices, and not the actual shortest paths. These shortest paths can, however, be easily determined by keeping track of the predecessors of vertices in the tree (exercise 1.8.2).

Dijkstra's algorithm is an example of what Edmonds (1965) calls a good algorithm. A graph-theoretic algorithm is *good* if the number of computational steps required for its implementation on any graph G is bounded above by a polynomial in ν and ε (such as $3\nu^2\varepsilon$). An algorithm whose implementation may require an exponential number of steps (such as 2^ν) might be very inefficient for some large graphs.

To see that Dijkstra's algorithm is good, note that the computations involved in boxes 2 and 3 of the flow diagram, totalled over all iterations, require $\nu(\nu-1)/2$ additions and $\nu(\nu-1)$ comparisons. One of the questions that is not elaborated upon in the flow diagram is the matter of deciding whether a vertex belongs to $\bar{S}$ or not (box 1). Dreyfus (1969) reports a technique for doing this that requires a total of $(\nu-1)^2$ comparisons. Hence, if we regard either a comparison or an addition as a basic computational unit, the total number of computations required for this algorithm is approximately $5\nu^2/2$, and thus of order ν^2. (A function $f(\nu, \varepsilon)$ is of *order*

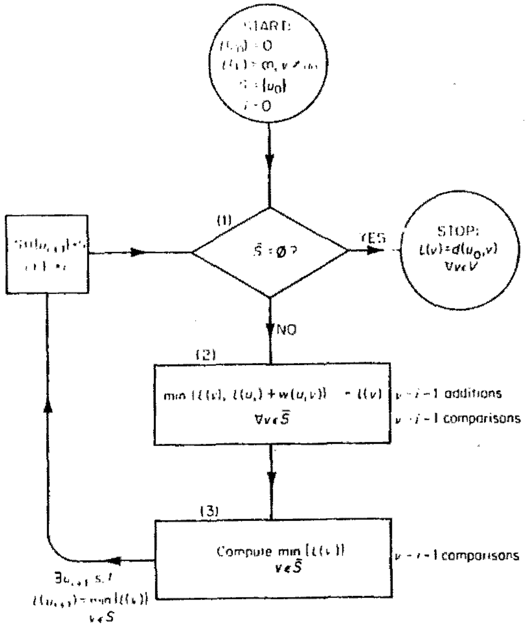


Figure 1.13. Dijkstra's algorithm

$g(\nu, \varepsilon)$ if there exists a positive constant c such that $f(\nu, \varepsilon)/g(\nu, \varepsilon)\leq c$ for all ν and ε.)

Although the shortest path problem can be solved by a good algorithm, there are many problems in graph theory for which no good algorithm is known. We refer the reader to Aho, Hopcroft and Ullman (1974) for further details.

Exercises

1.8.1 Find shortest paths from u_0 to all other vertices in the weighted graph of figure 1.11.

1.8.2 What additional instructions are needed in order that Dijkstra's algorithm determine shortest paths rather than merely distances?

1.8.3 A company has branches in each of six cities $C_1, C_2, \ldots, C_6$. The fare for a direct flight from C_i to C_j is given by the (i, j)th entry in the following matrix (∞ indicates that there is no direct flight):

$$\begin{bmatrix} 0 & 50 & \infty & 40 & 25 & 10 \\ 50 & 0 & 15 & 20 & \infty & 25 \\ \infty & 15 & 0 & 10 & 20 & \infty \\ 40 & 20 & 10 & 0 & 10 & 25 \\ 25 & \infty & 20 & 10 & 0 & 55 \\ 10 & 25 & \infty & 25 & 55 & 0 \end{bmatrix}$$

The company is interested in computing a table of cheapest routes between pairs of cities. Prepare such a table.

1.8.4 A wolf, a goat and a cabbage are on one bank of a river. A ferryman wants to take them across, but, since his boat is small, he can take only one of them at a time. For obvious reasons, neither the wolf and the goat nor the goat and the cabbage can be left unguarded. How is the ferryman going to get them across the river?

1.8.5 Two men have a full eight-gallon jug of wine, and also two empty jugs of five and three gallons capacity, respectively. What is the simplest way for them to divide the wine equally?

1.8.6 Describe a good algorithm for determining
(*a*) the components of a graph;
(*b*) the girth of a graph.
How good are your algorithms?

1.9 SPERNER'S LEMMA

Every continuous mapping f of a closed n-disc to itself has a fixed point (that is, a point x such that $f(x)=x$). This powerful theorem, known as *Brouwer's fixed-point theorem*, has a wide range of applications in modern mathematics. Somewhat surprisingly, it is an easy consequence of a simple combinatorial lemma due to Sperner (1928). And, as we shall see in this section, Sperner's lemma is, in turn, an immediate consequence of corollary 1.1.

Sperner's lemma concerns the decomposition of a simplex (line segment, triangle, tetrahedron and so on) into smaller simplices. For the sake of simplicity we shall deal with the two-dimensional case.

Let T be a closed triangle in the plane. A subdivision of T into a finite number of smaller triangles is said to be *simplicial* if any two intersecting triangles have either a vertex or a whole side in common (see figure 1.14*a*).

Suppose that a simplicial subdivision of T is given. Then a labelling of the vertices of triangles in the subdivision in three symbols 0, 1 and 2 is said to be *proper* if

(i) the three vertices of T are labelled 0, 1 and 2 (in any order), and
(ii) for $0\leq i<j\leq 2$, each vertex on the side of T joining vertices labelled i and j is labelled either i or j.

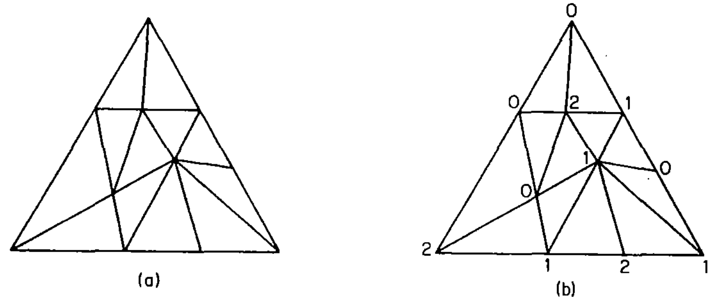


Figure 1.14. (*a*) A simplicial subdivision of a triangle; (*b*) a proper labelling of the subdivision

We call a triangle in the subdivision whose vertices receive all three labels a *distinguished* triangle. The proper labelling in figure 1.14*b* has three distinguished triangles.

Theorem 1.3 (*Sperner's lemma*) Every properly labelled simplicial subdivision of a triangle has an odd number of distinguished triangles.

Proof Let T_0 denote the region outside T, and let $T_1, T_2, \ldots, T_n$ be the triangles of the subdivision. Construct a graph on the vertex set $\{v_0, v_1, \ldots, v_n\}$ by joining v_i and v_j whenever the common boundary of T_i and T_j is an edge with labels 0 and 1 (see figure 1.15).

In this graph, v_0 is clearly of odd degree (exercise 1.9.1). It follows from corollary 1.1 that an odd number of the vertices $v_1, v_2, \ldots, v_n$ are of odd degree. Now it is easily seen that none of these vertices can have degree three, and so those with odd degree must have degree one. But a vertex v_i is of degree one if and only if the triangle T_i is distinguished □

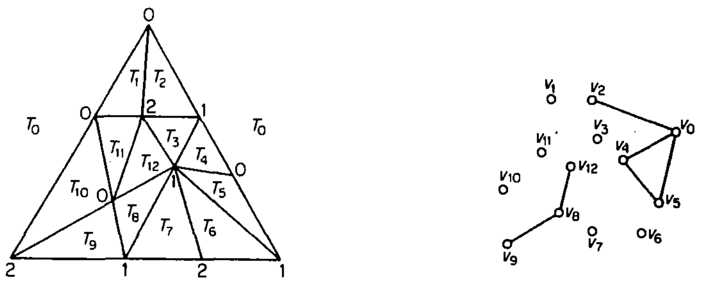


Figure 1.15

We shall now briefly indicate how Sperner's lemma can be used to deduce Brouwer's fixed-point theorem. Again, for simplicity, we shall only deal with the two-dimensional case. Since a closed 2-disc is homeomorphic to a closed triangle, it suffices to prove that a continuous mapping of a closed triangle to itself has a fixed point.

Let T be a given closed triangle with vertices x_0, x_1 and x_2. Then each point x of T can be written uniquely as $x = a_0x_0 + a_1x_1 + a_2x_2$, where each $a_i \geq 0$ and $\Sigma\, a_i = 1$, and we can represent x by the vector (a_0, a_1, a_2); the real numbers a_0, a_1 and a_2 are called the *barycentric coordinates* of x.

Now let f be any continuous mapping of T to itself, and suppose that

$$f(a_0, a_1, a_2) = (a_0', a_1', a_2')$$

Define S_i as the set of points (a_0, a_1, a_2) in T for which $a_i' \leq a_i$. To show that f has a fixed point, it is enough to show that $S_0 \cap S_1 \cap S_2 \neq \emptyset$. For suppose that $(a_0, a_1, a_2) \in S_0 \cap S_1 \cap S_2$. Then, by the definition of S_i, we have that $a_i' \leq a_i$ for each i, and this, coupled with the fact that $\Sigma\, a_i' = \Sigma\, a_i$, yields

$$(a_0', a_1', a_2') = (a_0, a_1, a_2)$$

In other words, (a_0, a_1, a_2) is a fixed point of f.

So consider an arbitrary subdivision of T and a proper labelling such that each vertex labelled i belongs to S_i; the existence of such a labelling is easily seen (exercise 1.9.2*a*). It follows from Sperner's lemma that there is a triangle in the subdivision whose three vertices belong to S_0, S_1 and S_2. Now this holds for any subdivision of T and, since it is possible to choose subdivisions in which each of the smaller triangles are of arbitrarily small diameter, we conclude that there exist three points of S_0, S_1 and S_2 which are arbitrarily close to one another. Because the sets S_i are closed (exercise 1.9.2*b*), one may deduce that $S_0 \cap S_1 \cap S_2 \neq \emptyset$.

For details of the above proof and other applications of Sperner's lemma, the reader is referred to Tompkins (1964).

Exercises

1.9.1 In the proof of Sperner's lemma, show that the vertex v_0 is of odd degree.

1.9.2 In the proof of Brouwer's fixed-point theorem, show that

(*a*) there exists a proper labelling such that each vertex labelled i belongs to S_i;

(*b*) the sets S_i are closed.

1.9.3 State and prove Sperner's lemma for higher dimensional simplices.

REFERENCES

Aho, A. V., Hopcroft, J. E. and Ullman, J. D. (1974). *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numer. Math.*, **1,** 269–71

Dreyfus, S. E. (1969). An appraisal of some shortest-path algorithms. *Operations Res.*, **17,** 395–412

Edmonds, J. (1965). Paths, trees and flowers. *Canad. J. Math.*, **17,** 449–67

Erdös, P. and Gallai, T. (1960). Graphs with prescribed degrees of vertices (Hungarian). *Mat. Lapok*, **11,** 264–74

Frucht, R. (1939). Herstellung von Graphen mit vorgegebener abstrakter Gruppe. *Compositio Math.*, **6,** 239–50

Hoffman, A. J. and Singleton, R. R. (1960). On Moore graphs with diameters 2 and 3. *IBM J. Res. Develop.*, **4,** 497–504

Sperner, E. (1928). Neuer Beweis für die Invarianz der Dimensionszahl und des Gebietes. *Hamburger Abhand.*, **6,** 265–72

Tompkins, C. B. (1964). Sperner's lemma and some extensions, in *Applied Combinatorial Mathematics*, ch. 15 (ed. E. F. Beckenbach), Wiley, New York, pp. 416–55

Whiting, P. D. and Hillier, J. A. (1960). A method for finding the shortest route through a road network. *Operational Res. Quart.*, **11,** 37–40

2 Trees

2.1 TREES

An *acyclic* graph is one that contains no cycles. A *tree* is a connected acyclic graph. The trees on six vertices are shown in figure 2.1.

Theorem 2.1 In a tree, any two vertices are connected by a unique path.

Proof By contradiction. Let G be a tree, and assume that there are two distinct (u, v)-paths P_1 and P_2 in G. Since $P_1 \neq P_2$, there is an edge $e = xy$ of P_1 that is not an edge of P_2. Clearly the graph $(P_1 \cup P_2) - e$ is connected. It therefore contains an (x, y)-path P. But then $P + e$ is a cycle in the acyclic graph G, a contradiction □

The converse of this theorem holds for graphs without loops (exercise 2.1.1).

Observe that all the trees on six vertices (figure 2.1) have five edges. In general we have:

Theorem 2.2 If G is a tree, then $\varepsilon = \nu - 1$.

Proof By induction on ν. When $\nu = 1$, $G \cong K_1$ and $\varepsilon = 0 = \nu - 1$.

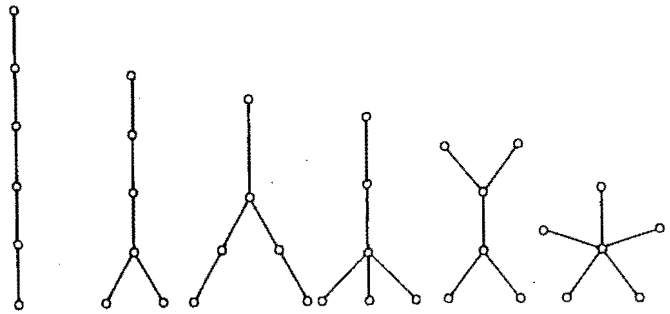


Figure 2.1. The trees on six vertices

Suppose the theorem true for all trees on fewer than ν vertices, and let G be a tree on $\nu \geq 2$ vertices. Let $uv \in E$. Then $G - uv$ contains no (u, v)-path, since uv is the unique (u, v)-path in G. Thus $G - uv$ is disconnected and so (exercise 1.6.8a) $\omega(G - uv) = 2$. The components G_1 and G_2 of $G - uv$, being acyclic, are trees. Moreover, each has fewer than ν vertices. Therefore, by the induction hypothesis

$$\varepsilon(G_i) = \nu(G_i) - 1 \quad \text{for} \quad i = 1, 2$$

Thus

$$\varepsilon(G) = \varepsilon(G_1) + \varepsilon(G_2) + 1 = \nu(G_1) + \nu(G_2) - 1 = \nu(G) - 1 \quad \square$$

Corollary 2.2 Every nontrivial tree has at least two vertices of degree one.

Proof Let G be a nontrivial tree. Then

$$d(v) \geq 1 \quad \text{for all} \quad v \in V$$

Also, by theorems 1.1 and 2.2, we have

$$\sum_{v \in V} d(v) = 2\varepsilon = 2\nu - 2$$

It now follows that $d(v) = 1$ for at least two vertices v $\square$

Another, perhaps more illuminating, way of proving corollary 2.2 is to show that the origin and terminus of a longest path in a nontrivial tree both have degree one (see exercise 2.1.2).

Exercises

2.1.1 Show that if any two vertices of a loopless graph G are connected by a unique path, then G is a tree.

2.1.2 Prove corollary 2.2 by showing that the origin and terminus of a longest path in a nontrivial tree both have degree one.

2.1.3 Prove corollary 2.2 by using exercise 1.7.2.

2.1.4 Show that every tree with exactly two vertices of degree one is a path.

2.1.5 Let G be a graph with $\nu - 1$ edges. Show that the following three statements are equivalent:

(a) G is connected;
(b) G is acyclic;
(c) G is a tree.

2.1.6 Show that if G is a tree with $\Delta \geq k$, then G has at least k vertices of degree one.

2.1.7 An acyclic graph is also called a *forest*. Show that

(a) each component of a forest is a tree;
(b) G is a forest if and only if $\varepsilon = \nu - \omega$.

2.1.8 A *centre* of G is a vertex u such that $\max_{v \in V} d(u, v)$ is as small as possible. Show that a tree has either exactly one centre or two, adjacent, centres.

2.1.9 Show that if G is a forest with exactly $2k$ vertices of odd degree, then there are k edge-disjoint paths $P_1, P_2, \ldots, P_k$ in G such that $E(G) = E(P_1) \cup E(P_2) \cup \ldots \cup E(P_k)$.

2.1.10* Show that a sequence $(d_1, d_2, \ldots, d_\nu)$ of positive integers is a degree sequence of a tree if and only if $\sum_{i=1}^{\nu} d_i = 2(\nu - 1)$.

2.1.11 Let T be an arbitrary tree on $k + 1$ vertices. Show that if G is simple and $\delta \geq k$ then G has a subgraph isomorphic to T.

2.1.12 A saturated hydrocarbon is a molecule C_mH_n in which every carbon atom has four bonds, every hydrogen atom has one bond, and no sequence of bonds forms a cycle. Show that, for every positive integer m, C_mH_n can exist only if $n = 2m + 2$.

2.2 CUT EDGES AND BONDS

A *cut edge* of G is an edge e such that $\omega(G - e) > \omega(G)$. The graph of figure 2.2 has the three cut edges indicated.

Theorem 2.3 An edge e of G is a cut edge of G if and only if e is contained in no cycle of G.

$\Rightarrow$

Proof Let e be a cut edge of G. Since $\omega(G - e) > \omega(G)$, there exist vertices u and v of G that are connected in G but not in $G - e$. There is therefore some (u, v)-path P in G which, necessarily, traverses e. Suppose that x and y are the ends of e, and that x precedes y on P. In $G - e$, u is connected to x by a section of P and y is connected to v by a section of P. If e were in a cycle C, x and y would be connected in $G - e$ by the path $C - e$. Thus, u and v would be connected in $G - e$, a contradiction.

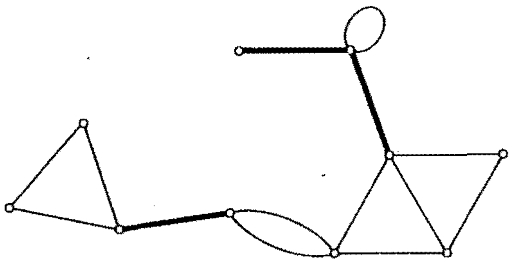


Figure 2.2. The cut edges of a graph

Conversely, suppose that $e = xy$ is not a cut edge of G; thus, $\omega(G-e) = \omega(G)$. Since there is an (x, y)-path (namely xy) in G, x and y are in the same component of G. It follows that x and y are in the same component of $G-e$, and hence that there is an (x, y)-path P in $G-e$. But then e is in the cycle $P+e$ of G □

Theorem 2.4 A connected graph is a tree if and only if every edge is a cut edge.

Proof Let G be a tree and let e be an edge of G. Since G is acyclic, e is contained in no cycle of G and is therefore, by theorem 2.3, a cut edge of G.

Conversely, suppose that G is connected but is not a tree. Then G contains a cycle C. By theorem 2.3, no edge of C can be a cut edge of G □

A *spanning tree* of G is a spanning subgraph of G that is a tree.

Corollary 2.4.1 Every connected graph contains a spanning tree.

Proof Let G be connected and let T be a minimal connected spanning subgraph of G. By definition $\omega(T) = 1$ and $\omega(T-e) > 1$ for each edge e of T. It follows that each edge of T is a cut edge and therefore, by theorem 2.4, that T, being connected, is a tree □

Figure 2.3 depicts a connected graph and one of its spanning trees.

Corollary 2.4.2 If G is connected, then $\varepsilon \geq \nu - 1$.

Proof Let G be connected. By corollary 2.4.1, G contains a spanning tree T. Therefore

$$\varepsilon(G) \geq \varepsilon(T) = \nu(T) - 1 = \nu(G) - 1 \quad \square$$

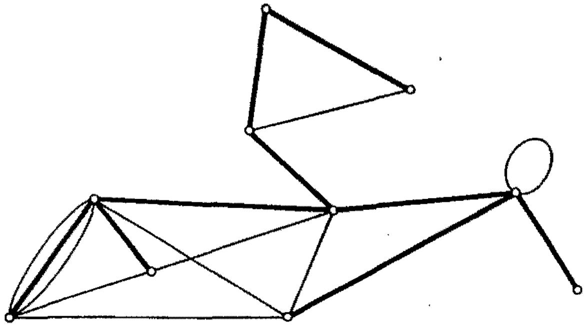


Figure 2.3. A spanning tree in a connected graph

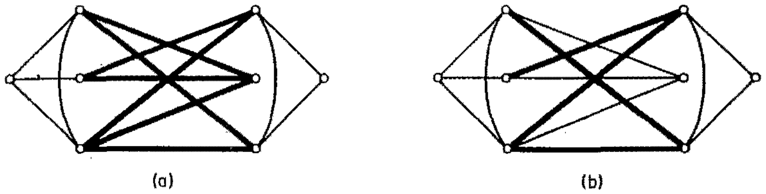


Figure 2.4. (*a*) An edge cut; (*b*) a bond

Theorem 2.5 Let T be a spanning tree of a connected graph G and let e be an edge of G not in T. Then $T+e$ contains a unique cycle.

Proof Since T is acyclic, each cycle of $T+e$ contains e. Moreover, C is a cycle of $T+e$ if and only if $C-e$ is a path in T connecting the ends of e. By theorem 2.1, T has a unique such path; therefore $T+e$ contains a unique cycle □

include all nodes in [illegible]

For subsets S and S' of V, we denote by $[S, S']$ the set of edges with one end in S and the other in S'. An *edge cut* of G is a subset of E of the form $\boxed{[S, \bar{S}]}$, where S is a nonempty proper subset of V and $\bar{S} = V \backslash S$. A minimal nonempty edge cut of G is called a *bond*; each cut edge e, for instance, gives rise to a bond $\{e\}$. If G is connected, then a bond B of G is a minimal subset of E such that $G-B$ is disconnected. Figure 2.4 indicates an edge cut and a bond in a graph.

any edge away get no edge cut

If H is a subgraph of G, the *complement of H in G*, denoted by $\bar{H}(G)$, is the subgraph $G-E(H)$. If G is connected, a subgraph of the form $\bar{T}$, where T is a spanning tree, is called a *cotree* of G.

Theorem 2.6 Let T be a spanning tree of a connected graph G, and let e be any edge of T. Then

(i) the cotree $\bar{T}$ contains no bond of G; [illegible] connected

(ii) $\bar{T}+e$ contains a unique bond of G. [illegible] e [illegible] proof

Proof (i) Let B be a bond of G. Then $G-B$ is disconnected, and so cannot contain the spanning tree T. Therefore B is not contained in $\bar{T}$. (ii) Denote by S the vertex set of one of the two components of $T-e$. The edge cut $B = [S, \bar{S}]$ is clearly a bond of G, and is contained in $\bar{T}+e$. Now, for any $b \in B$, $T-e+b$ is a spanning tree of G. Therefore every bond of G contained in $\bar{T}+e$ must include every such element b. It follows that B is the only bond of G contained in $\bar{T}+e$ □

The relationship between bonds and cotrees is analogous to that between cycles and spanning trees. Statement (i) of theorem 2.6 is the analogue for bonds of the simple fact that a spanning tree is acyclic, and (ii) is the analogue of theorem 2.5. This 'duality' between cycles and bonds will be further explored in chapter 12 (see also exercise 2.2.10).

Exercises

2.2.1 Show that G is a forest if and only if every edge of G is a cut edge.

2.2.2 Let G be connected and let $e \in E$. Show that

(*a*) e is in every spanning tree of G if and only if e is a cut edge of G;

(*b*) e is in no spanning tree of G if and only if e is a loop of G.

2.2.3 Show that if G is loopless and has exactly one spanning tree T, then $G = T$.

2.2.4 Let F be a maximal forest of G. Show that

(*a*) for every component H of G, $F \cap H$ is a spanning tree of H;

(*b*) $\varepsilon(F) = \nu(G) - \omega(G)$.

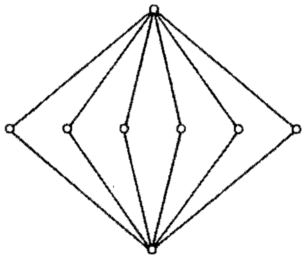
2.2.5 Show that G contains at least $\varepsilon - \nu + \omega$ distinct cycles.

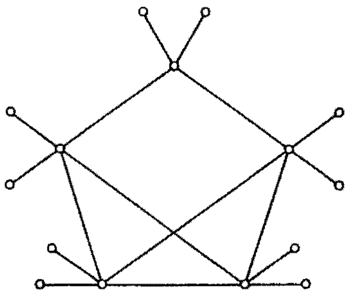
2.2.6 Show that

(*a*) if each degree in G is even, then G has no cut edge;

(*b*) if G is a k-regular bipartite graph with $k \geq 2$, then G has no cut edge.

2.2.7 Find the number of nonisomorphic spanning trees in the following graphs:





2.2.8 Let G be connected and let S be a nonempty proper subset of V. Show that the edge cut $[S, \bar{S}]$ is a bond of G if and only if both $G[S]$ and $G[\bar{S}]$ are connected.

2.2.9 Show that every edge cut is a disjoint union of bonds.

2.2.10 Let B_1 and B_2 be bonds and let C_1 and C_2 be cycles (regarded as sets of edges) in a graph. Show that

(*a*) $B_1 \Delta B_2$ is a disjoint union of bonds;

(*b*) $C_1 \Delta C_2$ is a disjoint union of cycles,

where Δ denotes symmetric difference;

(*c*) for any edge e, $(B_1 \cup B_2)\backslash\{e\}$ contains a bond;

(*d*) for any edge e, $(C_1 \cup C_2)\backslash\{e\}$ contains a cycle.

2.2.11 Show that if a graph G contains k edge-disjoint spanning trees then, for each partition $(V_1, V_2, \ldots, V_n)$ of V, the number of edges which have ends in different parts of the partition is at least $k(n-1)$.

(Tutte, 1961 and Nash-Williams, 1961 have shown that this necessary condition for G to contain k edge-disjoint spanning trees is also sufficient.)

2.2.12* Let S be an n-element set, and let $\mathscr{A} = \{A_1, A_2, \ldots, A_n\}$ be a family of n distinct subsets of S. Show that there is an element $x \in S$ such that the sets $A_1 \cup \{x\}$, $A_2 \cup \{x\}, \ldots, A_n \cup \{x\}$ are all distinct.

2.3 CUT VERTICES

A vertex v of G is a *cut vertex* if E can be partitioned into two nonempty subsets E_1 and E_2 such that $G[E_1]$ and $G[E_2]$ have just the vertex v in common. If G is loopless and nontrivial, then v is a cut vertex of G if and only if $\omega(G-v) > \omega(G)$. The graph of figure 2.5 has the five cut vertices indicated.

Theorem 2.7 A vertex v of a tree G is a cut vertex of G if and only if $d(v) > 1$.

Proof If $d(v) = 0$, $G \cong K_1$ and, clearly, v is not a cut vertex.

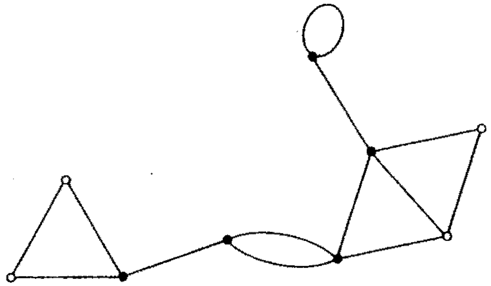


Figure 2.5. The cut vertices of a graph

If $d(v)=1$, $G-v$ is an acyclic graph with $\nu(G-v)-1$ edges, and thus (exercise 2.1.5) a tree. Hence $\omega(G-v)=1=\omega(G)$, and v is not a cut vertex of G.

If $d(v)>1$, there are distinct vertices u and w adjacent to v. The path uvw is a (u, w)-path in G. By theorem 2.1 uvw is the unique (u, w)-path in G. It follows that there is no (u, w)-path in $G-v$, and therefore that $\omega(G-v)>1=\omega(G)$. Thus v is a cut vertex of G □

Corollary 2.7 Every nontrivial loopless connected graph has at least two vertices that are not cut vertices.

Proof Let G be a nontrivial loopless connected graph. By corollary 2.4.1, G contains a spanning tree T. By corollary 2.2 and theorem 2.7, T has at least two vertices that are not cut vertices. Let v be any such vertex. Then

$$\omega(T-v)=1$$

Since T is a spanning subgraph of G, $T-v$ is a spanning subgraph of $G-v$ and therefore

$$\omega(G-v)\leq\omega(T-v)$$

It follows that $\omega(G-v)=1$, and hence that v is not a cut vertex of G. Since there are at least two such vertices v, the proof is complete □

Exercises

2.3.1 Let G be connected with $\nu\geq 3$. Show that

(*a*) if G has a cut edge, then G has a vertex v such that $\omega(G-v)>\omega(G)$;

(*b*) the converse of (*a*) is not necessarily true.

2.3.2 Show that a simple connected graph that has exactly two vertices which are not cut vertices is a path.

2.4 CAYLEY'S FORMULA

There is a simple and elegant recursive formula for the number of spanning trees in a graph. It involves the operation of contraction of an edge, which we now introduce. An edge e of G is said to be *contracted* if it is deleted and its ends are identified; the resulting graph is denoted by $G\cdot e$. Figure 2.6 illustrates the effect of contracting an edge.

It is clear that if e is a link of G, then

$$\nu(G\cdot e)=\nu(G)-1 \qquad \varepsilon(G\cdot e)=\varepsilon(G)-1 \quad \text{and} \quad \omega(G\cdot e)=\omega(G)$$

Therefore, if T is a tree, so too is $T\cdot e$.

We denote the number of spanning trees of G by $\tau(G)$.

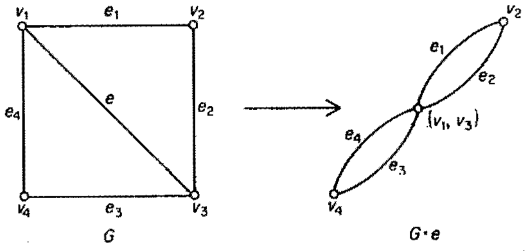


Figure 2.6. Contraction of an edge

Theorem 2.8 If e is a link of G, then $\tau(G)=\tau(G-e)+\tau(G\cdot e)$.

Proof Since every spanning tree of G that does not contain e is also a spanning tree of $G-e$, and conversely, $\tau(G-e)$ is the number of spanning trees of G that do not contain e.

Now to each spanning tree T of G that contains e, there corresponds a spanning tree $T\cdot e$ of $G\cdot e$. This correspondence is clearly a bijection (see figure 2.7). Therefore $\tau(G\cdot e)$ is precisely the number of spanning trees of G that contain e. It follows that $\tau(G)=\tau(G-e)+\tau(G\cdot e)$ □

Figure 2.8 illustrates the recursive calculation of $\tau(G)$ by means of theorem 2.8; the number of spanning trees in a graph is represented symbolically by the graph itself.

Although theorem 2.8 provides a method of calculating the number of spanning trees in a graph, this method is not suitable for large graphs. Fortunately, and rather surprisingly, there is a closed formula for $\tau(G)$ which expresses $\tau(G)$ as a determinant; we shall present this result in chapter 12. In the special case when G is complete, a simple formula for $\tau(G)$ was discovered by Cayley (1889). The proof we give is due to Prüfer (1918).

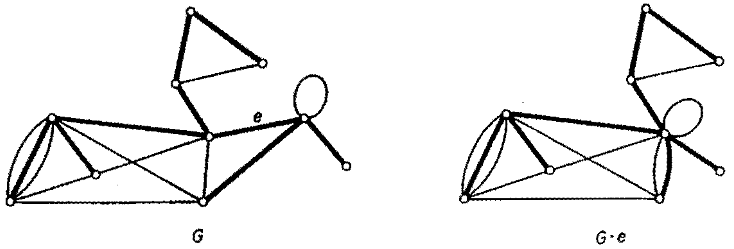


Figure 2.7

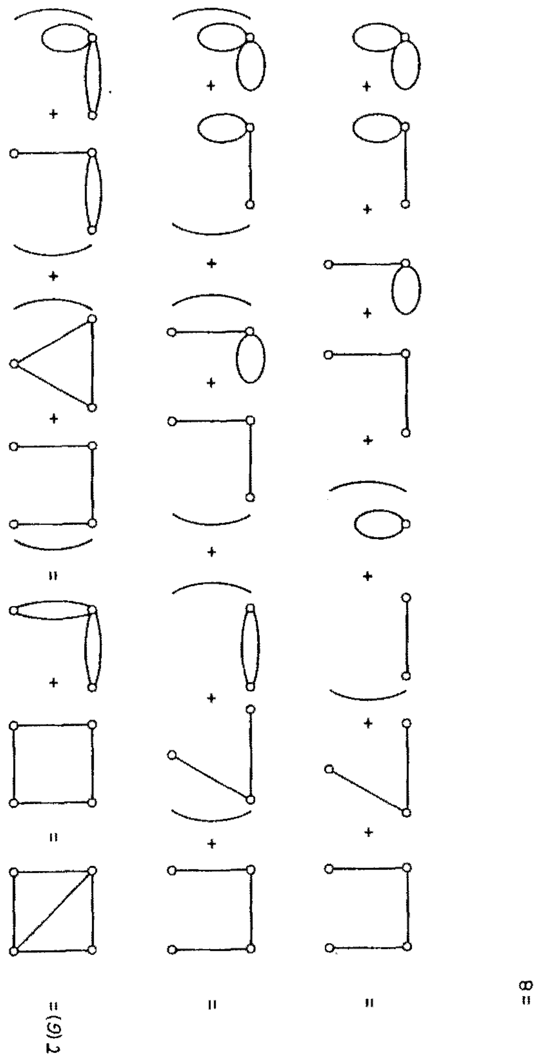


Figure 2.8. Recursive calculation of τ(G)

T 'parent tree', to tree

Theorem 2.9 $\tau(K_n) = n^{n-2}$.

Proof Let the vertex set of K_n be $N = \{1, 2, \ldots, n\}$. We note that n^{n-2} is the number of sequences of length $n-2$ that can be formed from N. Thus, to prove the theorem, it suffices to establish a one–one correspondence between the set of spanning trees of K_n and the set of such sequences.

With each spanning tree T of K_n, we associate a unique sequence $(t_1, t_2, \ldots, t_{n-2})$ as follows. Regarding N as an ordered set, let s_1 be the first vertex of degree one in T; the vertex adjacent to s_1 is taken as t_1. We now delete s_1 from T, denote by s_2 the first vertex of degree one in $T - s_1$, and take the vertex adjacent to s_2 as t_2. This operation is repeated until t_{n-2} has been defined and a tree with just two vertices remains; the tree in figure 2.9, for instance, gives rise to the sequence (4, 3, 5, 3, 4, 5). It can be seen that different spanning trees of K_n determine difference sequences.

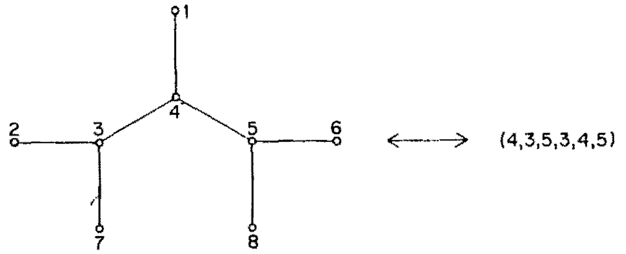


Figure 2.9

The reverse procedure is equally straightforward. Observe, first, that any vertex v of T occurs $d_T(v) - 1$ times in $(t_1, t_2, \ldots, t_{n-2})$. Thus the vertices of degree one in T are precisely those that do not appear in this sequence. To reconstruct T from $(t_1, t_2, \ldots, t_{n-2})$, we therefore proceed as follows. Let s_1 be the first vertex of N not in $(t_1, t_2, \ldots, t_{n-2})$; join s_1 to t_1. Next, let s_2 be the first vertex of $N \backslash \{s_1\}$ not in $(t_2, \ldots, t_{n-2})$, and join s_2 to t_2. Continue in this way until the $n-2$ edges $s_1t_1, s_2t_2, \ldots, s_{n-2}t_{n-2}$ have been determined. T is now obtained by adding the edge joining the two remaining vertices of $N \backslash \{s_1, s_2, \ldots, s_{n-2}\}$. It is easily verified that different sequences give rise to different spanning trees of K_n. We have thus established the desired one-one correspondence □

Note that n^{n-2} is not the number of nonisomorphic spanning trees of K_n, but the number of distinct spanning trees of K_n; there are just six nonisomorphic spanning trees of K_6 (see figure 2.1), whereas there are $6^4 = 1296$ distinct spanning trees of K_6.

Exercises

2.4.1 Using the recursion formula of theorem 2.8, evaluate the number of spanning trees in $K_{3,3}$.

2.4.2* A *wheel* is a graph obtained from a cycle by adding a new vertex and edges joining it to all the vertices of the cycle; the new edges are called the *spokes* of the wheel. Obtain an expression for the number of spanning trees in a wheel with n spokes.

2.4.3 Draw all sixteen spanning trees of K_4.

2.4.4 Show that if e is an edge of K_n, then $\tau(K_n - e) = (n-2)n^{n-3}$.

2.4.5 (*a*) Let H be a graph in which every two adjacent vertices are joined by k edges and let G be the underlying simple graph of H. Show that $\tau(H) = k^{\nu-1}\tau(G)$.

(*b*) Let H be the graph obtained from a graph G when each edge of G is replaced by a path of length k. Show that $\tau(H) = k^{\varepsilon-\nu+1}\tau(G)$.

(*c*) Deduce from (*b*) that $\tau(K_{2,n}) = n2^{n-1}$.

APPLICATIONS

2.5 THE CONNECTOR PROBLEM

A railway network connecting a number of towns is to be set up. Given the cost c_{ij} of constructing a direct link between towns v_i and v_j, design such a network to minimise the total cost of construction. This is known as the *connector problem*.

By regarding each town as a vertex in a weighted graph with weights $w(v_iv_j) = c_{ij}$, it is clear that this problem is just that of finding, in a weighted graph G, a connected spanning subgraph of minimum weight. Moreover, since the weights represent costs, they are certainly non-negative, and we may therefore assume that such a minimum-weight spanning subgraph is a spanning tree T of G. A minimum-weight spanning tree of a weighted graph will be called an *optimal tree*; the spanning tree indicated in the weighted graph of figure 2.10 is an optimal tree (exercise 2.5.1).

We shall now present a good algorithm for finding an optimal tree in a nontrivial weighted connected graph, thereby solving the connector problem.

Consider, first, the case when each weight $w(e) = 1$. An optimal tree is then a spanning tree with as few edges as possible. Since each spanning tree of a graph has the same number of edges (theorem 2.2), in this special case we merely need to construct some spanning tree of the graph. A simple inductive algorithm for finding such a tree is the following:

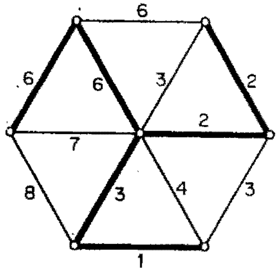


Figure 2.10. An optimal tree in a weighted graph

1. Choose a link e_1.
2. If edges $e_1, e_2, \ldots, e_i$ have been chosen, then choose e_{i+1} from $E\backslash\{e_1, e_2, \ldots, e_i\}$ in such a way that $G[\{e_1, e_2, \ldots, e_{i+1}\}]$ is acyclic.
3. Stop when step 2 cannot be implemented further.

This algorithm works because a maximal acyclic subgraph of a connected graph is necessarily a spanning tree. It was extended by Kruskal (1956) to solve the general problem; his algorithm is valid for arbitrary real weights.

Kruskal's Algorithm

1. Choose a link e_1 such that $w(e_1)$ is as small as possible.
2. If edges $e_1, e_2, \ldots, e_i$ have been chosen, then choose an edge e_{i+1} from $E\backslash\{e_1, e_2, \ldots, e_i\}$ in such a way that
   (i) $G[\{e_1, e_2, \ldots, e_{i+1}\}]$ is acyclic;
   (ii) $w(e_{i+1})$ is as small as possible subject to (i).
3. Stop when step 2 cannot be implemented further.

As an example, consider the table of airline distances in miles between six of the largest cities in the world, London, Mexico City, New York, Paris, Peking and Tokyo:

	L	MC	NY	Pa	Pe	T
L	—	5558	3469	214	5074	5959
MC	5558	—	2090	5725	7753	7035
NY	3469	2090	—	3636	6844	6757
Pa	214	5725	3636	—	5120	6053
Pe	5074	7753	6844	5120	—	1307
T	5959	7035	6757	6053	1307	—

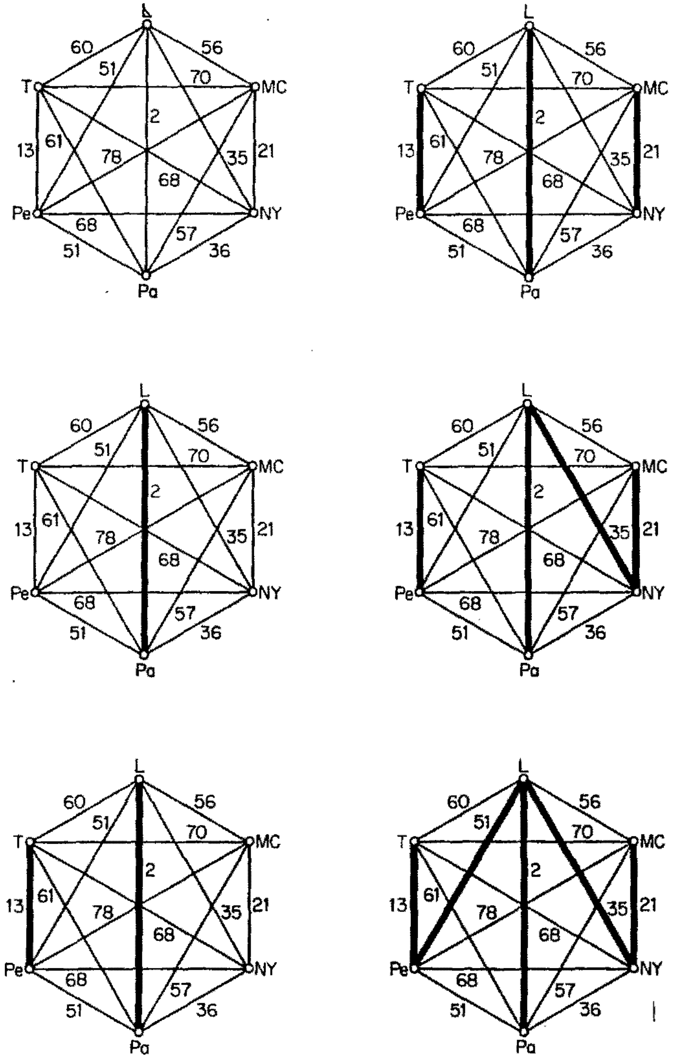


Figure 2.11

This table determines a weighted complete graph with vertices L, MC, NY, Pa, Pe and T. The construction of an optimal tree in this graph is shown in figure 2.11 (where, for convenience, distances are given in hundreds of miles).

Kruskal's algorithm clearly produces a spanning tree (for the same reason that the simpler algorithm above does). The following theorem ensures that such a tree will always be optimal.

Theorem 2.10 Any spanning tree $T^* = G[\{e_1, e_2, \ldots, e_{\nu-1}\}]$ constructed by Kruskal's algorithm is an optimal tree.

Proof By contradiction. For any spanning tree T of G other than T^*, denote by $f(T)$ the smallest value of i such that e_i is not in T. Now assume that T^* is not an optimal tree, and let T be an optimal tree such that $f(T)$ is as large as possible.

Suppose that $f(T) = k$; this means that $e_1, e_2, \ldots, e_{k-1}$ are in both T and T^*, but that e_k is not in T. By theorem 2.5, $T + e_k$ contains a unique cycle C. Let e'_k be an edge of C that is in T but not in T^*. By theorem 2.3, e'_k is not a cut edge of $T + e_k$. Hence $T' = (T + e_k) - e'_k$ is a connected graph with $\nu - 1$ edges, and therefore (exercise 2.1.5) is another spanning tree of G. Clearly

$$w(T') = w(T) + w(e_k) - w(e'_k) \qquad (2.1)$$

Now, in Kruskal's algorithm, e_k was chosen as an edge with the smallest weight such that $G[\{e_1, e_2, \ldots, e_k\}]$ was acyclic. Since $G[\{e_1, e_2, \ldots, e_{k-1}, e'_k\}]$ is a subgraph of T, it is also acyclic. We conclude that

$$w(e'_k) \geq w(e_k) \qquad (2.2)$$

Combining (2.1) and (2.2) we have

$$w(T') \leq w(T)$$

and so T', too, is an optimal tree. However

$$f(T') > k = f(T)$$

contradicting the choice of T. Therefore $T = T^*$, and T^* is indeed an optimal tree □

A flow diagram for Kruskal's algorithm is shown in figure 2.12. The edges are first sorted in order of increasing weight (box 1); this takes about $\varepsilon \log \varepsilon$ computations (see Knuth, 1973). Box 2 just checks to see how many edges have been chosen. (S is the set of edges already chosen and i is their number.) When $i = \nu - 1$, $S = \{e_1, e_2, \ldots, e_{\nu-1}\}$ is the edge set of an optimal tree T^* of G. In box 3, to check if $G[S \cup \{a_i\}]$ is acyclic, one must ascertain whether the ends of a_i are in different components of the forest $G[S]$ or not. This can be achieved in the following way. The vertices are labelled so that, at any stage, two vertices belong to the same component of $G[S]$ if and only label all vertices [illegible] [illegible] [illegible] [illegible] forming tree set [illegible] its vertices [illegible] [illegible] [illegible] find any time an edge whose vertices have same number then introduces a cycle

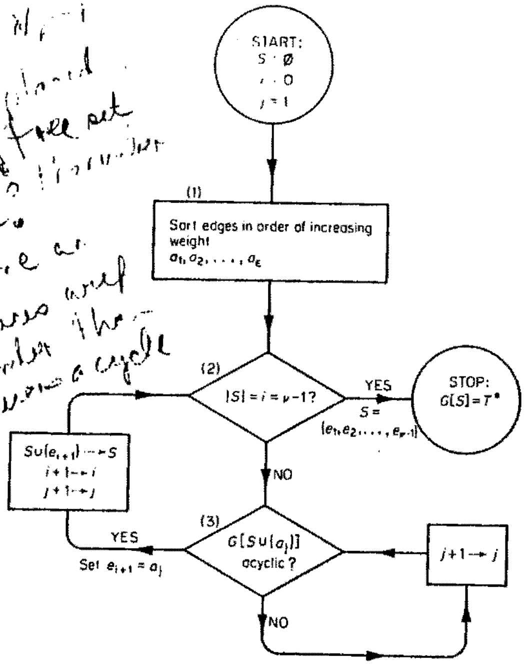

START:
S = Ø
i = 0
j = 1
(1)
Sort edges in order of increasing weight
$a_1, a_2, \ldots, a_\varepsilon$
(2)
$|S| = i = \nu-1$?
YES
$S = \{e_1, e_2, \ldots, e_{\nu-1}\}$
STOP:
$G[S] = T^*$
NO
$S \cup \{e_{i+1}\} \to S$
$i+1 \to i$
$j+1 \to j$
(3)
$G[S \cup \{a_j\}]$ acyclic?
YES
Set $e_{i+1} = a_j$
NO
$j+1 \to j$

Figure 2.12. Kruskal's algorithm

if they have the same label; initially, vertex v_l is assigned the label l, $1 \leq l \leq \nu$. With this labelling scheme, $G[S \cup \{a_j\}]$ is acyclic if and only if the ends of a_j have different labels. If this is the case, a_j is taken as e_{i+1}; otherwise, a_j is discarded and a_{j+1}, the next candidate for e_{i+1}, is tested. Once e_{i+1} has been added to S, the vertices in the two components of $G[S]$ that contain the ends of e_{i+1} are relabelled with the smaller of their two labels. For each edge, one comparison suffices to check whether its ends have the same or different labels; this takes ε computations. After edge e_{i+1} has been added to S, the relabelling of vertices takes at most ν comparisons; hence, for all $\nu-1$ edges $e_1, e_2, \ldots, e_{\nu-1}$ we need $\nu(\nu-1)$ computations. Kruskal's algorithm is therefore a good algorithm.

Exercises

2.5.1 Show, by applying Kruskal's algorithm, that the tree indicated in figure 2.10 is indeed optimal.

2.5.2 Adapt Kruskal's algorithm to solve the *connector problem with preassignments*: construct, at minimum cost, a network linking a number of towns, with the additional requirement that certain selected pairs of towns be directly linked.

2.5.3 Can Kruskal's algorithm be adapted to find

(*a*) a *maximum*-weight tree in a weighted connected graph?
(*b*) a minimum-weight maximal *forest* in a weighted graph?

If so, how?.

2.5.4 Show that the following Kruskal-type algorithm does not necessarily yield a minimum-weight spanning *path* in a weighted complete graph:

1. Choose a link e_1 such that $w(e_1)$ is as small as possible.
2. If edges $e_1, e_2, \ldots, e_i$ have been chosen, then choose an edge e_{i+1} from $E\backslash\{e_1, e_2, \ldots, e_i\}$ in such a way that
   (i) $G[\{e_1, e_2, \ldots, e_{i+1}\}]$ is a union of disjoint paths;
   (ii) $w(e_{i+1})$ is as small as possible subject to (i).
3. Stop when step 2 cannot be implemented further.

2.5.5 The *tree graph* of a connected graph G is the graph whose vertices are the spanning trees $T_1, T_2, \ldots, T_\tau$ of G, with T_i and T_j joined if and only if they have exactly $\nu-2$ edges in common. Show that the tree graph of any connected graph is connected.

REFERENCES

Cayley, A. (1889). A theorem on trees. *Quart. J. Math.*, **23**, 376–78

Knuth, D. E. (1973). *The Art of Computer Programming*, vol. 3: Sorting and Searching, Addison-Wesley, Reading, Mass., p. 184

Kruskal, J. B. Jr. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. Amer. Math. Soc.*, **7**, 48–50

Nash-Williams, C. St. J. A. (1961). Edge-disjoint spanning trees of finite graphs. *J. London Math. Soc.*, **36**, 445–50

Prüfer, H. (1918). Neuer Beweis eines Satzes über Permutationen. *Arch. Math. Phys.*, **27**, 742–44

Tutte, W. T. (1961). On the problem of decomposing a graph into *n* connected factors. *J. London Math. Soc.*, **36**, 221–30

3 Connectivity

3.1 CONNECTIVITY

In section 1.6 we introduced the concept of connection in graphs. Consider, now, the four connected graphs of figure 3.1.

G_1 is a tree, a minimal connected graph; deleting any edge disconnects it. G_2 cannot be disconnected by the deletion of a single edge, but can be disconnected by the deletion of one vertex, its cut vertex. There are no cut edges or cut vertices in G_3, but even so G_3 is clearly not as well connected as G_4, the complete graph on five vertices. Thus, intuitively, each successive graph is more strongly connected than the previous one. We shall now define two parameters of a graph, its connectivity and edge connectivity, which measure the extent to which it is connected.

A *vertex cut* of G is a subset V' of V such that $G-V'$ is disconnected. A k-*vertex cut* is a vertex cut of k elements. A complete graph has no vertex cut; in fact, the only graphs which do not have vertex cuts are those that contain complete graphs as spanning subgraphs. If G has at least one pair of distinct nonadjacent vertices, the vertex *connectivity* $\kappa(G)$ of G is the minimum k for which G has a k-vertex cut; otherwise, we define $\kappa(G)$ to be $\nu-1$. Thus $\kappa(G)=0$ if G is either trivial or disconnected. G is said to be k-*connected* if $\kappa(G)\geq k$. All nontrivial connected graphs are 1-connected.

connectivity greater or equal to K

Recall that an edge cut of G is a subset of E of the form $[S, \bar{S}]$, where S is a nonempty proper subset of V. A k-*edge cut* is an edge cut of k elements. If G is nontrivial and E' is an edge cut of G, then $G-E'$ is disconnected; we then define the *edge connectivity* $\kappa'(G)$ of G to be the minimum k for which G has a k-edge cut. If G is trivial, $\kappa'(G)$ is defined to be zero. Thus $\kappa'(G)=0$ if G is either trivial or disconnected, and $\kappa'(G)=1$ if G is a connected graph with a cut edge. G is said to be k-*edge-connected* if $\kappa'(G)\geq k$. All nontrivial connected graphs are 1-edge-connected.

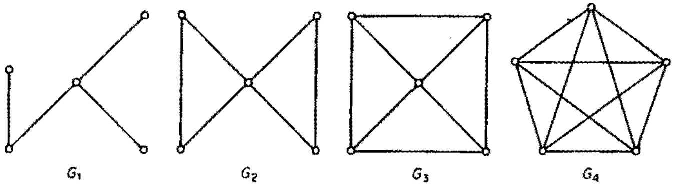


Figure 3.1

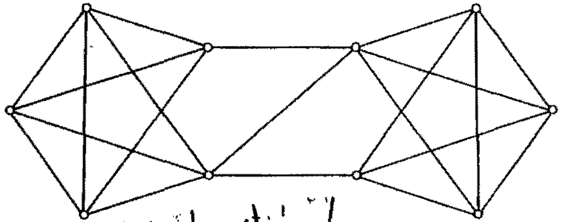


Figure 3.2

edge connectivity [illegible] vertex connectivity minimum degree

Theorem 3.1 $\kappa\leq\kappa'\leq\delta.$

Proof If G is trivial, then $\kappa'=0\leq\delta$. Otherwise, the set of links incident with a vertex of degree δ constitute a δ-edge cut of G. It follows that $\kappa'\leq\delta$.

We prove that $\kappa\leq\kappa'$ by induction on κ'. The result is true if $\kappa'=0$, since then G must be either trivial or disconnected. Suppose that it holds for all graphs with edge connectivity less than k, let G be a graph with $\kappa'(G)=k>0$, and let e be an edge in a k-edge cut of G. Setting $H=G-e$, we have $\kappa'(H)=k-1$ and so, by the induction hypothesis, $\kappa(H)\leq k-1$.

If H contains a complete graph as a spanning subgraph, then so does G and

$$\kappa(G)=\kappa(H)\leq k-1$$

Otherwise, let S be a vertex cut of H with $\kappa(H)$ elements. Since $H-S$ is disconnected, either $G-S$ is disconnected, and then

$$\kappa(G)\leq\kappa(H)\leq k-1$$

or else $G-S$ is connected and e is a cut edge of $G-S$. In this latter case, either $\nu(G-S)=2$ and

$$\kappa(G)\leq\nu(G)-1=\kappa(H)+1\leq k$$

or (exercise 2.3.1a) $G-S$ has a 1-vertex cut $\{v\}$, implying that $S\cup\{v\}$ is a vertex cut of G and

$$\kappa(G)\leq\kappa(H)+1\leq k$$

Thus in each case we have $\kappa(G)\leq k=\kappa'(G)$. The result follows by the principle of induction □

The inequalities in theorem 3.1 are often strict. For example, the graph G of figure 3.2 has $\kappa=2$, $\kappa'=3$ and $\delta=4$.

Exercises

3.1.1 (*a*) Show that if G is k-edge-connected, with $k>0$, and if E' is a set of k edges of G, then $\omega(G-E')\leq 2$.

(*b*) For $k>0$, find a k-connected graph G and a set V' of k vertices of G such that $\omega(G-V')>2$.

3.1.2 Show that if G is k-edge-connected, then $\varepsilon \geq k\nu/2$.

3.1.3 (*a*) Show that if G is simple and $\delta \geq \nu-2$, then $\kappa=\delta$.

(*b*) Find a simple graph G with $\delta=\nu-3$ and $\kappa<\delta$.

3.1.4 (*a*) Show that if G is simple and $\delta \geq \nu/2$, then $\kappa'=\delta$.

(*b*) Find a simple graph G with $\delta=[(\nu/2)-1]$ and $\kappa'<\delta$.

3.1.5 Show that if G is simple and $\delta \geq (\nu+k-2)/2$, then G is k-connected.

~~3.1.6~~ Show that if G is simple and 3-regular, then $\kappa=\kappa'$.

3.1.7 Show that if l, m and n are integers such that $0<l\leq m\leq n$, then there exists a simple graph G with $\kappa=l$, $\kappa'=m$, and $\delta=n$.

(G. Chartrand and F. Harary)

3.2 BLOCKS

2 vertex connected

A connected graph that has no cut vertices is called a *block*. Every block with at least three vertices is 2-connected. A *block of a graph* is a subgraph that is a block and is maximal with respect to this property. Every graph is the union of its blocks; this is illustrated in figure 3.3.

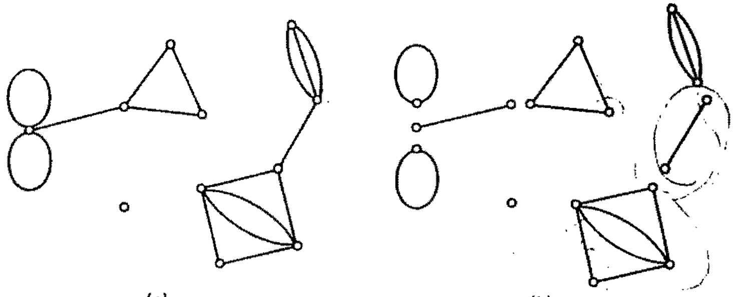


Figure 3.3. (*a*) G; (*b*) the blocks of G

A family of paths in G is said to be *internally-disjoint* if no vertex of G is an internal vertex of more than one path of the family. The following theorem is due to Whitney (1932).

Theorem 3.2 A graph G with $\nu \geq 3$ is 2-connected if and only if any two vertices of G are connected by at least two internally-disjoint paths.

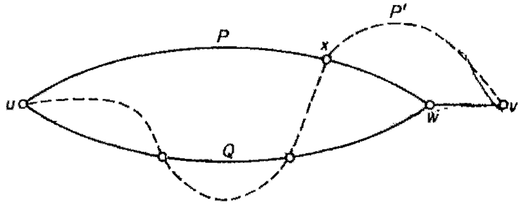


Figure 3.4

Proof If any two vertices of G are connected by at least two internally-disjoint paths then, clearly, G is connected and has no 1-vertex cut. Hence G is 2-connected.

Conversely, let G be a 2-connected graph. We shall prove, by induction on the distance $d(u, v)$ between u and v, that any two vertices u and v are connected by at least two internally-disjoint paths.

Suppose, first, that $d(u, v)=1$. Then, since G is 2-connected, the edge uv is not a cut edge and therefore, by theorem 2.3, it is contained in a cycle. It follows that u and v are connected by two internally-disjoint paths in G.

Now assume that the theorem holds for any two vertices at distance less than k, and let $d(u, v)=k\geq 2$. Consider a (u, v)-path of length k, and let w be the vertex that precedes v on this path. Since $d(u, w)=k-1$, it follows from the induction hypothesis that there are two internally-disjoint (u, w)-paths P and Q in G. Also, since G is 2-connected, $G-w$ is connected and so contains a (u, v)-path P'. Let x be the last vertex of P' that is also in $P\cup Q$ (see figure 3.4). Since u is in $P\cup Q$, there is such an x; we do not exclude the possibility that $x=v$.

We may assume, without loss of generality, that x is in P. Then G has two internally-disjoint (u, v)-paths, one composed of the section of P from u to x together with the section of P' from x to v, and the other composed of Q together with the path wv □

Corollary 3.2.1 If G is 2-connected, then any two vertices of G lie on a common cycle.

Proof This follows immediately from theorem 3.2 since two vertices lie on a common cycle if and only if they are connected by two internally-disjoint paths □

It is convenient, now, to introduce the operation of subdivision of an edge. An edge e is said to be *subdivided* when it is deleted and replaced by a path of length two connecting its ends, the internal vertex of this path being a new vertex. This is illustrated in figure 3.5.

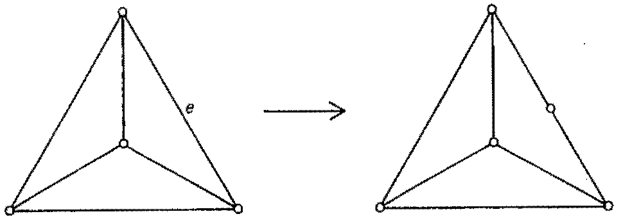


Figure 3.5. Subdivision of an edge

It can be seen that the class of blocks with at least three vertices is closed under the operation of subdivision. The proof of the next corollary uses this fact.

Corollary 3.2.2 If G is a block with $\nu \geq 3$, then any two edges of G lie on a common cycle.

Proof Let G be a block with $\nu \geq 3$, and let e_1 and e_2 be two edges of G. Form a new graph G' by subdividing e_1 and e_2, and denote the new vertices by v_1 and v_2. Clearly, G' is a block with at least five vertices, and hence is 2-connected. It follows from corollary 3.2.1 that v_1 and v_2 lie on a common cycle of G'. Thus e_1 and e_2 lie on a common cycle of G (see figure 3.6) □

Theorem 3.2 has a generalisation to k-connected graphs, known as *Menger's theorem*: a graph G with $\nu \geq k+1$ is k-connected if and only if any two distinct vertices of G are connected by at least k internally-disjoint paths. There is also an edge analogue of this theorem: a graph G is k-edge-connected if and only if any two distinct vertices of G are connected by at least k edge-disjoint paths. Proofs of these theorems will be given in chapter 11.

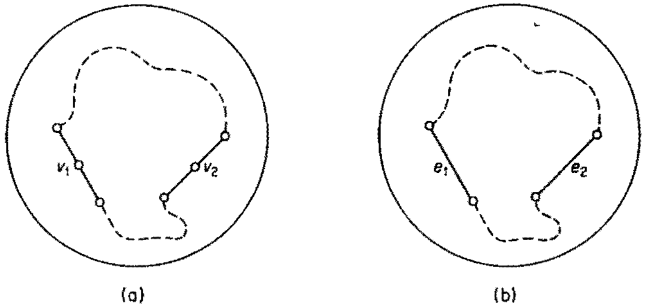


Figure 3.6. (*a*) G'; (*b*) G

Exercises

3.2.1 Show that a graph is 2-edge-connected if and only if any two vertices are connected by at least two edge-disjoint paths.

3.2.2 Give an example to show that if P is a (u, v)-path in a 2-connected graph G, then G does not necessarily contain a (u, v)-path Q internally-disjoint from P.

3.2.3 Show that if G has no even cycles, then each block of G is either K_1 or K_2, or an odd cycle.

3.2.4 Show that a connected graph which is not a block has at least two blocks that each contain exactly one cut vertex.

3.2.5 Show that the number of blocks in G is equal to $\omega + \sum_{v \in V}(b(v) - 1)$, where $b(v)$ denotes the number of blocks of G containing v.

3.2.6* Let G be a 2-connected graph and let X and Y be disjoint subsets of V, each containing at least two vertices. Show that G contains disjoint paths P and Q such that

(i) the origins of P and Q belong to X,
(ii) the termini of P and Q belong to Y, and
(iii) no internal vertex of P or Q belongs to $X \cup Y$.

3.2.7* A nonempty graph G is κ-*critical* if, for every edge e, $\kappa(G-e) < \kappa(G)$.

(*a*) Show that every κ-critical 2-connected graph has a vertex of degree two.
(Halin, 1969 has shown that, in general, every κ-critical k-connected graph has a vertex of degree k.)

(*b*) Show that if G is a κ-critical 2-connected graph with $\nu \geq 4$, then $\varepsilon \leq 2\nu - 4$. (G. A. Dirac)

3.2.8 Describe a good algorithm for finding the blocks of a graph.

APPLICATIONS

3.3 CONSTRUCTION OF RELIABLE COMMUNICATION NETWORKS

If we think of a graph as representing a communication network, the connectivity (or edge connectivity) becomes the smallest number of communication stations (or communication links) whose breakdown would jeopardise communication in the system. The higher the connectivity and edge connectivity, the more reliable the network. From this point of view, a tree network, such as the one obtained by Kruskal's algorithm, is not very reliable, and one is led to consider the following generalisation of the connector problem.

Let k be a given positive integer and let G be a weighted graph. Determine a minimum-weight k-connected spanning subgraph of G.

For $k = 1$, this problem reduces to the connector problem, which can be solved by Kruskal's algorithm. For values of k greater than one, the problem is unsolved and is known to be difficult. However, if G is a complete graph in which each edge is assigned unit weight, then the problem has a simple solution which we now present.

Observe that, for a weighted complete graph on n vertices in which each edge is assigned unit weight, a minimum-weight m-connected spanning subgraph is simply an m-connected graph on n vertices with as few edges as possible. We shall denote by $f(m, n)$ the least number of edges that an m-connected graph on n vertices can have. (It is, of course, assumed that $m < n$.) By theorems 3.1 and 1.1

$$f(m, n) \geq \{mn/2\} \tag{3.1}$$

We shall show that equality holds in (3.1) by constructing an m-connected graph $H_{m,n}$ on n vertices that has exactly $\{mn/2\}$ edges. The structure of $H_{m,n}$ depends on the parities of m and n; there are three cases.

Case 1 m even. Let $m = 2r$. Then $H_{2r,n}$ is constructed as follows. It has vertices $0, 1, \ldots, n-1$ and two vertices i and j are joined if $i - r \leq j \leq i + r$ (where addition is taken modulo n). $H_{4,8}$ is shown in figure 3.7*a*.

Case 2 m odd, n even. Let $m = 2r+1$. Then $H_{2r+1,n}$ is constructed by first drawing $H_{2r,n}$ and then adding edges joining vertex i to vertex $i + (n/2)$ for $1 \leq i \leq n/2$. $H_{5,8}$ is shown in figure 3.7*b*.

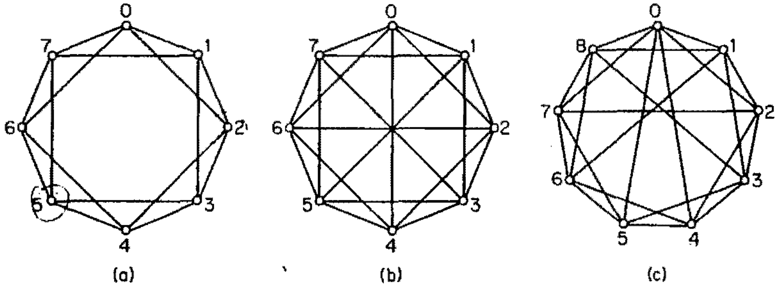


Figure 3.7. (*a*) $H_{4,8}$; (*b*) $H_{5,8}$; (*c*) $H_{5,9}$

Case 3 m odd, n odd. Let $m = 2r+1$. Then $H_{2r+1,n}$ is constructed by first drawing $H_{2r,n}$ and then adding edges joining vertex 0 to vertices $(n-1)/2$ and $(n+1)/2$ and vertex i to vertex $i + (n+1)/2$ for $1 \leq i < (n-1)/2$. $H_{5,9}$ is shown in figure 3.7c.

Theorem 3.3 (Harary, 1962) The graph $H_{m,n}$ is m-connected.

Proof Consider the case $m = 2r$. We shall show that $H_{2r,n}$ has no vertex cut of fewer than $2r$ vertices. If possible, let V' be a vertex cut with $|V'| < 2r$. Let i and j be vertices belonging to different components of $H_{2r,n} - V'$. Consider the two sets of vertices

$$S = \{i, i+1, \ldots, j-1, j\}$$

and

$$T = \{j, j+1, \ldots, i-1, i\}$$

where addition is taken modulo n. Since $|V'| < 2r$, we may assume, without loss of generality, that $|V' \cap S| < r$. Then there is clearly a sequence of distinct vertices in $S \backslash V'$ which starts with i, ends with j, and is such that the difference between any two consecutive terms is at most r. But such a sequence is an (i, j)-path in $H_{2r,n} - V'$, a contradiction. Hence $H_{2r,n}$ is $2r$-connected.

The case $m = 2r+1$ is left as an exercise (exercise 3.3.1) □

It is easy to see that $\varepsilon(H_{m,n}) = \{mn/2\}$. Thus, by theorem 3.3,

$$f(m, n) \leq \{mn/2\} \tag{3.2}$$

It now follows from (3.1) and (3.2) that

$$f(m, n) = \{mn/2\}$$

and that $H_{m,n}$ is an m-connected graph on n vertices with as few edges as possible.

We note that since, for any graph G, $\kappa \leq \kappa'$ (theorem 3.1), $H_{m,n}$ is also m-edge-connected. Thus, denoting by $g(m, n)$ the least possible number of edges in an m-edge-connected graph on n vertices, we have, for $1 < m < n$

$$g(m, n) = \{mn/2\} \tag{3.3}$$

Exercises

3.3.1 Show that $H_{2r+1,n}$ is $(2r+1)$-connected.

3.3.2 Show that $\kappa(H_{m,n}) = \kappa'(H_{m,n}) = m$.

3.3.3 Find a graph with nine vertices and 23 edges that is 5-connected but not isomorphic to the graph $H_{5,9}$ of figure 3.7c.

3.3.4 Show that (3.3) holds for all values of m and n with $m > 1$ and $n > 1$.

RELIABLE BROADCAST IN MOBILE WIRELESS NETWORKS[1]

S. Alagar and S. Venkatesan
Department of Computer Science, University of Texas at Dallas
Richardson, TX 75083

J. R. Cleveland
C3 Systems Division, Electrospace Systems, Inc., A Chrysler Company
Richardson, TX 75081

ABSTRACT

This paper presents preliminary results of our research on wireless networking that supports reliable communications between nomadic hosts engaged in distributed computing and collaborative conferencing. Our network model consists of a set of low-power, radio frequency (RF) transceivers which move relative to each other across an irregular terrain subject to RF propagation impairments. The low transmitter power defines a radio coverage which limits the probability of intercept and the number of neighbors but optimizes frequency reuse. The combination of low power and propagation environment produces a network characterized by stochastic link failures. The rapidity of these failures and perturbations to the network topology defeats the use of routing policies based on maintaining routing tables or determining least cost paths. With these conditions as the background, our work addresses the need to provide reliable information exchange, mitigate bottlenecks, avoid excessive traffic, and offer scalable services without the benefit of static base station or fixed backbone support. Meeting such challenges demands a robust, flexible information transport system that delivers all required information for diverse operational scenarios. The approach emphasizes the importance of achieving guaranteed delivery across a network of limited size operating in a hostile environment rather than obtaining a high throughput per unit area, typical of commercial enterprises.

The basic premise of the protocol is that host mobility and terrain prevents *a priori* knowledge of any host location and optimum path. Message broadcasting, or flood routing, provides the means for reliable delivery of information in the presence of uncertain connectivity and node locations. Knowledge of the network results, instead, from a measure of transmitted and received message traffic. Central to the protocol is the provision for each mobile host to retain a *HISTORY* of messages broadcast to and received from its neighbor(s). A host which receives a message broadcasts an acknowledgment to the sender, updates its local *HISTORY*, and then retransmits the message if it is not a duplicate message. Duplicated messages are discarded. If a sending host does not receive an acknowledgment from a neighbor within a certain time, it timeouts and resends the message. If a host does not receive an acknowledgment after several retries, it assumes that the link disconnection is not transient and stops sending the message. When a host detects a new neighbor, a handshake procedure results in the exchange of active messages not common to the respective *HISTORY* of each host. Once the handshake procedure terminates, the contents of the *HISTORY* for each host are identical. Thus, using handshake procedures, mobile hosts receive messages that they did not receive previously due to link disconnections. Idle hosts will periodically broadcast a sounding message to maintain their network presence.

1. INTRODUCTION. Winning the information war with complete and up-to-date intelligence is vital to the entire spectrum of possible operational requirements, whether engaged in war or corporate strategic planning. Military commanders engaged in rapid force projection, as well as public safety officers, medical staff, and corporate managers, demand accurate information regardless of location or situation. Each requires a clear and accurate picture of a changing situation to reach well-informed decisions and successful conclusion. Information must flow throughout the network toward the users at each level of the management hierarchy whether at the sustaining base or at the forward most part of the mission. Participating staff must have the capability to acquire or send accurate information that defines their space and situation. The information transport network must extend reliable voice, data, video, and imagery transmissions to nomadic users at any location. The availability of assured communications directly relates to mission success through computing, conferencing, and synchronized tasking while fixed or on-the-move. Invariably, this critical information is required when communications services normally provided by a reliable,

[1] This research was supported in part by NSF under Grant No. CRR-9110177, by the Texas Advanced Technology Program under Grant No. 9741-036, and by a grant from Electrospace Systems, Inc., a Chrysler Company.

0-7803-2489-7/95 $4.00 © 1995 IEEE

fixed infrastructure are unavailable or severely degraded.

The wireless networking of mobile (i.e., nomadic) subscribers is an emerging paradigm in the field of distributed command, control, and computing, with the potential to improve command and control responsiveness. In our context, the phrase "mobile wireless network" means that the network does not contain any static support stations. This network model supports the needs of subscribers to mobile command posts and mobile satellite ground entry points. In this role, the network must provide reliable information transfer, mitigate bottlenecks, avoid excessive traffic, offer scalable services, and, above all, adapt to dynamic topologies. The RF coverage area should conform as closely as possible to the area over which subscribers move, thereby offering a low probability of detection and improving frequency re-use. Low-cost implementation and operation are critical.

Nomadic wireless networks currently are characterized by limited bandwidth and frequent changes in link connectivity. The challenge is to allow updates from multiple users simultaneously, but only for those users that require the service. The major requirements include the capability to: minimize updates, provide fault-tolerant service, provide service scalability, and minimize communication and computation overhead. Many of the algorithms that assume static hosts or well-defined point-to-point links cannot be directly used for mobile systems due to the changes in physical connectivity and limited bandwidth of the wireless links. This has spawned considerable research in mobile computing: designing communication protocols [1, 2, 3, 4], file system operations [5, 6], managing data efficiently [7, 8], and providing fault tolerance [9]. Most research on these topics is based on a model in which the mobile hosts are supported by static base stations such as cellular telephony or personal communications systems (PCS). A typical PCS topology takes the form of a single-hop network in which each host is within radio range of the base station or all other hosts. In this paper, we consider the problem of providing reliable broadcast in mobile wireless networks where single-hop and known topologies may not exist. Applications include disaster relief operations, highly mobile military or law enforcement operations, and rapid response contingency operations where it is not economical to place support stations.

2. WIRELESS NETWORK MODEL. The model of the mobile wireless network consists of several mobile hosts distributed over an irregular terrain (Figure 1). The mobile hosts use low-power transmitters and novel, efficient receivers [10] to communicate. Emerging technologies and products for PCS applications that operate in the ISM bands with a transmitter power of 1W [11] provide the basis for practical implementation. In this network concept, the *cell* of a mobile host is the geographical area within which the mobile hosts can directly communicate with other mobile hosts. Note that the cell of a host does not remain fixed, but moves with each attached host. The nominal cell size (R) is determined by a path loss model that denotes the *local average* received signal power relative to the transmit power. A general path loss (PL) model that has been demonstrated through measurement uses a parameter $2 \le \mu \le 5$ [12] to denote the power law relationship between distance and received power. Based on both analysis and measured results, $\mu \approx 4$ for the microcell propagation environment beyond a characteristic distance R_o. The power law model takes the form [13]:

$$PL(R) = PL(R_o) + 10\mu \log(R/R_o) + X_\sigma \qquad (1)$$

where $PL(R_o)$ gives the power loss at the characteristic distance R_o and X_σ denotes a zero mean Gaussian random variable that reflects the fluctuations in average received power. Nelson and Kleinrock [14] show that for a slotted ALOHA network protocol, the optimum throughput occurs with a cell size defined by a range that includes an average of six nearest neighbors. Their results are reproduced in Figure 2. These results assume a random distribution of nodes across the terrain and a perfect capture condition. Perfect capture occurs when a node will always receive and detect the strongest of several simultaneous transmissions within its hearing range. The weaker signals appear as noise to the detection process. A non-caputre condition occurs if simultaneous transmissions always result in collisions. The reduced power supports frequency reuse, as well as low probability of detection. Their analysis also shows that mobile hosts with sufficient

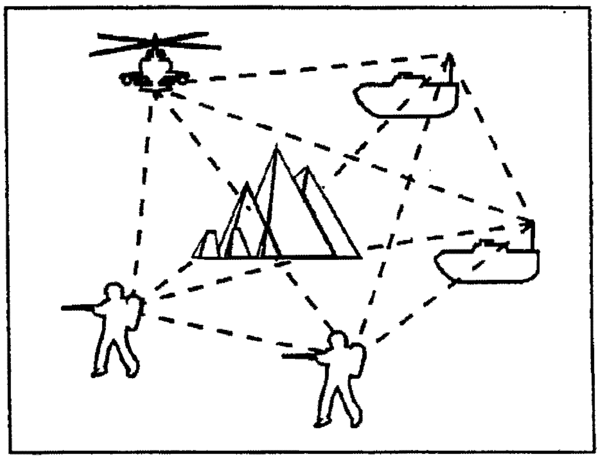


Figure 1. Model of a wireless computer network that experiences link failures due to range limitations and terrain impairments.

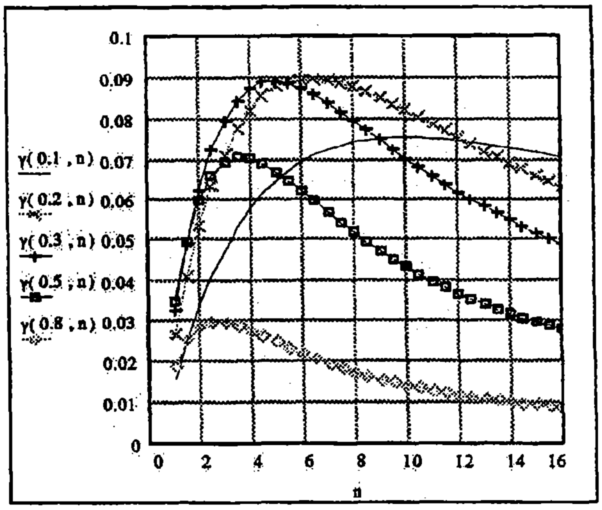


Figure 2. Normalized throughput for perfect capture with the slotted ALOHA protocol versus the average number of nearest neighbors for different probabilities that a node is busy.

transmitter power to reach all other hosts (i.e., a single-hop network) support a significantly lower throughput.

Detecting Neighbors. Neighbors are detected by a strategy common to a general class of survivable and adaptive network protocols that use sounding procedures [15, 16]. Two mobile hosts are *neighbors* if they can "hear" each other. Each host detects its neighbors by periodically broadcasting a probe message. A host that hears a probe message sends an acknowledgment to the probing host. Every host maintains a list of neighbors and periodically updates the list based on acknowledgments received. When two hosts become neighbors, a wireless link is established between them, and they execute a *handshake* procedure. As part of the handshake procedure, they each update their list of neighbors.

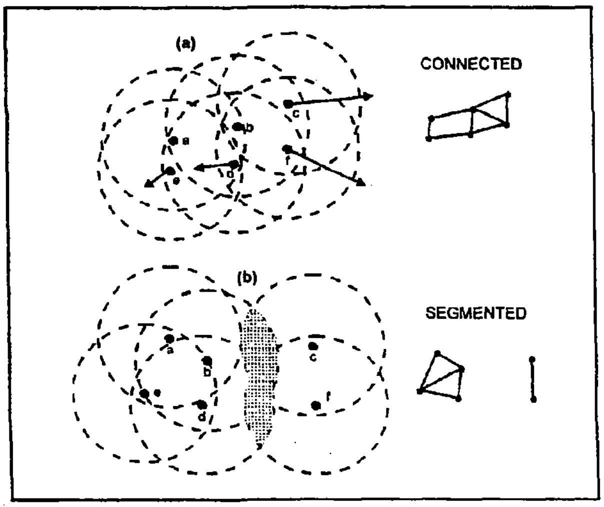


Figure 3. Examples of (a) a fully connected mobile wireless network and (b) a decomposed network due to mobility of hosts *c* and *f*. The shaded region indicates the common area within the range of hosts *c* and *f* and the rest of the network.

Link Disconnections. The wireless link between two neighbors is unreliable due to RF propagation effects such as loss of line-of-sight (LOS), moving out of range, multipath fading, or inclement weather. There are two types of link disconnections: (1) transient and (2) permanent. In the transient case, a host is unable to communicate briefly with a neighbor due to: (a) the neighbor moving out of sight; (b) multipath fading; or (c) inclement weather. Multipath fading has a time dependence that varies from microseconds to seconds, depending on the terrain and the host velocity [17]. Fade depths range up to 20 dB. The stochastic behavior of such transient link disconnections is very similar to that encountered for high frequency radio networks [18]. In the permanent case, a host is unable to communicate with a neighbor because their separation exceeds the range described by the cell geometry. We assume that each mobile host can communicate with an arbitrary mobile host in its cell without any interference (from other mobile hosts in the same cell) using techniques such as TDMA or code division multiple access (CDMA) spread spectrum signaling. One such technique is presented in [16].

3. BROADCAST CONSIDERATIONS.

Terrestrial networks provide the means to manage RF spectrum utilization to minimize the inherent latency for transmission, the probability of detection, and the cost of utilization not afforded by satellite services. Reliable broadcast in a mobile wireless network is not easy due to the following reasons: (1) It may be difficult to maintain a convenient structure (spanning tree, virtual ring) for broadcasting because of the mobility of hosts and the absence of an established backbone network. While an adaptive algorithm can be used to maintain the structure, the small cell size leads to cases where two neighbors may frequently move out of each others' range leading to the invocation of the adaptive algorithm frequently. (2) The wireless network itself may not be connected always (see Figure 3). They may be decomposed into several connected components for a while and merge after some time (we assume "quite often"). We also assume that permanent disconnections do not occur. In the next section we present a preliminary solution for reliable broadcast in mobile wireless network. Our solution is based on simple (restricted) flooding and handshaking.

Flooding ultimately involves transmitting the message to every node in the network, which is a disadvantage, particularly for large networks. The main advantage of flooding is that there is little explicit overhead and network management. As a consequence, no provisions are made to store or maintain routing or management data. Instead, hosts keep track of individual messages received and determine whether or not to retransmit the message. It is well suited to network requirements for highly mobile user groups on the digitized battlefield or in disaster relief operations where there is a need for reliable delivery in the presence of uncertain connectivity and rapid topology changes [19].

4. BROADCAST PROTOCOL. To broadcast a message, a mobile host transmits the message to all of its neighbors. On receiving a broadcast message, an intermediate mobile host retransmits the message to all of its neighbors. The technique would suffice if the network remained connected forever. Additional steps are necessary to cope with network link disconnections.

For example, mobile host h_i maintains a sequence counter, CNT$_i$. At any instant, the value of CNT$_i$ denotes the number of messages broadcast by host h_i. CNT$_i$ is incremented when h_i is about to broadcast a new message. In addition, host h_i maintains a history of messages (*HISTORY*$_i$) it has broadcast as well as received from other hosts. The j^{th} component of *HISTORY*$_i$ contains information about messages broadcast from h_i and received from h_j. A rebroadcast count and a time stamp provide the means to limit the propagation of the message in a large network.

A sample pictorial representation of *HISTORY*$_i$ is shown in Figure 4. Host h_j has received messages 1 and 4, but not messages 2 and 3. Similarly, h_k has received messages 2 and 4 from h_k, but not messages 1 and 3. It is easy to maintain *HISTORY*, for each h, as an array of lists. We now describe our solution for reliable broadcast.

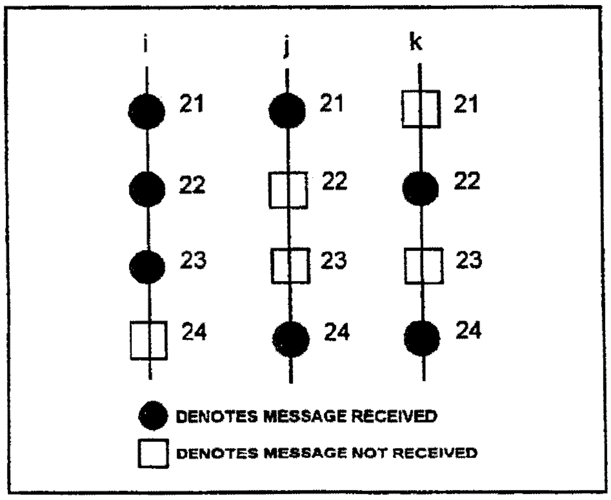


Figure 4. A snapshot of the status of the *HISTORY*$_i$ showing messages stored in h_i, h_j and h_k.

Normal Operation. Let us assume that host h_i wants to broadcast message m. The following occurs:

- Host h_i first increments CNT$_i$ and then transmits message (m, h_i, CNT$_i$) to all neighbors. It also stores (m, h_i, CNT$_i$) in a buffer locally.
- When host h_j receives message (m, h_i, CNT$_i$), it sends an acknowledgment to the sender, updates the i^{th} component of *HISTORY*$_j$ and buffers the message locally. Host$_j$ h then retransmits the message (m, h_i, CNT$_i$) to its neighbors.
- If h_j receives another copy of (m, h_i, CNT$_i$), it discards the message, but sends an acknowledgment to the sender.
- A mobile host h, after sending a message (m, h_i, CNT$_i$) to its neighbors, waits for acknowledgments from all of its neighbors. If h does not receive acknowledgment from a neighbor within a certain time, h timeouts and resends the message (with a hope that link disconnection is transient). If h does not receive acknowledgment after several retries, h assumes that the link disconnection is not transient and stops sending the message.

During periods of heavy message exchange activity, this strategy substitutes for the polling or sounding procedure described in Section 2.

Handshake Procedure. When host h_j detects a new neighbor, h_k, a handshake procedure is executed by hosts h_j and h_k:

- h_j sends *HISTORY*$_j$ to h_k and receives *HISTORY*$_k$ from h_k.
- $h_j$compares *HISTORY*$_k$ with *HISTORY*$_j$ to identify messages available in *HISTORY*$_j$, but not in *HISTORY*$_k$, and broadcasts those messages. Host h_k does likewise.
- h_k then receives the "new" messages send by h_j and updates *HISTORY*$_k$. h_j does the same for messages received from h_k. also sends these "new" messages to other neighbors.

At the conclusion of the handshake procedure, the contents of *HISTORY*$_j$ and *HISTORY*$_k$ are equal. Thus, using handshake procedure, mobile hosts receive messages that they did not receive due to link disconnections. The size of *HISTORY* stored at each h can be reduced as follows. If the first message received by h_j from h_k is CNT$_k$=t then it is sufficient for the kth component of *HISTORY*$_i$ to start from t. Storage for entry of messages 1 to t - 1 need not be provided. Further optimization can be done by storing either the *HISTORY* of messages received or the *HISTORY* messages not received depending on which list is smaller.

5. CONCLUSIONS AND FUTURE WORK. We have presented a protocol model designed to achieve assured delivery of information in a multi-hop nomadic wireless network. To mitigate a handicap of flood routing, the protocol includes a mechanism to restrict the retransmission of messages. The protocol accounts for the temporary separation of a node, or node segments, from other network members. Our continued research is devoted to methods which improve the protocol efficiency given the limitation of flood routing. In order to reduce the size of the buffer at each mobile host, a buffered message can be deleted after it is received by all the hosts. For each message a host receives, the host sends an acknowledgment to the sender of the message. Once acknowledgments from all hosts have reached the originator, the originator can direct the hosts to delete the message from the buffer [2]. To this end, we may have to broadcast and buffer acknowledgments also, which will increase the overhead. One of our objectives is to design an efficient acknowledgment policy that does not adversely increase the congestion and storage required at each host. Another option in deleting the buffered messages is to use timeouts, but this may not be suitable in critical applications where messages cannot be lost. Also the timeout period has to be chosen carefully (incorporating the mobility and link disconnections) so that the probability of message loss is very low. Some related research issues are: (1) deriving the necessary conditions, with respect to the host mobility pattern for our protocol to work; (2) identifying structures that are easy to maintain and are suitable for broadcasting; and (3) designing efficient routing schemes for unicasting messages.

We are investigating the efficiency and performance characteristics of survivable and adaptive network protocols with computer simulation techniques. Preliminary results will be reported on the evaluation of the algorithm in terms of message delay and acknowledgment overhead for different network sizes and routing restrictions. Our preliminary results indicate the viability of the message management protocol for collaborative computing in a dynamic computer network topology when the reliability of information is paramount.

References

[1] Alagar, S., and Venkatesan, S. Casual ordering in distributed mobile systems. (To appear in Proc. of Workshop on Mobile Systems and Applications, Dec., 1994.)

[2] Badrinath, B., and Acharya, A. Delivering multicast messages in networks with mobile hosts. In *Proceedings of the 13th International Conference on Distributed Computing Systems (1993)*, IEEE, pp. 292-299.

[3] Ioannidis, J., Duchamp, D., and Maguire, G. IP-based protocols for mobile internetworking. In *Proceedings of ACM SIGCOMM Symposium on Communication Architecture and Protocols* (1991), pp. 235-245.

[4] Teraoka, F., Yokote, Y., and Tokoro, M. A network architecture providing host migration transparency. In *Proceedings of ACM SIGCOMM* (September 1991).

[5] Kistler, J., and Satyanarayana, M. Disconnected operation in coda file system. *ACM Trans. Comput. Syst. 10,* 1 (February 1992).

[6] Tait, C. D., and Duchamp, D. Service interface and replica management algorithm for mobile file system clients. In *Proceedings of the 1st International Conference on Parallel and Distributed Information Systems* (1991).

[7] Alonso. R., and Korth, H. Database system issues in nomadic computing. Technical report, MITL, December 1992.

[8] Imielinski, T., and Badrinath, B. Data management for mobile computing. *ACM SIGMOD Record* (March 1993).

[9] Krishna, P., Vaidya, N., and Pradhan, D. Recovery in distributed mobile environments. In *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems* (1993), pp. 83-88.

[10] Darrell, Ash and Coon, Allan, "A Micropower SAW-Stabilized Superregenerative Data Receiver," Application Note 25, RF Monolithics, Inc., July 1992.

[11] Padgett, J. E., Gunther, C. G., and Hattori, T. Overview of wireless personnel communications. In *IEEE Commun. Mag.*, vol 33, no. 1, pp. 28-41, January, 1995.

[12] Sousa, E. S., Silvester, J., and Papavassiliou, T. D. Computer-aided modeling of spread spectrum packet radio networks. In *IEEE J. Selected Areas Commun.*, vol. 9, pp. 48-58, 1991.

[13] Andersen, J. B., Rappaport, T. S., and Yoshida, S. "Propagation Measurements and Models for Wireless Communications Channels," *IEEE Commun. Mag.* vol. 33, no. 1, pp. 42-49, January, 1995.

[14] Nelson, R. D. and Kleinrock, L. The spatial capacity of a slotted ALOHA multihop packet radio network with capture. In *IEEE Trans. Commun.*, vol. COM-32, no. 6, 1984, pp. 684-694.

[15] FED-STD-1045, Telecommunications: HF Radio Automatic Link Establishment, from Institute for Telecommunication Sciences, National Telecommunications and Information Administration, US Dept. of Commerce, 24 Jan 1990.

[16] Baker, D, J., Ephremides, A., and Flynn, J. A. The design and simulation of a mobile radio network with distributed control. In *IEEE Journal on Selected Areas in Communications SAC-2,1* (January 1984), 226-237.

[17] Greenstein, L. J., et. al. Microcells in personal communications systems. In *IEEE Commun. Mag.*, vol. 30, no. 12, 76-88, Dec., 1992.

[18] Maslin, N. M. Modeling in the hf network design process. In Fourth International Conference on HF Radio Systems and Techniques. (London, UK, 90-94, April, 1988).

[19] Leiner, B. M., *et. al.*, Issues in Packet Radio Network Design. In *Proc. IEEE* vol. 75, no. 1, pp. 6-20, January, 1987.

Internet RFC/STD/FYI/BCP Archives

RFC1832

[Index | Search | What's New | Comments | Help]

Network Working Group
Request for Comments: 1832
Category: Standards Track

R. Srinivasan
Sun Microsystems
August 1995

XDR: External Data Representation Standard

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

ABSTRACT

This document describes the External Data Representation Standard (XDR) protocol as it is currently deployed and accepted.

TABLE OF CONTENTS

1. INTRODUCTION

XDR is a standard for the description and encoding of data. It is useful for transferring data between different computer architectures, and has been used to communicate data between such diverse machines as the SUN WORKSTATION*, VAX*, IBM-PC*, and Cray*. XDR fits into the ISO presentation layer, and is roughly analogous in purpose to X.409, ISO Abstract Syntax Notation. The major difference between these two is that XDR uses implicit typing, while X.409 uses explicit typing.

XDR uses a language to describe data formats. The language can only be used only to describe data; it is not a programming language. This language allows one to describe intricate data formats in a concise manner. The alternative of using graphical representations (itself an informal language) quickly becomes incomprehensible when faced with complexity. The XDR language itself is similar to the C language [1], just as Courier [4] is similar to Mesa. Protocols such as ONC RPC (Remote Procedure Call) and the NFS* (Network File System) use XDR to describe the format of their data.

The XDR standard makes the following assumption: that bytes (or octets) are portable, where a byte is defined to be 8 bits of data. A given hardware device should encode the bytes onto the various media in such a way that other hardware devices may decode the bytes without loss of meaning. For example, the Ethernet* standard suggests that bytes be encoded in "little-endian" style [2], or least significant bit first.

2. BASIC BLOCK SIZE

The representation of all items requires a multiple of four bytes (or 32 bits) of data. The bytes are numbered 0 through n-1. The bytes are read or written to some byte stream such that byte m always precedes byte m+1. If the n bytes needed to contain the data are not a multiple of four, then the n bytes are followed by enough (0 to 3) residual zero bytes, r, to make the total byte count a multiple of 4.

We include the familiar graphic box notation for illustration and comparison. In most illustrations, each box (delimited by a plus sign at the 4 corners and vertical bars and dashes) depicts a byte.

Ellipses (...) between boxes show zero or more additional bytes where required.

```
        +--------+--------+...+--------+--------+...+--------+
        | byte 0 | byte 1 |...|byte n-1|    0   |...|    0   |   BLOCK
        +--------+--------+...+--------+--------+...+--------+
        |<-----------n bytes---------->|<------r bytes------>|
        |<-----------n+r (where (n+r) mod 4 = 0)>----------->|
```

3. XDR DATA TYPES

Each of the sections that follow describes a data type defined in the XDR standard, shows how it is declared in the language, and includes a graphic illustration of its encoding.

For each data type in the language we show a general paradigm declaration. Note that angle brackets (< and >) denote variablelength sequences of data and square brackets ([and]) denote fixed-length sequences of data. "n", "m" and "r" denote integers. For the full language specification and more formal definitions of terms such as "identifier" and "declaration", refer to section 5: "The XDR Language Specification".

For some data types, more specific examples are included. A more extensive example of a data description is in section 6: "An Example of an XDR Data Description".

3.1 Integer

An XDR signed integer is a 32-bit datum that encodes an integer in the range [-2147483648,2147483647]. The integer is represented in two's complement notation. The most and least significant bytes are 0 and 3, respectively. Integers are declared as follows:

```
      int identifier;

        (MSB)                   (LSB)
      +-------+-------+-------+-------+
      |byte 0 |byte 1 |byte 2 |byte 3 |                      INTEGER
      +-------+-------+-------+-------+
      <------------32 bits------------>
```

3.2. Unsigned Integer

An XDR unsigned integer is a 32-bit datum that encodes a nonnegative integer in the range [0,4294967295]. It is represented by an unsigned binary number whose most and least significant bytes are 0 and 3, respectively. An unsigned integer is declared as follows:

```
      unsigned int identifier;

        (MSB)                   (LSB)
         +-------+-------+-------+-------+
         |byte 0 |byte 1 |byte 2 |byte 3 |           UNSIGNED INTEGER
         +-------+-------+-------+-------+
         <------------32 bits------------>
```

3.3 Enumeration

Enumerations have the same representation as signed integers. Enumerations are handy for describing subsets of the integers. Enumerated data is declared as follows:

```
      enum { name-identifier = constant, ... } identifier;
```

For example, the three colors red, yellow, and blue could be described by an enumerated type:

```
      enum { RED = 2, YELLOW = 3, BLUE = 5 } colors;
```

It is an error to encode as an enum any other integer than those that have been given assignments in the enum declaration.

3.4 Boolean

Booleans are important enough and occur frequently enough to warrant their own explicit type in the standard. Booleans are declared as follows:

```
      bool identifier;
```

This is equivalent to:

```
      enum { FALSE = 0, TRUE = 1 } identifier;
```

3.5 Hyper Integer and Unsigned Hyper Integer

The standard also defines 64-bit (8-byte) numbers called hyper integer and unsigned hyper integer. Their representations are the obvious extensions of integer and unsigned integer defined above.

They are represented in two's complement notation. The most and least significant bytes are 0 and 7, respectively. Their declarations:

```
hyper identifier; unsigned hyper identifier;

      (MSB)                                                   (LSB)
    +-------+-------+-------+-------+-------+-------+-------+-------+
    |byte 0 |byte 1 |byte 2 |byte 3 |byte 4 |byte 5 |byte 6 |byte 7 |
    +-------+-------+-------+-------+-------+-------+-------+-------+
    <----------------------------64 bits---------------------------->
                                               HYPER INTEGER
                                               UNSIGNED HYPER INTEGER
```

3.6 Floating-point

The standard defines the floating-point data type "float" (32 bits or 4 bytes). The encoding used is the IEEE standard for normalized single-precision floating-point numbers [3]. The following three fields describe the single-precision floating-point number:

S: The sign of the number. Values 0 and 1 represent positive and negative, respectively. One bit.

E: The exponent of the number, base 2. 8 bits are devoted to this field. The exponent is biased by 127.

F: The fractional part of the number's mantissa, base 2. 23 bits are devoted to this field.

Therefore, the floating-point number is described by:

```
      (-1)**S * 2**(E-Bias) * 1.F
```

It is declared as follows:

```
      float identifier;
```

```
      +-------+-------+-------+-------+
      |byte 0 |byte 1 |byte 2 |byte 3 |              SINGLE-PRECISION
      S|   E   |           F          |         FLOATING-POINT NUMBER
      +-------+-------+-------+-------+
      1|<- 8 ->|<-------23 bits------>|
      <------------32 bits------------>
```

Just as the most and least significant bytes of a number are 0 and 3, the most and least significant bits of a single-precision floating-point number are 0 and 31. The beginning bit (and most significant bit) offsets of S, E, and F are 0, 1, and 9, respectively. Note that these numbers refer to the mathematical positions of the bits, and NOT to their actual physical locations (which vary from medium to medium).

The IEEE specifications should be consulted concerning the encoding for signed zero, signed infinity (overflow), and denormalized numbers (underflow) [3]. According to IEEE specifications, the "NaN" (not a number) is system dependent and should not be interpreted within XDR as anything other than "NaN".

3.7 Double-precision Floating-point

The standard defines the encoding for the double-precision floating-point data type "double" (64 bits or 8 bytes). The encoding used is the IEEE standard for normalized double-precision floating-point numbers [3]. The standard encodes the following three fields, which describe the double-precision floating-point number:

S: The sign of the number. Values 0 and 1 represent positive and negative, respectively. One bit.

E: The exponent of the number, base 2. 11 bits are devoted to this field. The exponent is biased by 1023.

F: The fractional part of the number's mantissa, base 2. 52 bits are devoted to this field.

Therefore, the floating-point number is described by:

```
      (-1)**S * 2**(E-Bias) * 1.F
```

It is declared as follows:

```
      double identifier;

      +------+------+------+------+------+------+------+------+
      |byte 0|byte 1|byte 2|byte 3|byte 4|byte 5|byte 6|byte 7|
      S|    E   |                 F                          |
      +------+------+------+------+------+------+------+------+
      1|<--11-->|<-----------------52 bits------------------->|
      <-----------------------64 bits------------------------->
                                 DOUBLE-PRECISION FLOATING-POINT
```

Just as the most and least significant bytes of a number are 0 and 3, the most and least significant bits of a double-precision floating-point number are 0 and 63. The beginning bit (and most significant bit) offsets of S, E , and F are 0, 1, and 12, respectively. Note

that these numbers refer to the mathematical positions of the bits, and NOT to their actual physical locations (which vary from medium to medium).

The IEEE specifications should be consulted concerning the encoding for signed zero, signed infinity (overflow), and denormalized numbers (underflow) [3]. According to IEEE specifications, the "NaN" (not a number) is system dependent and should not be interpreted within XDR as anything other than "NaN".

3.8 Quadruple-precision Floating-point

The standard defines the encoding for the quadruple-precision floating-point data type "quadruple" (128 bits or 16 bytes). The encoding used is designed to be a simple analog of of the encoding used for single and double-precision floating-point numbers using one form of IEEE double extended precision. The standard encodes the following three fields, which describe the quadruple-precision floating-point number:

S: The sign of the number. Values 0 and 1 represent positive and negative, respectively. One bit.

E: The exponent of the number, base 2. 15 bits are devoted to this field. The exponent is biased by 16383.

F: The fractional part of the number's mantissa, base 2. 112 bits are devoted to this field.

Therefore, the floating-point number is described by:

```
      (-1)**S * 2**(E-Bias) * 1.F
```

It is declared as follows:

```
      quadruple identifier;

      +------+-------------+------+------+------+-...--+------+
      |byte 0|byte 1|byte 2|byte 3|byte 4|byte 5| ...  |byte15|
S'    E       !                  F                      ¦
+------+------+------+------+------+------+-...--+------+
1|<----15---->|<-------------112 bits------------------>|
<-----------------------128 bits------------------------>
                             QUADRUPLE-PRECISION FLOATING-POINT
```

Just as the most and least significant bytes of a number are 0 and 3, the most and least significant bits of a quadruple-precision floating-point number are 0 and 127. The beginning bit (and most significant bit) offsets of S, E , and F are 0, 1, and 16, respectively. Note that these numbers refer to the mathematical positions of the bits, and NOT to their actual physical locations (which vary from medium to medium).

The encoding for signed zero, signed infinity (overflow), and denormalized numbers are analogs of the corresponding encodings for single and double-precision floating-point numbers [5], [6]. The "NaN" encoding as it applies to quadruple-precision floating-point numbers is system dependent and should not be interpreted within XDR as anything other than "NaN".

3.9 Fixed-length Opaque Data

At times, fixed-length uninterpreted data needs to be passed among machines. This data is called "opaque" and is declared as follows:

```
opaque identifier[n];
```

where the constant n is the (static) number of bytes necessary to contain the opaque data. If n is not a multiple of four, then the n bytes are followed by enough (0 to 3) residual zero bytes, r, to make the total byte count of the opaque object a multiple of four.

```
    0        1     ...
+--------+--------+...+--------+--------+...+--------+
| byte 0 | byte 1 |...|byte n-1|    0   |...|    0   |
+--------+--------+...+--------+--------+...+--------+
|<-----------n bytes---------->|<------r bytes------>|
|<-----------n+r (where (n+r) mod 4 = 0)------------>|
                                     FIXED-LENGTH OPAQUE
```

3.10 Variable-length Opaque Data

The standard also provides for variable-length (counted) opaque data, defined as a sequence of n (numbered 0 through n-1) arbitrary bytes to be the number n encoded as an unsigned integer (as described below), and followed by the n bytes of the sequence.

Byte m of the sequence always precedes byte m+1 of the sequence, and byte 0 of the sequence always follows the sequence's length (count). If n is not a multiple of four, then the n bytes are followed by enough (0 to 3) residual zero bytes, r, to make the total byte count a multiple of four. Variable-length opaque data is declared in the following way:

```
opaque identifier<m>;
```

```
   or
      opaque identifier<>;
```

The constant m denotes an upper bound of the number of bytes that the sequence may contain. If m is not specified, as in the second declaration, it is assumed to be (2**32) - 1, the maximum length. The constant m would normally be found in a protocol specification. For example, a filing protocol may state that the maximum data transfer size is 8192 bytes, as follows:

```
      opaque filedata<8192>;

         0     1     2     3     4     5   ...
      +-----+-----+-----+-----+-----+-----+...+-----+-----+...+-----+
      |        length n       |byte0|byte1|...| n-1 |  0  |...|  0  |
      +-----+-----+-----+-----+-----+-----+...+-----+-----+...+-----+
      |<-------4 bytes------->|<------n bytes------>|<---r bytes--->|
                              |<----n+r (where (n+r) mod 4 = 0)---->|
                                               VARIABLE-LENGTH OPAQUE
```

It is an error to encode a length greater than the maximum described in the specification.

3.11 String

The standard defines a string of n (numbered 0 through n-1) ASCII bytes to be the number n encoded as an unsigned integer (as described above), and followed by the n bytes of the string. Byte m of the string always precedes byte m+1 of the string, and byte 0 of the string always follows the string's length. If n is not a multiple of four, then the n bytes are followed by enough (0 to 3) residual zero bytes, r, to make the total byte count a multiple of four. Counted byte strings are declared as follows:

```
      string object<m>;
   or
      string object<>;
```

The constant m denotes an upper bound of the number of bytes that a string may contain. If m is not specified, as in the second

declaration, it is assumed to be (2**32) - 1, the maximum length. The constant m would normally be found in a protocol specification. For example, a filing protocol may state that a file name can be no longer than 255 bytes, as follows:

```
      string filename<255>;

         0     1     2     3     4     5   ...
      +-----+-----+-----+-----+-----+-----+...+-----+-----+...+-----+
      |        length n       |byte0|byte1|...| n-1 |  0  |...|  0  |
      +-----+-----+-----+-----+-----+-----+...+-----+-----+...+-----+
      |<-------4 bytes------->|<------n bytes------>|<---r bytes--->|
                              |<----n+r (where (n+r) mod 4 = 0)---->|
                                                               STRING
```

It is an error to encode a length greater than the maximum described in the specification.

3.12 Fixed-length Array

Declarations for fixed-length arrays of homogeneous elements are in the following form:

```
      type-name identifier[n];
```

Fixed-length arrays of elements numbered 0 through n-1 are encoded by individually encoding the elements of the array in their natural order, 0 through n-1. Each element's size is a multiple of four bytes. Though all elements are of the same type, the elements may have different sizes. For example, in a fixed-length array of strings, all elements are of type "string", yet each element will vary in its length.

```
      +---+---+---+---+---+---+---+---+...+---+---+---+---+
      |   element 0   |   element 1   |...|  element n-1  |
      +---+---+---+---+---+---+---+---+...+---+---+---+---+
      |<--------------------n elements------------------->|

                                               FIXED-LENGTH ARRAY
```

3.13 Variable-length Array

Counted arrays provide the ability to encode variable-length arrays of homogeneous elements. The array is encoded as the element count n (an unsigned integer) followed by the encoding of each of the array's elements, starting with element 0 and progressing through element n- 1. The declaration for variable-length arrays follows this form:

```
      type-name identifier<m>;
   or
      type-name identifier<>;
```

The constant m specifies the maximum acceptable element count of an array; if m is not specified, as in the second declaration, it is assumed to be (2**32) - 1.

```
        0  1  2  3
      +--+--+--+--+--+--+--+--+--+--+--+--+...+--+--+--+--+
      |     n     | element 0 | element 1 |...|element n-1|
      +--+--+--+--+--+--+--+--+--+--+--+--+...+--+--+--+--+
      |<-4 bytes->|<--------------n elements------------->|
                                                    COUNTED ARRAY
```

It is an error to encode a value of n that is greater than the maximum described in the specification.

3.14 Structure

Structures are declared as follows:

```
      struct {
         component-declaration-A;
         component-declaration-B;
```

```
            ...
         } identifier;
```

The components of the structure are encoded in the order of their declaration in the structure. Each component's size is a multiple of four bytes, though the components may be different sizes.

```
         +-------------+-------------+...
         | component A | component B |...                      STRUCTURE
         +-------------+-------------+...
```

3.15 Discriminated Union

A discriminated union is a type composed of a discriminant followed by a type selected from a set of prearranged types according to the value of the discriminant. The type of discriminant is either "int", "unsigned int", or an enumerated type, such as "bool". The component types are called "arms" of the union, and are preceded by the value of the discriminant which implies their encoding. Discriminated unions are declared as follows:

```
         union switch (discriminant-declaration) {
         case discriminant-value-A:

            arm-declaration-A;
         case discriminant-value-B:
            arm-declaration-B;
         ...
         default: default-declaration;
         } identifier;
```

Each "case" keyword is followed by a legal value of the discriminant. The default arm is optional. If it is not specified, then a valid encoding of the union cannot take on unspecified discriminant values. The size of the implied arm is always a multiple of four bytes.

The discriminated union is encoded as its discriminant followed by the encoding of the implied arm.

```
           0   1   2   3
         +---+---+---+---+---+---+---+---+
         |  discriminant |  implied arm  |          DISCRIMINATED UNION
         +---+---+---+---+---+---+---+---+
         |<---4 bytes--->|
```

3.16 Void

An XDR void is a 0-byte quantity. Voids are useful for describing operations that take no data as input or no data as output. They are also useful in unions, where some arms may contain data and others do not. The declaration is simply as follows:

```
         void;
```

Voids are illustrated as follows:

```
           ++
      | |                                                 VOID
      ++
    --><-- 0 bytes
```

3.17 Constant

The data declaration for a constant follows this form:

```
      const name-identifier = n;
```

"const" is used to define a symbolic name for a constant; it does not declare any data. The symbolic constant may be used anywhere a regular constant may be used. For example, the following defines a symbolic constant DOZEN, equal to 12.

```
      const DOZEN = 12;
```

3.18 Typedef

"typedef" does not declare any data either, but serves to define new identifiers for declaring data. The syntax is:

```
      typedef declaration;
```

The new type name is actually the variable name in the declaration part of the typedef. For example, the following defines a new type called "eggbox" using an existing type called "egg":

```
      typedef egg eggbox[DOZEN];
```

Variables declared using the new type name have the same type as the new type name would have in the typedef, if it was considered a variable. For example, the following two declarations are equivalent in declaring the variable "fresheggs":

```
      eggbox  fresheggs; egg     fresheggs[DOZEN];
```

When a typedef involves a struct, enum, or union definition, there is another (preferred) syntax that may be used to define the same type. In general, a typedef of the following form:

```
      typedef <<struct, union, or enum definition>> identifier;
```

may be converted to the alternative form by removing the "typedef" part and placing the identifier after the "struct", "union", or "enum" keyword, instead of at the end. For example, here are the two ways to define the type "bool":

```
      typedef enum {    /* using typedef */
         FALSE = 0,
         TRUE = 1
      } bool;

      enum bool {       /* preferred alternative */
         FALSE = 0,
         TRUE = 1
      };
```

The reason this syntax is preferred is one does not have to wait until the end of a declaration to figure out the name of the new type.

3.19 Optional-data

Optional-data is one kind of union that occurs so frequently that we give it a special syntax of its own for declaring it. It is declared as follows:

```
type-name *identifier;
```

This is equivalent to the following union:

```
union switch (bool opted) {
case TRUE:
   type-name element;
case FALSE:
   void;
} identifier;
```

It is also equivalent to the following variable-length array declaration, since the boolean "opted" can be interpreted as the length of the array:

```
type-name identifier<1>;
```

Optional-data is not so interesting in itself, but it is very useful for describing recursive data-structures such as linked-lists and trees. For example, the following defines a type "stringlist" that encodes lists of arbitrary length strings:

```
struct *stringlist {
   string item<>;
   stringlist next;
};
```

It could have been equivalently declared as the following union:

```
union stringlist switch (bool opted) {
case TRUE:
   struct {
      string item<>;
      stringlist next;
   } element;
case FALSE:
   void;
};
```

or as a variable-length array:

```
struct stringlist<1> {

   string item<>;
   stringlist next;
};
```

Both of these declarations obscure the intention of the stringlist type, so the optional-data declaration is preferred over both of them. The optional-data type also has a close correlation to how recursive data structures are represented in high-level languages such as Pascal or C by use of pointers. In fact, the syntax is the same as that of the C language for pointers.

3.20 Areas for Future Enhancement

The XDR standard lacks representations for bit fields and bitmaps, since the standard is based on bytes. Also missing are packed (or binary-coded) decimals.

The intent of the XDR standard was not to describe every kind of data that people have ever sent or will ever want to send from machine to machine. Rather, it only describes the most commonly used data-types of high-level languages such as Pascal or C so that applications written in these languages will be able to communicate easily over some medium.

One could imagine extensions to XDR that would let it describe almost any existing protocol, such as TCP. The minimum necessary for this are support for different block sizes and byte-orders. The XDR discussed here could then be considered the 4-byte big-endian member of a larger XDR family.

4. DISCUSSION

(1) Why use a language for describing data? What's wrong with diagrams?

There are many advantages in using a data-description language such as XDR versus using diagrams. Languages are more formal than diagrams and lead to less ambiguous descriptions of data. Languages are also easier to understand and allow one to think of other issues instead of the low-level details of bit-encoding. Also, there is a close analogy between the types of XDR and a high-level language such as C or Pascal. This makes the implementation of XDR encoding and decoding modules an easier task. Finally, the language specification itself is an ASCII string that can be passed from machine to machine to perform on-the-fly data interpretation.

(2) Why is there only one byte-order for an XDR unit?

Supporting two byte-orderings requires a higher level protocol for determining in which byte-order the data is encoded. Since XDR is not a protocol, this can't be done. The advantage of this, though, is that data in XDR format can be written to a magnetic tape, for example, and any machine will be able to interpret it, since no higher level protocol is necessary for determining the byte-order.

(3) Why is the XDR byte-order big-endian instead of little-endian? Isn't this unfair to little-endian machines such as the VAX(r), which has to convert from one form to the other?

Yes, it is unfair, but having only one byte-order means you have to be unfair to somebody. Many architectures, such as the Motorola 68000* and IBM 370*, support the big-endian byte-order.

(4) Why is the XDR unit four bytes wide?

There is a tradeoff in choosing the XDR unit size. Choosing a small size such as two makes the encoded data small, but causes alignment problems for machines that aren't aligned on these boundaries. A large size such as eight means the data will be aligned on virtually every machine, but causes the encoded data to grow too big. We chose four as a compromise. Four is big enough to support most architectures efficiently, except for rare machines such as the eight-byte aligned Cray*. Four is also small enough to keep the encoded data restricted to a reasonable size.

(5) Why must variable-length data be padded with zeros?

It is desirable that the same data encode into the same thing on all machines, so that encoded data can be meaningfully compared or checksummed. Forcing the padded bytes to be zero ensures this.

(6) Why is there no explicit data-typing?

Data-typing has a relatively high cost for what small advanta[illegible] t may have. One cost is the expansion of data due to the inser[illegible] re fields. Another is the added cost of interpreting these type [illegible] s and acting accordingly. And most protocols already know what type they expect, so data-typing supplies only redundant information. However, one can still get the benefits of data-typing using XDR. One way is to encode two things: first a string which is the XDR data description of the encoded data, and then the encoded data itself. Another way is to assign a value to all the types in XDR, and then define a universal type which takes this value as its discriminant and for each value, describes the corresponding data type.

5. THE XDR LANGUAGE SPECIFICATION

5.1 Notational Conventions

This specification uses an extended Back-Naur Form notation for describing the XDR language. Here is a brief description of the notation:

(1) The characters '|', '(', ')', '[', ']', '"', and '*' are special. (2) Terminal symbols are strings of any characters surrounded by double quotes. (3) Non-terminal symbols are strings of non-special characters. (4) Alternative items are separated by a vertical bar ("|"). (5) Optional items are enclosed in brackets. (6) Items are grouped together by enclosing them in parentheses. (7) A '*' following an item means 0 or more occurrences of that item.

For example, consider the following pattern:

```
      "a " "very" (", " "very")* [" cold " "and "]  " rainy "
      ("day" | "night")
```

An infinite number of strings match this pattern. A few of them are:

```
      "a very rainy day"
      "a very, very rainy day"
      "a very cold and  rainy day"
      "a very, very, very cold and  rainy night"
```

5.2 Lexical Notes

(1) Comments begin with '/*' and terminate with '*/'. (2) White space serves to separate items and is otherwise ignored. (3) An identifier is a letter followed by an optional sequence of letters, digits or underbar ('_'). The case of identifiers is not ignored. (4) A constant is a sequence of one or more decimal digits, optionally preceded by a minus-sign ('-').

5.3 Syntax Information

```
   declaration:
        type-specifier identifier
      | type-specifier identifier "[" value "]"
      | type-specifier identifier "<" [ value ] ">"
      | "opaque" identifier "[" value "]"
      | "opaque" identifier "<" [ value ] ">"
      | "string" identifier "<" [ value ] ">"
      | type-specifier "*" identifier
      | "void"

   value:
        constant
      | identifier

   type-specifier:
        [ "unsigned" ] "int"
      | [ "unsigned" ] "hyper"
      | "float"
      | "double"
      | "quadruple"
      | "bool"
      | enum-type-spec
      | struct-type-spec
      | union-type-spec
      | identifier

   enum-type-spec:
      "enum" enum-body

   enum-body:
      "{"
         ( identifier "=" value )
         ( "," identifier "=" value )*
      "}"

   struct-type-spec:
      "struct" struct-body

   struct-body:
      "{"
```

```
            ( declaration ";" )
            ( declaration ";" )*
         "}"

      union-type-spec:
         "union" union-body

      union-body:
         "switch" "(" declaration ")" "{"
            ( "case" value ":" declaration ";" )
            ( "case" value ":" declaration ";" )*
            [ "default" ":" declaration ";" ]
         "}"

      constant-def:
         "const" identifier "=" constant ";"

      type-def:
           "typedef" declaration ";"
         | "enum" identifier enum-body ";"
         | "struct" identifier struct-body ";"
         | "union" identifier union-body ";"

      definition:
           type-def
         | constant-def

      specification:
           definition *
```

5.4 Syntax Notes

(1) The following are keywords and cannot be used as identifiers: "bool", "case", "const", "default", "double", "quadruple", "enum", "float", "hyper", "opaque", "string", "struct", "switch", "typedef", "union", "unsigned" and "void".

(2) Only unsigned constants may be used as size specifications for arrays. If an identifier is used, it must have been declared previously as an unsigned constant in a "const" definition.

(3) Constant and type identifiers within the scope of a specification are in the same name space and must be declared uniquely within this scope.

(4) Similarly, variable names must be unique within the scope of struct and union declarations. Nested struct and union declarations create new scopes.

(5) The discriminant of a union must be of a type that evaluates to an integer. That is, "int", "unsigned int", "bool", an enumerated type or any typedefed type that evaluates to one of these is legal. Also, the case values must be one of the legal values of the discriminant. Finally, a case value may not be specified more than once within the scope of a union declaration.

6. AN EXAMPLE OF AN XDR DATA DESCRIPTION

Here is a short XDR data description of a thing called a "file", which might be used to transfer files from one machine to another.

```
      const MAXUSERNAME = 32;     /* max length of a user name */
      const MAXFILELEN = 65535;   /* max length of a file      */
      const MAXNAMELEN = 255;     /* max length of a file name */

      /*
       * Types of files:
       */
      enum filekind {
         TEXT = 0,       /* ascii data */
         DATA = 1,       /* raw data   */
         EXEC = 2        /* executable */
      };

      /*
       * File information, per kind of file:
       */
      union filetype switch (filekind kind) {
      case TEXT:
         void;                           /* no extra information */
      case DATA:
         string creator<MAXNAMELEN>;     /* data creator         */
      case EXEC:
         string interpretor<MAXNAMELEN>; /* program interpretor  */
      };

      /*
       * A complete file:
       */
      struct file {
         string filename<MAXNAMELEN>; /* name of file    */
         filetype type;               /* info about file */
         string owner<MAXUSERNAME>;   /* owner of file   */
         opaque data<MAXFILELEN>;     /* file data       */
      };
```

Suppose now that there is a user named "john" who wants to store his lisp program "sillyprog" that contains just the data "(quit)". His file would be encoded as follows:

```
    OFFSET  HEX BYTES       ASCII    COMMENTS
    ------  ---------       -----    --------
     0      00 00 00 09     ....     -- length of filename = 9
     4      73 69 6c 6c     sill     -- filename characters
     8      79 70 72 6f     ypro     -- ... and more characters ...
    12      67 00 00 00     g...     -- ... and 3 zero-bytes of fill
    16      00 00 00 02     ....     -- filekind is EXEC = 2
    20      00 00 00 04     ....     -- length of interpretor = 4
    24      6c 69 73 70     lisp     -- interpretor characters
    28      00 00 00 04     ....     -- length of owner = 4
    32      6a 6f 68 6e     john     -- owner characters
    36      00 00 00 06     ....     -- length of file data = 6
    40      28 71 75 69     (qui     -- file data bytes ...
    44      74 29 00 00     t)..     -- ... and 2 zero-bytes of fill
```

7. TRADEMARKS AND OWNERS

SUN WORKSTATION	Sun Microsystems, Inc.
VAX	Digital Equipment Corporation
IBM-PC	International Business Machines Corporation
Cray	Cray Research
NFS	Sun Microsystems, Inc.
Ethernet	Xerox Corporation.
Motorola 68000	Motorola, Inc.
IBM 370	International Business Machines Corporation

APPENDIX A: ANSI/IEEE Standard 754-1985

The definition of NaNs, signed zero and infinity, and denormalized numbers from [3] is reproduced here for convenience. The definitions for quadruple-precision floating point numbers are analogs of those for single and double-precision floating point numbers, and are defined in [3].

In the following, 'S' stands for the sign bit, 'E' for the exponent, and 'F' for the fractional part. The symbol 'u' stands for an undefined bit (0 or 1).

For single-precision floating point numbers:

Type	S (1 bit)	E (8 bits)	F (23 bits)
signalling NaN	u	255 (max)	.0uuuuu---u (with at least one 1 bit)
quiet NaN	u	255 (max)	.1uuuuu---u
negative infinity	1	255 (max)	.000000---0
positive infinity	0	255 (max)	.000000---0
negative zero	1	0	.000000---0
positive zero	0	0	.000000---0

For double-precision floating point numbers:

Type	S (1 bit)	E (11 bits)	F (52 bits)
signalling NaN	u	2047 (max)	.0uuuuu---u (with at least one 1 bit)
quiet NaN	u	2047 (max)	.1uuuuu---u
negative infinity	1	2047 (max)	.000000---0
positive infinity	0	2047 (max)	.000000---0
negative zero	1	0	.000000---0
positive zero	0	0	.000000---0

For quadruple-precision floating point numbers:

Type	S (1 bit)	E (15 bits)	F (112 bits)
signalling NaN	u	32767 (max)	.0uuuuu---u (with at least one 1 bit)
quiet NaN	u	32767 (max)	.1uuuuu---u
negative infinity	1	32767 (max)	.000000---0
positive infinity	0	32767 (max)	.000000---0
negative zero	1	0	.000000---0
positive zero	0	0	.000000---0

Subnormal numbers are represented as follows:

Precision	Exponent	Value
Single	0	(-1)**S * 2**(-126) * 0.F
Double	0	(-1)**S * 2**(-1022) * 0.F
Quadruple	0	(-1)**S * 2**(-16382) * 0.F

APPENDIX B: REFERENCES

[1] Brian W. Kernighan & Dennis M. Ritchie, "The C Programming Language", Bell Laboratories, Murray Hill, New Jersey, 1978.

[2] Danny Cohen, "On Holy Wars and a Plea for Peace", IEEE Computer, October 1981.

[3] "IEEE Standard for Binary Floating-Point Arithmetic", ANSI/IEEE Standard 754-1985, Institute of Electrical and Electronics Engineers, August 1985.

[4] "Courier: The Remote Procedure Call Protocol", XEROX Corporation, XSIS 038112, December 1981.

[5] "The SPARC Architecture Manual: Version 8", Prentice Hall, ISBN 0-13-825001-4.

[6] "HP Precision Architecture Handbook", June 1987, 5954-9906.

[7] Srinivasan, R., "Remote Procedure Call Protocol Version 2", RFC 1831, Sun Microsystems, Inc., August 1995.

Security Considerations

Security issues are not discussed in this memo.

Author's Address

```
Raj Srinivasan
Sun Microsystems, Inc.
ONC Technologies
2550 Garcia Avenue
M/S MTV-5-40
Mountain View, CA  94043
USA

Phone: 415-336-2478
Fax:   415-336-6015
EMail: raj@eng.sun.com
```

[Index | Search | What's New | Comments | Help]

t.120

A Primer on the T.120 Series Standard

DataBeam™

A PRIMER ON THE T.120 SERIES STANDARDS

The T.120 standard contains a series of communication and application protocols and services that provide support for real-time, multipoint data communications. These multipoint facilities are important building blocks for a whole new range of collaborative applications, including desktop data conferencing, multi-user applications, and multi-player gaming.

Broad in scope, T.120 is a comprehensive specification that solves several problems that have historically slowed market growth for applications of this nature. Perhaps most importantly, T.120 resolves complex technological issues in a manner that is acceptable to both the computing and telecommunications industries.

Established by the International Telecommunications Union (ITU), T.120 is a family of open standards that was defined by leading data communication practitioners in the industry. Over 100 key international vendors, including Apple, AT&T, British Telecom, Cisco Systems, Intel, MCI, Microsoft, and PictureTel, have committed to implementing T.120-based products and services.

Broad vendor support means that end users will be able to choose from a variety of interoperable products.

While T.120 has emerged as a critical element in the data communications landscape, the only information that currently exists on the topic is a weighty and complicated set of standards documents. This primer bridges this information gap by summarizing T.120's major benefits, fundamental architectural elements, and core capabilities.

Key Benefits of T.120

So why all the excitement about T.120? The bottom line is that it provides exceptional benefits to end users, vendors, and developers tasked with implementing real-time applications. The following list is a high-level overview of the major benefits associated with the T.120 standard:

Multipoint Data Delivery

T.120 provides an elegant abstraction for developers to create and manage a multipoint domain with ease. From an application perspective, data is seamlessly delivered to multiple parties in "realtime."

Interoperability

T.120 allows endpoint applications from multiple vendors to interoperate. T.120 also specifies how applications may interoperate with (or through) a variety of network bridging products and services that also support the T.120 standard.

Reliable Data Delivery

Error-corrected data delivery ensures that all endpoints will receive each data transmission.

T.120 Benefits

- ✔ Multipoint Data Delivery
- ✔ Interoperability
- ✔ Reliable Data Delivery
- ✔ Multicast Enabled Delivery
- ✔ Network Transparency
- ✔ Platform Independence
- ✔ Network Independence
- ✔ Support for Varied Topologies
- ✔ Application Independence
- ✔ Scalability
- ✔ Co-existence with Other Standards
- ✔ Extendability

Multicast Enabled Delivery

In muliticast enabled networks, T.120 can employ reliable (ordered, guaranteed) and unreliable delivery services. Unreliable data delivery is also available without multicast. By using multicast, the T.120 infrastructure reduces network congestion and improves performance for the end user. The T.120 infrastructure can use both unicast and multicast simultaneously, providing a flexible solution for mixed unicast and multicast networks. The Multicast Adaptation Protocol (MAP) is expected to be ratified in early 1998.

Network Transparency

Applications are completely shielded from the underlying data transport mechanism being used. Whether the transport is a high-speed LAN or a simple dial-up modem, the application developer is only concerned with a single, consistent set of application services.

Platform Independence

Because the T.120 standard is completely free from any platform dependencies, it will readily take advantage of the inevitable advances in computing technology. In fact, DataBeam's customers have already ported the T.120 source code easily from Windows to a variety of environments, including OS/2, MAC/OS, several versions of UNIX, and other proprietary real-time operating systems.

Network Independence

The T.120 standard supports a broad range of transport options, including the Public Switched Telephone Networks (PSTN or POTS), Integrated Switched Digital Networks (ISDN), Packet Switched Digital Networks (PSDN), Circuit Switched Digital Networks (CSDN), and popular local area network protocols (such as TCP/IP and IPX via reference protocol). Furthermore, these vastly different network transports, operating at different speeds, can easily co-exist in the same multipoint conference.

Support for Varied Topologies

Multipoint conferences can be set up with virtually no limitation on network topology. Star topologies, with a single Multipoint Control Unit (MCU) will be common early on. The standard also supports a wide variety of other topologies ranging from those with multiple, cascaded MCUs to topologies as simple as a daisy-chain. In complex multipoint conferences, topology may have a significant impact on efficiency and performance.

Application Independence

Although the driving market force behind T.120 was teleconferencing, its designers purposely sought to satisfy a much broader range of application needs. Today, T.120 provides a generic, real-time communications facility that can be used by many different applications. These applications include interactive gaming, virtual reality and simulations, real-time subscription news feeds, and process control applications.

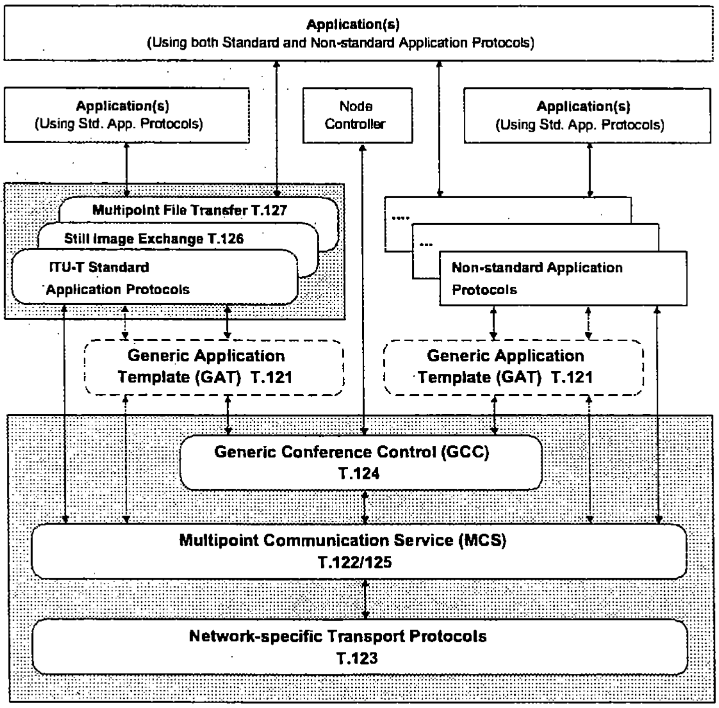
Scalability

T.120 is defined to be easily scalable from simple PC-based architectures to complex multi-processor environments characterized by their high performance. Resources for T.120 applications are plentiful, with practical limits imposed only by the confines of the specific platform running the software.

Co-existence with Other Standards

T.120 was designed to work alone or within the larger context of other ITU standards, such as the H.32x family of video conferencing standards. T.120 also supports and cross-references other important ITU standards, such as V.series modems.

FIGURE 1: MODEL OF ITU T.120 SERIES ARCHITECTURE


Application(s)
(Using both Standard and Non-standard Application Protocols)
Application(s)
(Using Std. App. Protocols)
Node Controller
Application(s)
(Using Std. App. Protocols)
Multipoint File Transfer T.127
Still Image Exchange T.126
ITU-T Standard Application Protocols
....
...
Non-standard Application Protocols
Generic Application Template (GAT) T.121
Generic Application Template (GAT) T.121
Generic Conference Control (GCC) T.124
Multipoint Communication Service (MCS) T.122/125
Network-specific Transport Protocols T.123

Extendability

The T.120 standard can be freely extended to include a variety of new capabilities, such as support for new transport stacks (like ATM or Frame Relay), improved security measures, and new application-level protocols.

Architectural Overview

The T.120 architecture relies on a multi-layered approach with defined protocols and service definitions between layers. Each layer presumes that all layers exist below. Figure 1 provides a graphical representation of the T.120 architecture.

The lower level layers (T.122, T.123, T.124, and T.125) specify an application-independent mechanism for providing multipoint data communication services to any application that can use these facilities. The upper level layers (T.126 and T.127) define protocols for specific conferencing applications, such as shared whiteboarding and multipoint file transfer. Applications using these standardized protocols can co-exist in the same conference with applications using proprietary protocols. In fact, a single application may even use a mix of standardized and non-standardized protocols.

Component Overview

The following overview describes the key characteristics and concepts behind each individual component of the T.120 standard. This overview starts at the bottom of the T.120 stack and progresses upward.

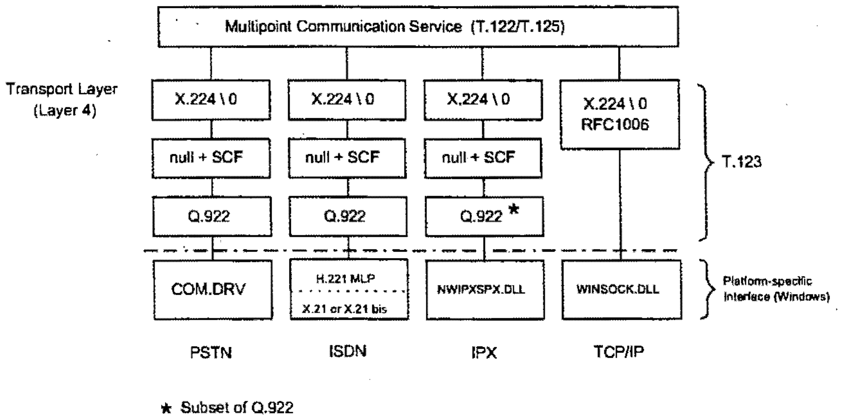
Transport Stacks - T.123

T.120 applications expect the underlying transport to provide reliable delivery of its Protocol Data Units (PDUs) and to segment and sequence that data. T.123 specifies transport profiles for each of the following:

- Public Switched Telephone Networks (PSTN)
- Integrated Switched Digital Networks (ISDN)
- Circuit Switched Digital Networks (CSDN)
- Packet Switched Digital Networks (PSDN)
- TCP/IP
- Novell Netware IPX (via reference profile)

As highlighted below in Figure 2, the T.123 layer presents a uniform OSI transport interface and services (X.214/X.224)

Figure 2: Cross-section of T.123 Transports (Basic Mode Profiles)



to the MCS layer above. The T.123 layer includes built-in error correction facilities so application developers do not have to rely on special hardware facilities to perform this function.

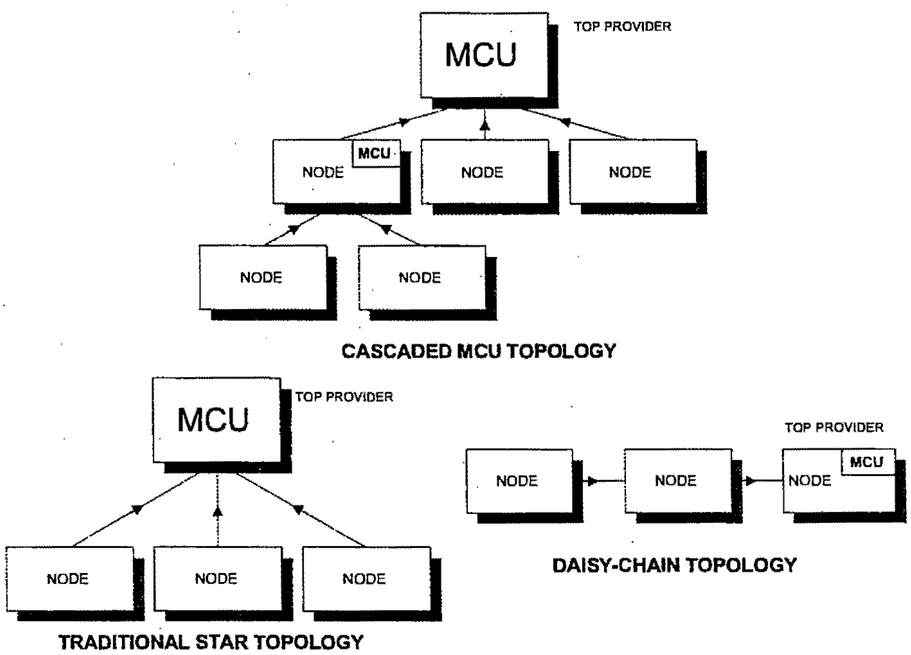
In a given computing environment, a transport stack typically plugs into a local facility that provides an interface to the specific transport connection. For example, in the Windows environment, DataBeam's transport stacks plug into COMM.DRV for modem communications, WINSOCK.DLL for TCP/IP and UDP/IP communications, and NWIPXSPX. DLL for Novell IPX communications support.

The MCU is a logical construct whose role may be served by a node on a desktop or by special-purpose equipment within the network.

The Multicast Adaptation Protocol (MAP) service layer is a new extension to MCS. MAP manages unicast- and multicast-based transports. MAP can be used with any transport where multicast is available, such as IP networks. While multicast provides unreliable delivery, many applications using T.120 require reliable services. Developers can incorporate a variety of multicast error correction schemes into MAP, thereby selecting the scheme most closely aligned with their application.

In 1996, the ITU is expected to adopt extensions to support important new transport facilities, such as Asynchronous Transfer Mode (ATM) and H.324 POTS videophone. It is necessary to note that developers can easily produce a proprietary transport stack (supporting, for example, AppleTalk) that transparently uses the services above T.123. An important function of MCUs or T.120-enabled bridges, routers, or gateways is to provide transparent interworking across different network boundaries.

FIGURE 3: EXAMPLES OF VALID MCS TOPOLOGIES



CASCADED MCU TOPOLOGY

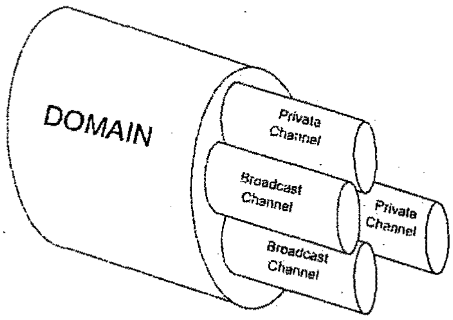
DAISY-CHAIN TOPOLOGY

TRADITIONAL STAR TOPOLOGY

Multipoint Communication Service (MCS) - T.122, T.125

T.122 defines the multipoint services available to the developer, while T.125 specifies the data transmission protocol. Together they form MCS, the multipoint "engine" of the T.120 conference. MCS relies on T.123 to deliver the data. (Use of MCS is entirely independent of the actual T.123 transport stack(s) that is loaded.)

FIGURE 4: CHANNEL DIAGRAM



MCS is a powerful tool that can be used to solve virtually any multipoint application design requirement. MCS is an elegant abstraction of a complex organism. Learning to use MCS effectively is the key to successfully developing real-time applications.

How MCS Works

In a conference, multiple endpoints (or MCS *nodes*) are logically connected together to form what T.120 refers to as a *domain*. Domains generally equate to the concept of a conference. An application may actually be *attached* to multiple domains simultaneously. For example, the chairperson of a large online conference may simultaneously monitor information being discussed among several activity groups.

In a T.120 conference, nodes connect upward to a Multipoint Control Unit (MCU). The MCU model in T.120 provides a reliable approach that works in both public and private networks. Multiple MCUs may be easily chained together in a single domain. Figure 3 illustrates three potential topology structures. Each domain has a single *Top Provider* or MCU that houses the information base critical to the conference. If the Top Provider either fails or leaves a conference, the conference is terminated. If a lower level MCU (i.e., not the Top Provider) fails, only the nodes on the tree below that MCU are dropped from the conference. Because all nodes contain MCS, they are all potentially "MCUs."

One of the critical features of the T.120 approach is the ability to direct data. This feature allows applications to communicate efficiently. MCS applications direct data within a domain via the use of *channels*. An application can choose to use multiple channels simultaneously for whatever purposes it needs (for example, separating annotation and file transfer operations). Application instances choose to obtain information by *subscribing* to whichever channel(s) contains the desired data. These channel assignments can be dynamically changed during the life of the conference. Figure 4 presents an overview of multiple channels in use within a domain.

It is the application developer's responsibility to determine how to use channels

TABLE 1: CHANNEL SETUP EXAMPLE

Channel	Type	Priority	Routing
1	Error Control Channels	Top	Standard
2	Annotations	High	Uniform
3	Bitmap Images	Medium	Uniform
4	File Transfer	Low	Standard

within an application. For example, an application may send control information along a single channel and application data along a series of channels that may vary depending upon the type of data being sent. The application developer may also take advantage of the MCS concept of *private channels* to direct data to a discrete subset of a given conference.

Data may be sent with one of four *priority* levels. MCS applications may also specify that data is routed along the quickest path of delivery using the *standard* send command. If the application uses the *uniform* send command, it ensures that data from multiple senders will arrive at all destinations in the same order. Uniform data always travels all the way up the tree to the Top Provider. Table 1 provides an example of how a document conferencing application could set up its channels. Reliable or unreliable data delivery is determined by the application.

There are no constraints on the size of the data sent from an application to MCS. Segmentation of data is automatically performed on behalf of the application. However, after receiving the data it is the application's responsibility to reassemble the data by monitoring flags provided when the data is delivered.

Tokens are the last major facility provided by MCS. Services are provided to grab, pass, inhibit, release, and query tokens. Token resources may be used as either exclusive (i.e., locking) or non-exclusive entities.

Tokens can be used by an application in a number of ways. For example, an application may specify that only the holder of a specific token, such as the conductor, may send information in the conference.

Another popular use of tokens is to coordinate tasks within a domain. For example, suppose a teacher wants to be sure that every student in a distance learning session answered a particular question before displaying the answer. Each node in the underlying application *inhibits* a specific token after receiving the request to answer the question. The token is released by each node when an answer is provided. In the background, the teacher's application continuously polls the state of the token. When all nodes have released the token, the application presents the teacher with a visual cue that the class is ready for the answer.

Generic Conference Control (GCC) - T.124

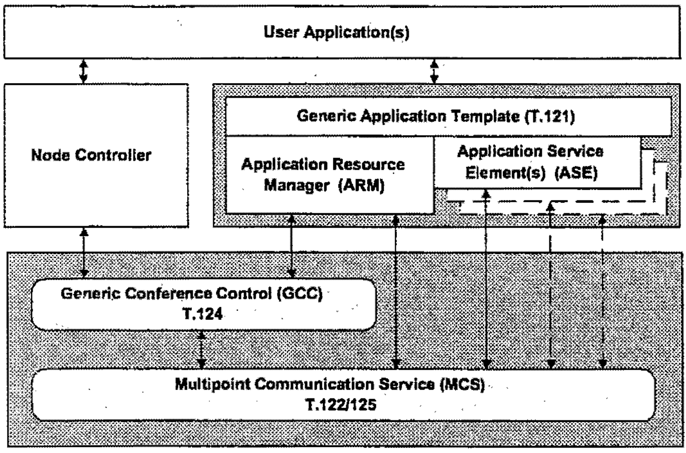
Generic Conference Control provides a comprehensive set of facilities for establishing and managing the multipoint conference. It is with GCC that we first see features that are specific to the electronic conference.

At the heart of GCC is an important information base about the state of the various conferences it may be servicing. One node, which may be the MCU itself, serves as the Top Provider for GCC information. Any actions or requests from lower GCC nodes ultimately filter up to this Top Provider.

One of GCC's most important roles is to maintain information about the nodes and applications that are in a conference.

Using mechanisms in GCC, applications create conferences, join conferences, and invite others to conferences. As endpoints join and leave conferences, the information base in GCC is updated and can be used to automatically notify all endpoints when these actions occur. GCC also knows who is the Top Provider for the conference. However, GCC does not contain detailed topology information about the means by which nodes from lower branches are connected to the conference.

FIGURE 5: T.121 GENERIC APPLICATION TEMPLATE


User Application(s)
Node Controller
Generic Application Template (T.121)
Application Resource Manager (ARM)
Application Service Element(s) (ASE)
Generic Conference Control (GCC) T.124
Multipoint Communication Service (MCS) T.122/125

Every application in a conference must register its unique *application key* with GCC. This enables any subsequent joining nodes to find compatible applications. Furthermore, GCC provides robust facilities for applications to exchange capabilities and arbitrate feature sets. In this way, applications from different vendors can readily establish whether or not they can interoperate and at what feature level. This arbitration facility is the mechanism used to ensure backward compatibility between different versions of the same application.

GCC also provides conference security. This allows applications to incorporate password protection or "lock" facilities to prevent uninvited users from joining a conference.

Another key function of GCC is its ability to dynamically track MCS resources. Since multiple applications can use MCS at the same time, applications rely on GCC to prevent conflicts for MCS resources, such as channels and tokens. This ensures that applications do not step on each other by attaching to the same channel or requesting a token already in use by another application.

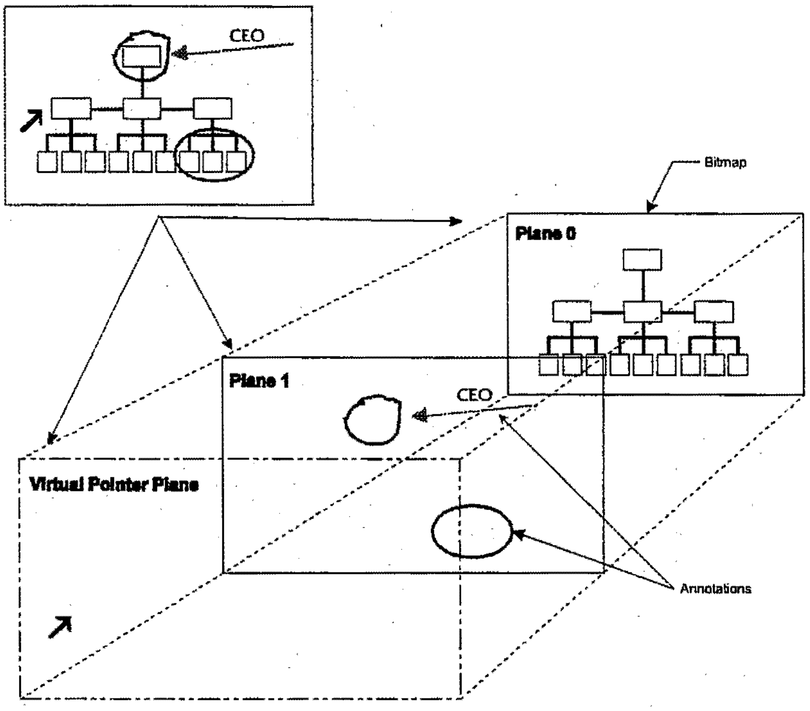
Finally, GCC provides capabilities for supporting the concept of *conductorship* in a conference. GCC allows the application to identify the conductor and a means in which to transfer the conductor's "baton." The developer is free to decide how to use these conductorship facilities within the application.

T.124 Revised

As part of the ongoing enhancement process for the T.120 standards, the ITU has completed a draft revision of T.124. The new version, called T.124 Revised, introduces a number of changes to improve scalability. The most significant changes address the need to distribute roster information to all nodes participating in a conference, as well as improvements in the efficiency of sending roster refresh information (from the Top Provider) any time a node joins or leaves a conference.

To improve the distribution of roster information, the concept of Node Categories was introduced. These categories provide a way for a T.124 node to join or leave a conference without affecting the roster information that was distributed throughout a conference. In addi-

FIGURE 6: T.126 WORKSPACE DIAGRAM



tion, the Full Roster Refresh, which was previously sent any time a new node joined a conference, was eliminated by sending out roster details from the Top Provder. These changes will not affect backward compatibility to earlier revisions of T.124. This revision will go to the ITU for Decision in March of 1998.

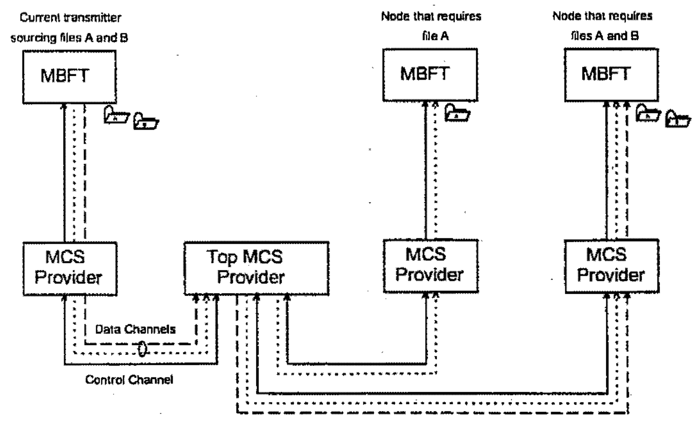
Generic Application Template (GAT) - T.121

T.121 provides a template for T.120 resource management that developers should use as a guide for building application protocols. T.121 is mandatory for standardized application protocols and is highly recommended for non-standard application protocols. The template ensures consistency and reduces the potential for unforeseen interaction between different protocol implementations.

Within the T.121 model, GAT defines a generic Application Resource Manager (ARM). This entity manages GCC and MCS resources on behalf of the application protocol-specific functionality defined as an Application Service Element (ASE). Figure 5 demonstrates the GAT model within the T.120 architecture. Simply put, GAT provides a consistent model for managing T.120 resources required by the application to which the developer adds application-specific functionality.

GAT's functionality is considered to be generic and common to all application protocols. GAT's services include enrolling the application in GCC and attaching to MCS domains. GAT also manages channels, tokens, and capabilities on behalf of the application. On a broader scale, GAT responds to GCC indica-

FIGURE 7: T.127 FILE TRANSFER MODEL


Current transmitter sourcing files A and B
Node that requires file A
Node that requires files A and B
MBFT
MBFT
MBFT
MCS Provider
Top MCS Provider
MCS Provider
MCS Provider
Data Channels
Control Channel

tions and can invoke peer applications on other nodes in the conference.

Still Image Exchange and Annotation (SI) - T.126

T.126 defines a protocol for viewing and annotating still images transmitted between two or more applications. This capability is often referred to as document conferencing or *shared whiteboarding*.

An important benefit of T.126 is that it readily shares visual information between applications that are running on dramatically different platforms. For example, a Windows-based desktop application could easily interoperate with a collaboration program running on a PowerMac. Similarly, a group-oriented conferencing system, without a PC-style interface, could share data with multiple users running common PC desktop software.

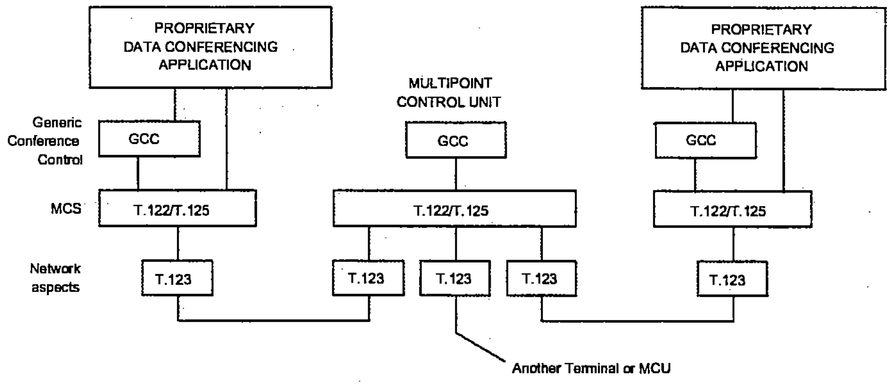
As Figure 6 illustrates, T.126 presents the concept of shared virtual *workspaces* that are manipulated by the endpoint applications. Each workspace may contain a collection of objects that include bitmap images and annotation primitives, such as rectangles and freehand lines. Bitmaps typically originate from application information, such as a word processing document or a presentation slide. Because of their size, bitmaps are often compressed to improve performance over lower-speed communication links.

T.126 is designed to provide a minimum set of capabilities required to share information between disparate applications. Because T.126 is simply a protocol, it does not provide any of the API-level structures that allow application developers to easily incorporate shared whiteboarding into an application. These types of facilities can only be found in toolkit-level implementations of the standard (such as DataBeam's Shared Whiteboard Application Toolkit, known as SWAT).

Multipoint Binary File Transfer - T.127

T.127 specifies a means for applications to transmit files between multiple endpoints in a conference. Files can be transferred to all participants in the conference or to a specified subset of the conference. Multiple file transfer operations may occur simultaneously in any given conference and developers can specify priority levels for the file delivery. Finally, T.127 provides options for compressing files before delivering the data. Figure 7 dis-

FIGURE 8: NETWORK-LEVEL INTEROPERABILITY DIAGRAM



plays a view of conference-wide and individual file transfers.

Node Controller

The Node Controller manages defined GCC Service Access Points (SAPs). This provides the node flexibility in responding to GCC events. Most of these GCC events relate to establishing conferences, adding or removing nodes from a conference, and breaking down and distributing information. The Node Controller's primary responsibility is to translate these events and respond appropriately.

Some GCC events can be handled automatically; for example, when a remote party joins a conference, each local Node Controller can post a simple message informing the local user that "Bill Smith has joined the conference." Other events may require user intervention; for example, when a remote party issues an invitation to join a conference, the local Node Controller posts a dialog box stating that "Mary Jones has invited you to the Design Review conference. <Accept> <Decline>."

Node controllers can be MCU-based, terminal-based, or dual-purpose. DataBeam's application, FarSite, for example, contains a dual-purpose Node Controller. The range of functionality found within a Node Controller can vary dramatically by implementation.

Only one Node Controller can exist on an active T.120 endpoint. Therefore, if multiple applications need to simultaneously use T.120 services, the Node Controller needs to be accessible to each application. The local interface to the Node Controller is application- and vendor-specific and is not detailed in the T.120 documentation.

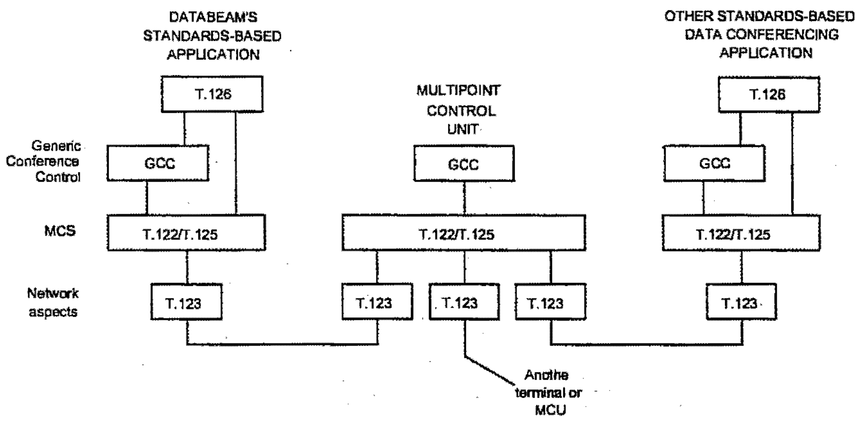
INTEROPERABILITY

Buyers overwhelmingly rate interoperability as the number one purchase criteria in their evaluation of teleconferencing products. For most end users, interoperability translates to "my application can talk to your application"—regardless of which vendor supplied the product or on what platform it runs. When examining the T.120 standard closely, buyers can see that it provides for two levels of interoperability: application-level interoperability and network-level interoperability.

Network-level Interoperability

Network-level interoperability means that a given product can interwork with like products through the infrastructure of

FIGURE 9: APPLICATION-LEVEL INTEROPERABILITY DIAGRAM



network products and services that support T.120. For example, T.120-based conferencing bridges (MCUs) that can support hundreds of simultaneous users are now being developed. If an application supports only the lower layers of T.120, customers can use these MCUs to host a multipoint conference only if everyone in the conference is using the exact same product. Figure 8 displays network interoperability through a conference of *like products.*

Application-level Interoperability

The upper levels of T.120 specify protocols for common conferencing applications, such as shared whiteboarding and binary file transfer. Applications supporting these protocols can interoperate with any other application that provides similar support, regardless of the vendor or platform used. For example, through T.126, users of DataBeam's FarSite application will be able to share and mark up documents with users of group conferencing systems. This interoperability will exist in simple point-to-point conferences as well as large multipoint conferences using a conference bridge. Figure 9 represents application-level interoperability between two standards-based applications connected in a conference.

In the short-term, network-level interoperability will be the most common form of T.120 support found in conferencing applications. This is largely due to the fact that the lower-level T.120 layers were ratified by the ITU more than a year in advance of the application-level layers. However, end users will not be satisfied with network interoperability alone. For the market to grow, vendors will have to deliver the same application-level interoperability (or endpoint interoperability) that customers enjoy today with fax machines and telephones.

RATIFICATION OF THE T.120 AND FUTURE T.130 STANDARDS

The Recommendations for the core multipoint communications infrastructure components (T.122, T.123, T.124 and T.125) were ratified by the ITU between March of 1993 and March of 1995. The first of the application standards (T.126 and T.127) was approved in

March of 1995. An overview of the T.120 series was approved in February of 1996 as Recommendation T.120. T.121 (GAT) was also approved at that time. Stable drafts of these recommendations existed for some time prior to the ratification, thereby providing a means for DataBeam to actively develop products in parallel to the standardization effort.

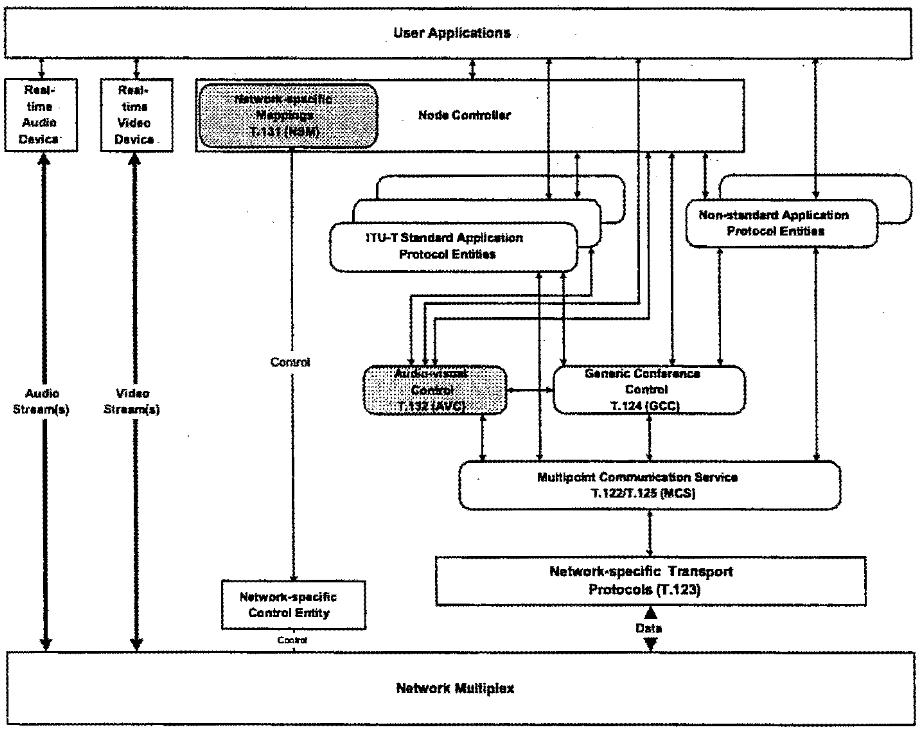
The existing ratified standards are being actively discussed for possible amendments and extensions. This commonly occurs when implementation and interoperability issues arise.

T.130 Audio-visual Control For Multimedia Conferencing

The T.130 series of recommendations define an architecture, a management and control protocol, and a set of services which together make up an Audio-Visual Control system (AVC). This system supports the use of real-time streams and services in a multimedia conferencing environment. The protocol and services section, outlined in T.132, consists of two parts: management and control. Together, they allow Network Elements, such as the traditional MCU, Gateway, or Conference Server, to provide T.132 audio and video services to their endpoints. Some of the services include Stream Identification, On-Air Identification, Video Switching, Audio Mixing, Remote Device Control, and Continuous Presence.

The T.130 series is built upon existing ITU-T conferencing recommendations such as the H.320 audio-visual conferencing series and the T.120 series for multipoint data conferencing. The T.130 series is compatible with systems, such as H.323, in which audio and video are

FIGURE 10: AUDIO-VISUAL CONTROL ARCHITECTURE


User Applications
Real-time Audio Device
Real-time Video Device
Network-specific Mappings T.131 (NSM)
Node Controller
ITU-T Standard Application Protocol Entities
Non-standard Application Protocol Entities
Control
Audio Stream(s)
Video Stream(s)
Audio-visual Control T.132 (AVC)
Generic Conference Control T.124 (GCC)
Multipoint Communication Service T.122/T.125 (MCS)
Network-specific Transport Protocols (T.123)
Network-specific Control Entity
Control
Data
Network Multiplex

transmitted independently of T.120, as well as systems which are capable of transmitting multiple media types within a common multiplex.

Unlike other standardized methods for managing real-time streams within a conference, T.130 provides some unique capabilities:

- Contains a network- and platform-independent control protocol for managing real-time streams
- Coordinates operations across network boundaries
- Processes and distributes media streams within a conference environment
- Delivers of Quality of Service (QoS) to multimediacommunications applications
- Provides distributed conference management
- Leverages the functionality of existing multimedia protocols

T.130 can be used in any conferencing scenario where there is a need for multipoint audio or video. T.130 relies upon the services of GCC and MCS to transmit control data, but the audio and video streams are transported in independent logical channels due to the transmission requirements of real-time data flows. (See Figure 10).

T.130 and T.132 were determined in March of 1997 and should be ratified in January of 1998. T.131, which defines network-specific mappings to allow AVC to communicate with the underlying Multimedia Control Protocol, such as H.245, should be determined in the Fall of 1997.

Vendor Community Support for T.120

More than 100 multinational companies have pledged their support for the T.120 standard and more are being added to this list every week. Public supporters of T.120 include international market leaders, such as Apple, AT&T, British Telecom, Cisco Systems, Deutsche Telecom, IBM, Intel, MCI, Microsoft, Motorola, PictureTel, and DataBeam.

Most supporters of T.120 are also members of the International Multimedia Teleconferencing Consortium (IMTC). The goals of the IMTC are to promote the awareness and adoption of ITU teleconferencing standards, including T.120 and H.32x. The IMTC provides a forum for interoperability testing and helps to define Application Programming Interfaces (APIs). DataBeam's co-founder and chief technical officer, C. J. "Neil" Starkey, serves as the president of the IMTC. Previously, Starkey served for six years as chairman of the ITU study group that defined T.120.

New Markets for T.120 Deployment

The teleconferencing community is the first market segment to adopt the T.120 standard. Because the technology is broad in scope, it can be effectively used by a number of other application software vendors and equipment providers.

The computing paradigm is rapidly extending past today's personal productivity model. Over the next two years, we will witness the development of a new generation of application software that incorporates multi-party collaboration. Independent Software Vendors (ISVs) have begun to adopt T.120 as the means in which to incorporate real-time collaboration capabilities into common desktop applications, such as word processing and presentation graphics. Engineering products, such as Computer Aided Design (CAD) software, are also on the migration path to T.120 technology. Other ISVs with a strong interest in T.120 include developers of fax, remote control, document imaging, and "overtime" collaboration products, such as Lotus Notes.

With T.120 technology in the hands of operating system providers and horizontal application vendors, network equipment providers are beginning to take notice. For vendors of PBXs, network bridges, hubs, routers and switches, T.120 represents an important opportunity to provide value-added capabilities within their network products. In the short-term, these features will represent an opportunity for competitive advantage. However, within the next year, T.120 support will be a required feature.

Finally, we can envision a whole range of T.120 applications in the areas of interactive video, network gaming, and simulations. From Nintendo to DOOM to set-top boxes, the need for bidirectional multipoint data communications is acute. The ability to use a common set of APIs and protocols that are broadly supported from the desktop through the network will drive the adoption of T.120 into these important emerging markets.

IMTC, ITU, AND T.120

Standards have played an important part in the establishment and growth of several consumer and telecommunications markets. By creating a basic commonality, standards ensure compatibility among products from different manufacturers. This encourages companies to produce varying solutions and encourages end users to purchase the solutions without fear of obsolescence or incompatibility.

The work of both the IMTC and the ITU represents organized efforts to promote a basic connectivity protocol that will encourage the growth of the multimedia telecommunications market. The Standards First™ initiative, which is supported by many industry leaders, requires a minimum of H.320 and T.120 compliance, which is enough to establish this basic connectivityprotocol. Manufacturers are then able to build on the basic compliance by adding features to their products, creating Standards Plus equipment.

With Standards First, the IMTC has the end users' interests in mind. By ensuring interoperability among equipment from competing manufacturers, Standards First also ensures that a customer's initial investment is protected and future system upgrades are possible. The IMTC is helping to educate the industry and the public about the importance, function, and status of standards. In addition, the organization provides a coordination point for industry leaders to communicate their interests to the ITU-T. As the multipoint multimedia teleconferencing industry continues its rapid growth, the development and implementation of standards for interoperability, and the work of the IMTC, will be instrumental in securing the market's future.

IMPLEMENTING T.120

With the T.120 set of standards in place, third-party developers are faced with yet another challenge— implementation. DataBeam's Collaborative Computing Toolkit Series (CCTS™) has jump-started the conferencing industry by providing the first standards-based toolkits for developing multipoint, data-sharing applications. These toolkits encapsulate the complex system-wide, multipoint communications stacks that allow application developers to rapidly embed sophisticated real-time, data-sharing capabilities into new or existing products. Simply stated, CCTS provides a seamless solution for parties developing standards-based communication solutions.

As a result, DataBeam envisions an acceleration in the development of software applications and network infrastructure products such as, PBXs, bridges, routers, network switches, and LAN servers, that incorporate T.120. In addition, the industry will grow well beyond today's existing paradigms and the world will begin to see a whole range of new products and services that incorporate T.120. Users waiting for the standards dust to settle can now feel confident that with the support of vendors like Microsoft, DataBeam's T.120-based Collaborative Computing Toolkit Series is the best solution for industry-wide interoperability.

T.120 Information Sources

DataBeam Corporation
3191 Nicholasville Road
Lexington, Kentucky 40503
USA

Phone: (606) 245-3500
Fax: (606) 245-3528
E-Mail: info@databeam.com
Web Page: http://www.databeam.com

International Telecommunications Union
Sales Service
Place des Nations
CH-1211 Genéve 20
Switzerland

Phone: +41 22 730 6141
Fax: +41 22 730 5194
E-Mail: sales@itu.ch
Web Page: http://www.itu.ch

International Multimedia Teleconferencing Consortium, Inc.
111 Deerwood Road, Suite 372
San Ramon, California 94583
USA

Phone: (510) 743-4455
Fax: (510) 743-9011
E-Mail: dkamlani@imtc.fabrik.com
Web Page: http://www.imtc.org/imtc

DataBeam™

Updated May 14,1997.

DISTRIBUTED GAME ENVIRONMENT

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is related to U.S. Patent Application No. _____________, entitled "BROADCASTING NETWORK," filed on July 31, 2000 (Attorney Docket No. 030048001 US); U.S. Patent Application No._______________, entitled "JOINING A BROADCAST CHANNEL," filed on July 31, 2000 (Attorney Docket No. 030048002 US); U.S. Patent Application No._____________, "LEAVING A BROADCAST CHANNEL," filed on July 31, 2000 (Attorney Docket No. 030048003 US); U.S. Patent Application No._____________, entitled "BROADCASTING ON A BROADCAST CHANNEL," filed on July 31, 2000 (Attorney Docket No. 030048004 US); U.S. Patent Application No._____________, entitled "CONTACTING A BROADCAST CHANNEL," filed on July 31, 2000 (Attorney Docket No. 030048005 US); U.S. Patent Application No._____________, entitled "DISTRIBUTED AUCTION SYSTEM," filed on July 31, 2000 (Attorney Docket No. 030048006 US); U.S. Patent Application No._____________, entitled "AN INFORMATION DELIVERY SERVICE," filed on July 31, 2000 (Attorney Docket No. 030048007 US); U.S. Patent Application No._____________, entitled "DISTRIBUTED CONFERENCING SYSTEM," filed on July 31, 2000 (Attorney Docket No. 030048008 US); and U.S. Patent Application No._____________, entitled "DISTRIBUTED GAME ENVIRONMENT," filed on July 31, 2000 (Attorney Docket No. 030048009 US), the disclosures of which are incorporated herein by reference.

TECHNICAL FIELD

The described technology relates generally to a computer network and more particularly, to a broadcast channel for a subset of a computers of an underlying network.

BACKGROUND

There are a wide variety of computer network communications techniques such as point-to-point network protocols, client/server middleware, multicasting network protocols, and peer-to-peer middleware. Each of these communications techniques have their advantages and disadvantages, but none is particularly well suited to the simultaneous sharing of information among computers that are widely distributed. For example, collaborative processing applications, such as a network meeting programs, have a need to distribute information in a timely manner to all participants who may be geographically distributed.

The point-to-point network protocols, such as UNIX pipes, TCP/IP, and UDP, allow processes on different computers to communicate via point-to-point connections. The interconnection of all participants using point-to-point connections, while theoretically possible, does not scale well as a number of participants grows. For example, each participating process would need to manage its direct connections to all other participating processes. Programmers, however, find it very difficult to manage single connections, and management of multiple connections is much more complex. In addition, participating processes may be limited to the number of direct connections that they can support. This limits the number of possible participants in the sharing of information.

The client/server middleware systems provide a server that coordinates the communications between the various clients who are sharing the information. The server functions as a central authority for controlling access to shared resources. Examples of client/server middleware systems include remote procedure calls ("RPC"), database servers, and the common object request broker architecture ("CORBA"). Client/server middleware systems are not particularly well suited to sharing of information among many participants. In particular, when a client stores information to be shared at the server, each other client would need to poll the server to determine that new information is being shared. Such polling places a very high overhead on the communications network. Alternatively, each client may register a callback with the server, which the server then invokes when new information is available to be shared. Such a callback technique presents a performance bottleneck because a single server needs to call back to each client whenever new information is to be shared. In addition, the reliability of the entire sharing of information depends upon the reliability of the single server. Thus, a failure at a single computer (*i.e.*, the server) would prevent communications between any of the clients.

The multicasting network protocols allow the sending of broadcast messages to multiple recipients of a network. The current implementations of such multicasting network protocols tend to place an unacceptable overhead on the underlying network. For example, UDP multicasting would swamp the Internet when trying to locate all possible participants. IP multicasting has other problems that include needing special-purpose infrastructure (*e.g.*, routers) to support the sharing of information efficiently.

The peer-to-peer middleware communications systems rely on a multicasting network protocol or a graph of point-to-point network protocols. Such peer-to-peer middleware is provided by the T.120 Internet standard, which is used in such products as Data Connection's D.C.-share and Microsoft's NetMeeting. These peer-to-peer middleware systems rely upon a user to assemble a point-to-point graph of the connections used for sharing the information. Thus, it is neither suitable nor desirable to use peer-to-peer middleware systems when more than a small number of participants is desired. In addition, the underlying architecture of the T.120 Internet standard is a tree structure, which relies on the root node of the tree for reliability of the entire network. That is, each message must pass through the root node in order to be received by all participants.

It would be desirable to have a reliable communications network that is suitable for the simultaneous sharing of information among a large number of the processes that are widely distributed.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates a graph that is 4-regular and 4-connected which represents a broadcast channel.

Figure 2 illustrates a graph representing 20 computers connected to a broadcast channel.

Figures 3A and 3B illustrate the process of connecting a new computer Z to the broadcast channel.

Figure 4A illustrates the broadcast channel of Figure 1 with an added computer.

Figure 4B illustrates the broadcast channel of Figure 4A with an added computer.

Figure 4C also illustrates the broadcast channel of Figure 4A with an added computer.

Figure 5A illustrates the disconnecting of a computer from the broadcast channel in a planned manner.

Figure 5B illustrates the disconnecting of a computer from the broadcast channel in an unplanned manner.

Figure 5C illustrates the neighbors with empty ports condition.

Figure 5D illustrates two computers that are not neighbors who now have empty ports.

Figure 5E illustrates the neighbors with empty ports condition in the small regime.

Figure 5F illustrates the situation of Figure 5E when in the large regime.

Figure 6 is a block diagram illustrating components of a computer that is connected to a broadcast channel.

Figure 7 is a block diagram illustrating the sub-components of the broadcaster component in one embodiment.

Figure 8 is a flow diagram illustrating the processing of the connect routine in one embodiment.

Figure 9 is a flow diagram illustrating the processing of the seek portal computer routine in one embodiment.

Figure 10 is a flow diagram illustrating the processing of the contact process routine in one embodiment.

Figure 11 is a flow diagram illustrating the processing of the connect request routine in one embodiment.

Figure 12 is a flow diagram of the processing of the check for external call routine in one embodiment.

Figure 13 is a flow diagram of the processing of the achieve connection routine in one embodiment.

Figure 14 is a flow diagram illustrating the processing of the external dispatcher routine in one embodiment.

Figure 15 is a flow diagram illustrating the processing of the handle seeking connection call routine in one embodiment.

Figure 16 is a flow diagram illustrating processing of the handle connection request call routine in one embodiment.

Figure 17 is a flow diagram illustrating the processing of the add neighbor routine in one embodiment.

Figure 18 is a flow diagram illustrating the processing of the forward connection edge search routine in one embodiment.

Figure 19 is a flow diagram illustrating the processing of the handle edge proposal call routine.

Figure 20 is a flow diagram illustrating the processing of the handle port connection call routine in one embodiment.

Figure 21 is a flow diagram illustrating the processing of the fill hole routine in one embodiment.

Figure 22 is a flow diagram illustrating the processing of the internal dispatcher routine in one embodiment.

Figure 23 is a flow diagram illustrating the processing of the handle broadcast message routine in one embodiment.

Figure 24 is a flow diagram illustrating the processing of the distribute broadcast message routine in one embodiment.

Figure 26 is a flow diagram illustrating the processing of the handle connection port search statement routine in one embodiment.

Figure 27 is a flow diagram illustrating the processing of the court neighbor routine in one embodiment.

Figure 28 is a flow diagram illustrating the processing of the handle connection edge search call routine in one embodiment.

Figure 29 is a flow diagram illustrating the processing of the handle connection edge search response routine in one embodiment.

Figure 30 is a flow diagram illustrating the processing of the broadcast routine in one embodiment.

Figure 31 is a flow diagram illustrating the processing of the acquire message routine in one embodiment.

Figure 32 is a flow diagram illustrating processing of the handle condition check message in one embodiment.

Figure 33 is a flow diagram illustrating processing of the handle condition repair statement routine in one embodiment.

Figure 34 is a flow diagram illustrating the processing of the handle condition double check routine.

DETAILED DESCRIPTION

A broadcast technique in which a broadcast channel overlays a point-to-point communications network is provided. The broadcasting of a message over the broadcast channel is effectively a multicast to those computers of the network that are currently connected to the broadcast channel. In one embodiment, the broadcast technique provides a logical broadcast channel to which host computers through their executing processes can be connected. Each computer that is connected to the broadcast channel can broadcast messages onto and receive messages off of the broadcast channel. Each computer that is connected to the broadcast channel receives all messages that are broadcast while it is connected. The logical broadcast channel is implemented using an underlying network system (*e.g.*, the Internet) that allows each computer connected to the underlying network system to send messages to each other connected computer using each computer's address. Thus, the broadcast technique effectively provides a broadcast channel using an underlying network system that sends messages on a point-to-point basis.

The broadcast technique overlays the underlying network system with a graph of point-to-point connections (*i.e.*, edges) between host computers (*i.e.*, nodes) through which the broadcast channel is implemented. In one embodiment, each computer is connected to four other computers, referred to as neighbors. (Actually, a process executing on a computer is connected to four other processes executing on this or four other computers.) To broadcast a message, the originating computer sends the message to each of its neighbors using its point-to-point connections. Each computer that receives the message then sends the message to its three other neighbors using the point-to-point connections. In this way, the message is propagated to each computer using the underlying network to effect the broadcasting of the message to each computer over a logical broadcast channel. A graph in which each node is connected to four other nodes is referred to as a 4-regular graph. The use of a 4-regular graph means that a computer would become disconnected from the broadcast channel only if all four of the connections to its neighbors fail. The graph used by the broadcast technique also has the property that it would take a failure of four computers to divide the graph into disjoint sub-graphs, that is two separate broadcast channels. This property is referred to as being 4-connected. Thus, the graph is both 4-regular and 4-connected.

Figure 1 illustrates a graph that is 4-regular and 4-connected which represents the broadcast channel. Each of the nine nodes A-I represents a computer that is connected to the broadcast channel, and each of the edges represents an "edge" connection between two computers of the broadcast channel. The time it takes to broadcast a message to each computer on the broadcast channel depends on the speed of the connections between the computers and the number of connections between the originating computer and each other computer on the broadcast channel. The minimum number of connections that a message would need to traverse between each pair of computers is the "distance" between the computers (*i.e.*, the shortest path between the two nodes of the graph). For example, the distance between computers A and F is one because computer A is directly connected to computer F. The distance between computers A and B is two because there is no direct connection between computers A and B, but computer F is directly connected to computer B. Thus, a message originating at computer A would be sent directly to computer F, and then sent from computer F to computer B. The maximum of the distances between the computers is the "diameter" of broadcast channel. The diameter of the broadcast channel represented by Figure 1 is two. That is, a message sent by any computer would traverse no more than two connections to reach every other computer. Figure 2 illustrates a graph representing 20 computers connected to a broadcast channel. The diameter of this broadcast channel is 4. In particular, the shortest path between computers 1 and 3 contains four connections (1-12, 12-15, 15-18, and 18-3).

The broadcast technique includes (1) the connecting of computers to the broadcast channel (*i.e.*, composing the graph), (2) the broadcasting of messages over the broadcast channel (*i.e.*, broadcasting through the graph), and (3) the disconnecting of computers from the broadcast channel (*i.e.*, decomposing the graph) composing the graph.

Composing the Graph

To connect to the broadcast channel, the computer seeking the connection first locates a computer that is currently fully connected to the broadcast channel and then establishes a connection with four of the computers that are already connected to the broadcast channel. (This assumes that there are at least four computers already connected to the broadcast channel. When there are fewer than five computers connected, the broadcast channel cannot be a 4-regular graph. In such a case, the broadcast channel is considered to be in a "small regime." The broadcast technique for the small regime is described below in detail. When five or more computers are connected, the broadcast channel is considered to be in the "large regime." This description assumes that the broadcast channel is in the large regime, unless specified otherwise.) Thus, the process of connecting to the broadcast channel includes locating the broadcast channel, identifying the neighbors for the connecting computer, and then connecting to each identified neighbor. Each computer is aware of one or more "portal computers" through which that computer may locate the broadcast channel. A seeking computer locates the broadcast channel by contacting the portal computers until it finds one that is currently fully connected to the broadcast channel. The found portal computer then directs the identifying of four computers (*i.e.*, to be the seeking computer's neighbors) to which the seeking computer is to connect. Each of these four computers then cooperates with the seeking computer to effect the connecting of the seeking computer to the broadcast channel. A computer that has started the process of locating a portal computer, but does not yet have a neighbor, is in the "seeking connection state." A computer that is connected to at least one neighbor, but not yet four neighbors, is in the "partially connected state." A computer that is currently, or has been, previously connected to four neighbors is in the "fully connected state."

Since the broadcast channel is a 4-regular graph, each of the identified computers is already connected to four computers. Thus, some connections between computers need to be broken so that the seeking computer can connect to four computers. In one embodiment, the broadcast technique identifies two pairs of computers that are currently connected to each other. Each of these pairs of computers breaks the connection between them, and then each of the four computers (two from each pair) connects to the seeking computer. Figures 3A and 3B illustrate the process of a new computer Z connecting to the broadcast channel. Figure 3A illustrates the broadcast channel before computer Z is connected. The pairs of computers B and E and computers C and D are the two pairs that are identified as the neighbors for the new computer Z. The connections between each of these pairs is broken, and a connection between computer Z and each of computers B, C, D, and E is established as indicated by Figure 3B. The process of breaking the connection between two neighbors and reconnecting each of the former neighbors to another computer is referred to as "edge pinning" as the edge between two nodes may be considered to be stretched and pinned to a new node.

Each computer connected to the broadcast channel allocates five communications ports for communicating with other computers. Four of the ports are referred to as "internal" ports because they are the ports through which the messages of the broadcast channels are sent. The connections between internal ports of neighbors are referred to as "internal" connections. Thus, the internal connections of the broadcast channel form the 4-regular and 4-connected graph. The fifth port is referred to as an "external" port because it is used for sending non-broadcast messages between two computers. Neighbors can send non-broadcast messages either through their internal ports of their connection or through their external ports. A seeking computer uses external ports when locating a portal computer.

In one embodiment, the broadcast technique establishes the computer connections using the TCP/IP communications protocol, which is a point-to-point protocol, as the underlying network. The TCP/IP protocol provides for reliable and ordered delivery of messages between computers. The TCP/IP protocol provides each computer with a "port space" that is shared among all the processes that may execute on that computer. The ports are identified by numbers from 0 to 65,535. The first 2056 ports are reserved for specific applications (*e.g.*, port 80 for HTTP messages). The remainder of the ports are user ports that are available to any process. In one embodiment, a set of port numbers can be reserved for use by the computer connected to the broadcast channel. In an alternative embodiment, the port numbers used are dynamically identified by each computer. Each computer dynamically identifies an available port to be used as its call-in port. This call-in port is used to establish connections with the external port and the internal ports. Each computer that is connected to the broadcast channel can receive non-broadcast messages through its external port. A seeking computer tries "dialing" the port numbers of the portal computers until a portal computer "answers," a call on its call-in port. A portal computer answers when it is connected to or attempting to connect to the broadcast channel and its call-in port is dialed. (In this description, a telephone metaphor is used to describe the connections.) When a computer receives a call on its call-in port, it transfers the call to another port. Thus, the seeking computer actually communicates through that transfer-to port, which is the external port. The call is transferred so that other computers can place calls to that computer via the call-in port. The seeking computer then communicates via that external port to request the portal computer to assist in connecting the seeking computer to the broadcast channel. The seeking computer could identify the call-in port number of a portal computer by successively dialing each port in port number order. As discussed below in detail, the broadcast technique uses a hashing algorithm to select the port number order, which may result in improved performance.

A seeking computer could connect to the broadcast channel by connecting to computers either directly connected to the found portal computer or directly connected to one of its neighbors. A possible problem with such a scheme for identifying the neighbors for the seeking computer is that the diameter of the broadcast channel may increase when each seeking computer uses the same found portal computer and establishes a connection to the broadcast channel directly through that found portal computer. Conceptually, the graph becomes elongated in the direction of where the new nodes are added. Figures 4A-4C illustrate that possible problem. Figure 4A illustrates the broadcast channel of Figure 1 with an added computer. Computer J was connected to the broadcast channel by edge pinning edges C-D and E-H to computer J. The diameter of this broadcast channel is still two. Figure 4B illustrates the broadcast channel of Figure 4A with an added computer. Computer K was connected to the broadcast channel by edge pinning edges E-J and B-C to computer K. The diameter of this broadcast channel is three, because the shortest path from computer G to computer K is through edges G-A, A-E, and E-K. Figure 4C also illustrates the broadcast channel of Figure 4A with an added computer. Computer K was connected to the broadcast channel by edge pinning edges D-G and E-J to computer K. The diameter of this broadcast channel is, however, still two. Thus, the selection of neighbors impacts the diameter of the broadcast channel. To help minimize the diameter, the broadcast technique uses a random selection technique to identify the four neighbors of a computer in the seeking connection state. The random selection technique tends to distribute the connections to new seeking computers throughout the computers of the broadcast channel which may result in smaller overall diameters.

Broadcasting Through the Graph

As described above, each computer that is connected to the broadcast channel can broadcast messages onto the broadcast channel and does receive all messages that are broadcast on the broadcast channel. The computer that originates a message to be broadcast sends that message to each of its four neighbors using the internal connections. When a computer receives a broadcast message from a neighbor, it sends the message to its three other neighbors. Each computer on the broadcast channel, except the originating computer, will thus receive a copy of each broadcast message from each of its four neighbors. Each computer, however, only sends the first copy of the message that it receives to its neighbors and disregards subsequently received copies. Thus, the total number of copies of a message that is sent between the computers is 3N+1, where N is the number of computers connected to the broadcast channel. Each computer sends three copies of the message, except for the originating computer, which sends four copies of the message.

The redundancy of the message sending helps to ensure the overall reliability of the broadcast channel. Since each computer has four connections to the broadcast channel, if one computer fails during the broadcast of a message, its neighbors have three other connections through which they will receive copies of the broadcast message. Also, if the internal connection between two computers is slow, each computer has three other connections through which it may receive a copy of each message sooner.

Each computer that originates a message numbers its own messages sequentially. Because of the dynamic nature of the broadcast channel and because there are many possible connection paths between computers, the messages may be received out of order. For example, the distance between an originating computer and a certain receiving computer may be four. After sending the first message, the originating computer and receiving computer may become neighbors and thus the distance between them changes to one. The first message may have to travel a distance of four to reach the receiving computer. The second message only has to travel a distance of one. Thus, it is possible for the second message to reach the receiving computer before the first message.

When the broadcast channel is in a steady state (*i.e.*, no computers connecting or disconnecting from the broadcast channel), out-of-order messages are not a problem because each computer will eventually receive both messages and can queue messages until all earlier ordered messages are received. If, however, the broadcast channel is not in a steady state, then problems can occur. In particular, a computer may connect to the broadcast channel after the second message has already been received and forwarded on by its new neighbors. When a new neighbor eventually receives the first message, it sends the message to the newly connected computer. Thus, the newly connected computer will receive the first message, but will not receive the second message. If the newly connected computer needs to process the messages in order, it would wait indefinitely for the second message.

One solution to this problem is to have each computer queue all the messages that it receives until it can send them in their proper order to its neighbors. This solution, however, may tend to slow down the propagation of messages through the computers of the broadcast channel. Another solution that may have less impact on the propagation speed is to queue messages only at computers who are neighbors of the newly connected computers. Each already connected neighbor would forward messages as it receives them to its other neighbors who are not newly connected, but not to the newly connected neighbor. The already connected neighbor would only forward messages from each originating computer to the newly connected computer when it can ensure that no gaps in the messages from that originating computer will occur. In one embodiment, the already connected neighbor may track the highest sequence number of the messages already received and forwarded on from each originating computer. The already connected computer will send only higher numbered messages from the originating computers to the newly connected computer. Once all lower numbered messages have been received from all originating computers, then the already connected computer can treat the newly connected computer as its other neighbors and simply forward each message as it is received. In another embodiment, each computer may queue messages and only forwards to the newly connected computer those messages as the gaps are filled in. For example, a computer might receive messages 4 and 5 and then receive message 3. In such a case, the already connected computer would forward queue messages 4 and 5. When message 3 is finally received, the already connected computer will send messages 3, 4, and 5 to the newly connected computer. If messages 4 and 5 were sent to the newly connected computer before message 3, then the newly connected computer would process messages 4 and 5 and disregard message 3. Because the already connected computer queues messages 4 and 5, the newly connected computer will be able to process message 3. It is possible that a newly connected computer will receive a set of messages from an originating computer through one neighbor and then receive another set of message from the same originating computer through another neighbor. If the second set of messages contains a message that is ordered earlier than the messages of the first set received, then the newly connected computer may ignore that earlier ordered message if the computer already processed those later ordered messages.

Decomposing the Graph

A connected computer disconnects from the broadcast channel either in a planned or unplanned manner. When a computer disconnects in a planned manner, it sends a disconnect message to each of its four neighbors. The disconnect message includes a list that identifies the four neighbors of the disconnecting computer. When a neighbor receives the disconnect message, it tries to connect to one of the computers on the list. In one embodiment, the first computer in the list will try to connect to the second computer in the list, and the third computer in the list will try to connect to the fourth computer in the list. If a computer cannot connect (*e.g.*, the first and second computers are already connected), then the computers may try connecting in various other combinations. If connections cannot be established, each computer broadcasts a message that it needs to establish a connection with another computer. When a computer with an available internal port receives the message, it can then establish a connection with the computer that broadcast the message. Figures 5A-5D illustrate the disconnecting of a computer from the broadcast channel. Figure 5A illustrates the disconnecting of a computer from the broadcast channel in a planned manner. When computer H decides to disconnect, it sends its list of neighbors to each of its neighbors (computers A, E, F and I) and then disconnects from each of its neighbors. When computers A and I receive the message they establish a connection between them as indicated by the dashed line, and similarly for computers E and F.

When a computer disconnects in an unplanned manner, such as resulting from a power failure, the neighbors connected to the disconnected computer recognize the disconnection when each attempts to send its next message to the now disconnected computer. Each former neighbor of the disconnected computer recognizes that it is short one connection (*i.e.*, it has a hole or empty port). When a connected computer detects that one of its neighbors is now disconnected, it broadcasts a port connection request on the broadcast channel, which indicates that it has one internal port that needs a connection. The port connection request identifies the call-in port of the requesting computer. When a connected computer that is also short a connection receives the connection request, it communicates with the requesting computer through its external port to establish a connection between the two computers. Figure 5B illustrates the disconnecting of a computer from the broadcast channel in an unplanned manner. In this illustration, computer H has disconnected in an unplanned manner. When each of its neighbors, computers A, E, F, and I, recognizes the disconnection, each neighbor broadcasts a port connection request indicating that it needs to fill an empty port. As shown by the dashed lines, computers F and I and computers A and E respond to each other's requests and establish a connection.

It is possible that a planned or unplanned disconnection may result in two neighbors each having an empty internal port. In such a case, since they are neighbors, they are already connected and cannot fill their empty ports by connecting to each other. Such a condition is referred to as the "neighbors with empty ports" condition. Each neighbor broadcasts a port connection request when it detects that it has an empty port as described above. When a neighbor receives the port connection request from the other neighbor, it will recognize the condition that its neighbor also has an empty port. Such a condition may also occur when the broadcast channel is in the small regime. The condition can only be corrected when in the large regime. When in the small regime, each computer will have less than four neighbors. To detect this condition in the large regime, which would be a problem if not repaired, the first neighbor to receive the port connection request recognizes the condition and sends a condition check message to the other neighbor. The condition check message includes a list of the neighbors of the sending computer. When the receiving computer receives the list, it compares the list to its own list of neighbors. If the lists are different, then this condition has occurred in the large regime and repair is needed. To repair this condition, the receiving computer will send a condition repair request to one of the neighbors of the sending computer which is not already a neighbor of the receiving computer. When the computer receives the condition repair request, it disconnects from one of its neighbors (other than the neighbor that is involved with the condition) and connects to the computer that sent the condition repair request. Thus, one of the original neighbors involved in the condition will have had a port filled. However, two computers are still in need of a connection, the other original neighbor and the computer that is now disconnected from the computer that received the condition repair request. Those two computers send out port connection requests. If those two computers are not neighbors, then they will connect to each other when they receive the requests. If, however, the two computers are neighbors, then they repeat the condition repair process until two non-neighbors are in need of connections.

It is possible that the two original neighbors with the condition may have the same set of neighbors. When the neighbor that receives the condition check message determines that the sets of neighbors are the same, it sends a condition double check message to one of its neighbors other than the neighbor who also has the condition. When the computer receives the condition double check message, it determines whether it has the same set of neighbors as the sending computer. If so, the broadcast channel is in the small regime and the condition is not a problem. If the set of neighbors are different, then the computer that received the condition double check message sends a condition check message to the original neighbors with the condition. The computer that receives that condition check message directs one of it neighbors to connect to one of the original neighbors with the condition by sending a condition repair message. Thus, one of the original neighbors with the condition will have its port filled.

Figure 5C illustrates the neighbors with empty ports condition. In this illustration, computer H disconnected in an unplanned manner, but computers F and I responded to the port connection request of the other and are now connected together. The other former neighbors of computer H, computers A and E, are already neighbors, which gives rise to the neighbors with empty ports condition. In this example, computer E received the port connection request from computer A, recognized the possible condition, and sent (since they are neighbors via the internal connection) a condition check message with a list of its neighbors to computer A. When computer A received the list, it recognized that computer E has a different set of neighbor (*i.e.*, the broadcast channel is in the large regime). Computer A selected computer D, which is a neighbor of computer E and sent it a condition repair request. When computer D received the condition repair request, it disconnected from one of its neighbors (other than computer E), which is computer G in this example. Computer D then connected to computer A. Figure 5D illustrates two computers that are not neighbors who now have empty ports. Computers E and G now have empty ports and are not currently neighbors. Therefore, computers E and G can connect to each other.

Figures 5E and 5F further illustrate the neighbors with empty ports condition. Figure 5E illustrates the neighbors with empty ports condition in the small regime. In this example, if computer E disconnected in an unplanned manner, then each computer broadcasts a port connection request when it detects the disconnect. When computer A receives the port connection request form computer B, it detects the neighbors with empty ports condition and sends a condition check message to computer B. Computer B recognizes that it has the same set of neighbors (computer C and D) as computer A and then sends a condition double check message to computer C. Computer C recognizes that the broadcast channel is in the small regime because is also has the same set of neighbors as computers A and B, computer C may then broadcast a message indicating that the broadcast channel is in the small regime.

Figure 5F illustrates the situation of Figure 5E when in the large regime. As discussed above, computer C receives the condition double check message from computer B. In this case, computer C recognizes that the broadcast channel is in the large regime because it has a set of neighbors that is different from computer B. The edges extending up from computer C and D indicate connections to other computers. Computer C then sends a condition check message to computer B. When computer B receives the condition check message, it sends a condition repair message to one of the neighbors of computer C. The computer that receives the condition repair message disconnects from one of its neighbors, other than computer C, and tries to connect to computer B and the neighbor from which it disconnected tries to connect to computer A.

Port Selection

As described above, the TCP/IP protocol designates ports above number 2056 as user ports. The broadcast technique uses five user port numbers on each computer: one external port and four internal ports. Generally, user ports cannot be statically allocated to an application program because other applications programs executing on the same computer may use conflicting port numbers. As a result, in one embodiment, the computers connected to the broadcast channel dynamically allocate their port numbers. Each computer could simply try to locate the lowest number unused port on that computer and use that port as the call-in port. A seeking computer, however, does not know in advance the call-in port number of the portal computers when the port numbers are dynamically allocated. Thus, a seeking computer needs to dial ports of a portal computer starting with the lowest port number when locating the call-in port of a portal computer. If the portal computer is connected to (or attempting to connect to) the broadcast channel, then the seeking computer would eventually find the call-in port. If the portal computer is not connected, then the seeking computer would eventually dial every user port. In addition, if each application program on a computer tried to allocate low-ordered port numbers, then a portal computer may end up with a high-numbered port for its call-in port because many of the low-ordered port numbers would be used by other application programs. Since the dialing of a port is a relatively slow process, it would take the seeking computer a long time to locate the call-in port of a portal computer. To minimize this time, the broadcast technique uses a port ordering algorithm to identify the port number order that a portal computer should use when finding an available port for its call-in port. In one embodiment, the broadcast technique uses a hashing algorithm to identify the port order. The algorithm preferably distributes the ordering of the port numbers randomly through out the user port number space and only selects each port number once. In addition, every time the algorithm is executed on any computer for a given channel type and channel instance, it generates the same port ordering. As described below, it is possible for a computer to be connected to multiple broadcast channels that are uniquely identified by channel type and channel instance. The algorithm may be "seeded" with channel type and channel instance in order to generate a unique ordering of port numbers for each broadcast channel. Thus, a seeking computer will dial the ports of a portal computer in the same order as the portal computer used when allocating its call-in port.

If many computers are at the same time seeking connection to a broadcast channel through a single portal computer, then the ports of the portal computer may be busy when called by seeking computers. The seeking computers would typically need to keep on redialing a busy port. The process of locating a call-in port may be significantly slowed by such redialing. In one embodiment, each seeking computer may each reorder the first few port numbers generated by the hashing algorithm. For example, each seeking computer could randomly reorder the first eight port numbers generated by the hashing algorithm. The random ordering could also be weighted where the first port number generated by the hashing algorithm would have a 50% chance of being first in the reordering, the second port number would have a 25% chance of being first in the reordering, and so on. Because the seeking computers would use different orderings, the likelihood of finding a busy port is reduced. For example, if the first eight port numbers are randomly selected, then it is possible that eight seeking computers could be simultaneously dialing ports in different sequences which would reduce the chances of dialing a busy port.

Locating a Portal Computer

Each computer that can connect to the broadcast channel has a list of one or more portal computers through which it can connect to the broadcast channel. In one embodiment, each computer has the same set of portal computers. A seeking computer locates a portal computer that is connected to the broadcast channel by successively dialing the ports of each portal computer in the order specified by an algorithm. A seeking computer could select the first portal computer and then dial all its ports until a call-in port of a computer that is fully connected to the broadcast channel is found. If no call-in port is found, then the seeking computer would select the next portal computer and repeat the process until a portal computer with such a call-in port is found. A problem with such a seeking technique is that all user ports of each portal computer are dialed until a portal computer fully connected to the broadcast channel is found. In an alternate embodiment, the seeking computer selects a port number according to the algorithm and then dials each portal computer at that port number. If no acceptable call-in port to the broadcast channel is found, then the seeking computer selects the next port number and repeats the process. Since the call-in ports are likely allocated at lower-ordered port numbers, the seeking computer first dials the port numbers that are most likely to be call-in ports of the broadcast channel. The seeking computers may have a maximum search depth, that is the number of ports that it will dial when seeking a portal computer that is fully connected. If the seeking computer exhausts its search depth, then either the broadcast channel has not yet been established or, if the seeking computer is also a portal computer, it can then establish the broadcast channel with itself as the first fully connected computer.

When a seeking computer locates a portal computer that is itself not fully connected, the two computers do not connect when they first locate each other because the broadcast channel may already be established and accessible through a higher-ordered port number on another portal computer. If the two seeking computers were to connect to each other, then two disjoint broadcast channels would be formed. Each seeking computer can share its experience in trying to locate a portal computer with the other seeking computer. In particular, if one seeking computer has searched all the portal computers to a depth of eight, then the one seeking computer can share that it has searched to a depth of eight with another seeking computer. If that other seeking computer has searched to a depth of, for example, only four, it can skip searching through depths five through eight and that other seeking computer can advance its searching to a depth of nine.

In one embodiment, each computer may have a different set of portal computers and a different maximum search depth. In such a situation, it may be possible that two disjoint broadcast channels are formed because a seeking computer cannot locate a fully connected port computer at a higher depth. Similarly, if the set of portal computers are disjoint, then two separate broadcast channels would be formed.

Identifying Neighbors for a Seeking Computer

As described above, the neighbors of a newly connecting computer are preferably selected randomly from the set of currently connected computers. One advantage of the broadcast channel, however, is that no computer has global knowledge of the broadcast channel. Rather, each computer has local knowledge of itself and its neighbors. This limited local knowledge has the advantage that all the connected computers are peers (as far as the broadcasting is concerned) and the failure of any one computer (actually any three computers when in the 4-regular and 4-connect form) will not cause the broadcast channel to fail. This local knowledge makes it difficult for a portal computer to randomly select four neighbors for a seeking computer.

To select the four computers, a portal computer sends an edge connection request message through one of its internal connections that is randomly selected. The receiving computer again sends the edge connection request message through one of its internal connections that is randomly selected. This sending of the message corresponds to a random walk through the graph that represents the broadcast channel. Eventually, a receiving computer will decide that the message has traveled far enough to represent a randomly selected computer. That receiving computer will offer the internal connection upon which it received the edge connection request message to the seeking computer for edge pinning. Of course, if either of the computers at the end of the offered internal connection are already neighbors of the seeking computer, then the seeking computer cannot connect through that internal connection. The computer that decided that the message has traveled far enough will detect this condition of already being a neighbor and send the message to a randomly selected neighbor.

In one embodiment, the distance that the edge connection request message travels is established by the portal computer to be approximately twice the estimated diameter of the broadcast channel. The message includes an indication of the distance that it is to travel. Each receiving computer decrements that distance to travel before sending the message on. The computer that receives a message with a distance to travel that is zero is considered to be the randomly selected computer. If that randomly selected computer cannot connect to the seeking computer (*e.g.*, because it is already connected to it), then that randomly selected computer forwards the edge connection request to one of its neighbors with a new distance to travel. In one embodiment, the forwarding computer toggles the new distance to travel between zero and one to help prevent two computers from sending the message back and forth between each other.

Because of the local nature of the information maintained by each computer connected to the broadcast channel, the computers need not generally be aware of the diameter of the broadcast channel. In one embodiment, each message sent through the broadcast channel has a distance traveled field. Each computer that forwards a message increments the distance traveled field. Each computer also maintains an estimated diameter of the broadcast channel. When a computer receives a message that has traveled a distance that indicates that the estimated diameter is too small, it updates its estimated diameter and broadcasts an estimated diameter message. When a computer receives an estimated diameter message that indicates a diameter that is larger than its own estimated diameter, it updates its own estimated diameter. This estimated diameter is used to establish the distance that an edge connection request message should travel.

External Data Representation

The computers connected to the broadcast channel may internally store their data in different formats. For example, one computer may use 32-bit integers, and another computer may use 64-bit integers. As another example, one computer may use ASCII to represent text and another computer may use Unicode. To allow communications between heterogeneous computers, the messages sent over the broadcast channel may use the XDR ("eXternal Data Representation") format.