

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.

5 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.

#### 10 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.  
15 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.

20 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  
25 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  
30 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.

#### 25 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.

15 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

Figure 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 implemented 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.

Figure 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 the 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.

#### A Distributed Game Environment

In one embodiment, a game environment is implemented using broadcast channels. The game environment is provided by a game application program executing on

each player's computer that interacts with a broadcaster component. Each player joins a game (e.g., a first person shooter game) by connecting to the broadcast channel on which the game is played. Each time a player takes an action in the game a message representing that action is broadcast on the game's broadcast channel. In addition, a player may send messages (e.g., strategy information) to one or more other players by broadcasting a message. When the game application program receives an indication of an action, either received on the broadcast channel or generated by the player at this computer, it updates its current state of the game. The game may terminate when one of the players reaches a certain score, defeats all other players, all players leave the game, and so on.

To facilitate the creation of games for the game environment, an application programming interface ("API") is provided to assist game developers. The API may provide high-level game functions that would be used by most types of first person shooter games. For example, the API may include functions for indicating that a player has moved to a new position, for shooting in a certain direction, for reporting a score, for announcing the arrival and departure of players, for sending a message to another player, and so on.

The game environment may provide a game web site through which players can view the state of current games and register new games. The game web server would include a mapping between each game and the broadcast channel on which the game is to be played. When joining a game, the user would download the broadcaster component and the game application program from the web server. The user would also download the description of the game, which may include the graphics for the game. The web server would also provide the channel type and channel instance associated with the game and the identification of the portal computers for the game. The game environment may also have a game monitor computer that connects to each game, monitors the activity of the game, and reports the activity to the web server. With this activity information, the web server can provide information on the current state (e.g., number of players) of each game.

The game environment may also be used for games other than first person shooter games. For example, a variation of a society simulation game can be played where players sign up for different roles. If a role is unfulfilled or a player in that role is not playing, then an automated player can take over the role.

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>i.e.</i> , 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
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

Figures 8-34 are flow diagrams illustrating the processing of the broadcaster component in one embodiment. Figure 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.

Figure 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.

Figure 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.

Figure 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.

5 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

10 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

15 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

20 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

25 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.

Figure 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

30 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 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.

Figure 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.

Figure 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 1412, else the routine continues at block 1413. 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.

Figure 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.

Figure 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.

Figure 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 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.

Figure 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 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.

Figure 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.

Figure 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.

Figure 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.

Figure 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.

Figure 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.

Figure 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.

Figure 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.

Figure 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.

Figure 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.

Figure 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.

Figure 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.

Figure 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.

Figures 32-34 are flow diagrams illustrating the processing of messages associated with the neighbors with empty ports condition. Figure 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.

Figure 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 holes, 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.

Figure 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.

## CLAIMS

1           1.     A computer network for providing a game environment for a plurality of  
2 participants, each participant having connections to at least three neighbor participants,  
3 wherein an originating participant sends data to the other participants by sending the data  
4 through each of its connections to its neighbor participants and wherein each participant  
5 sends data that it receives from a neighbor participant to its other neighbor participants.

1           2.     The computer network of claim 1 wherein each participant is connected  
2 to 4 other participants.

1           3.     The computer network of claim 1 wherein each participant is connected  
2 to an even number of other participants.

1           4.     The computer network of claim 1 wherein the network is  $m$ -regular,  
2 where  $m$  is the number of neighbor participants of each participant.

1           5.     The computer network of claim 1 wherein the network is  $m$ -connected,  
2 where  $m$  is the number of neighbor participants of each participant.

1           6.     The computer network of claim 1 wherein the network is  $m$ -regular and  
2  $m$ -connected, where  $m$  is the number of neighbor participants of each participant.

1           7.     The computer network of claim 1 wherein all the participants are peers.

1           8.     The computer network of claim 1 wherein the connections are peer-to-  
2 peer connections.



1           9.     The computer network of claim 1 wherein the connections are TCP/IP  
2 connections.

1           10.    The computer network of claim 1 wherein each participant is a process  
2 executing on a computer.

1           11.    The computer network of claim 1 wherein a computer hosts more than  
2 one participant.

1           12.    The computer network of claim 1 wherein each participant sends to each  
2 of its neighbors only one copy of the data.

1           13.    The computer network of claim 1 wherein the interconnections of  
2 participants form a broadcast channel for a game of interest.

1           14.    A distributed game system comprising:  
2 a plurality of broadcast channels, each broadcast channel for playing a  
3 game;  
4 means for identifying a broadcast channel for a game of interest; and  
5 means for connecting to the identified broadcast channel.

1           15.    The distributed game system of claim 14 wherein means for identifying  
2 a game of interest includes accessing a web server that maps games to corresponding  
3 broadcast channel.

1           16.    The distributed game system of claim 14 wherein a broadcast channel is  
2 formed by player computers that are each interconnected to at least three other computers.

3004-8009 US 00

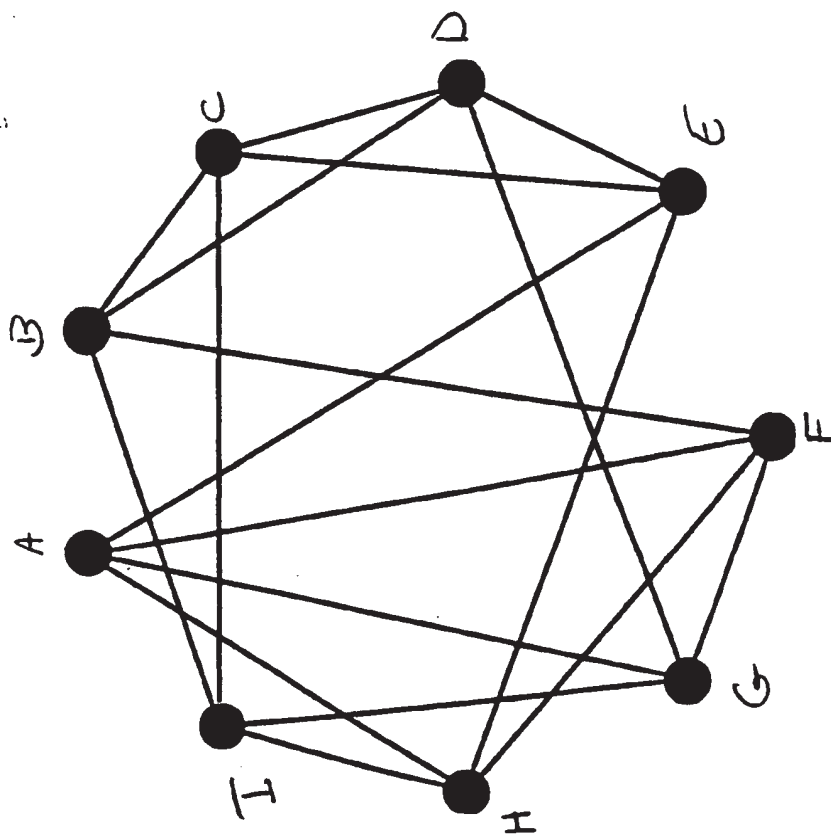


Fig 1

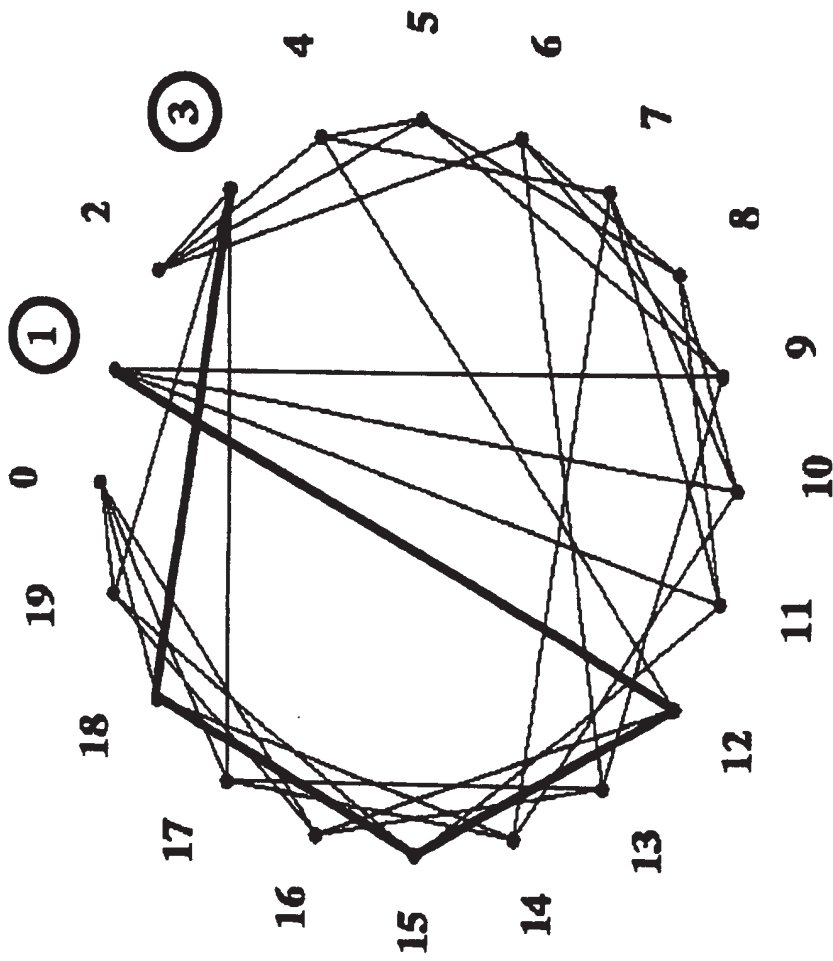


Fig 2

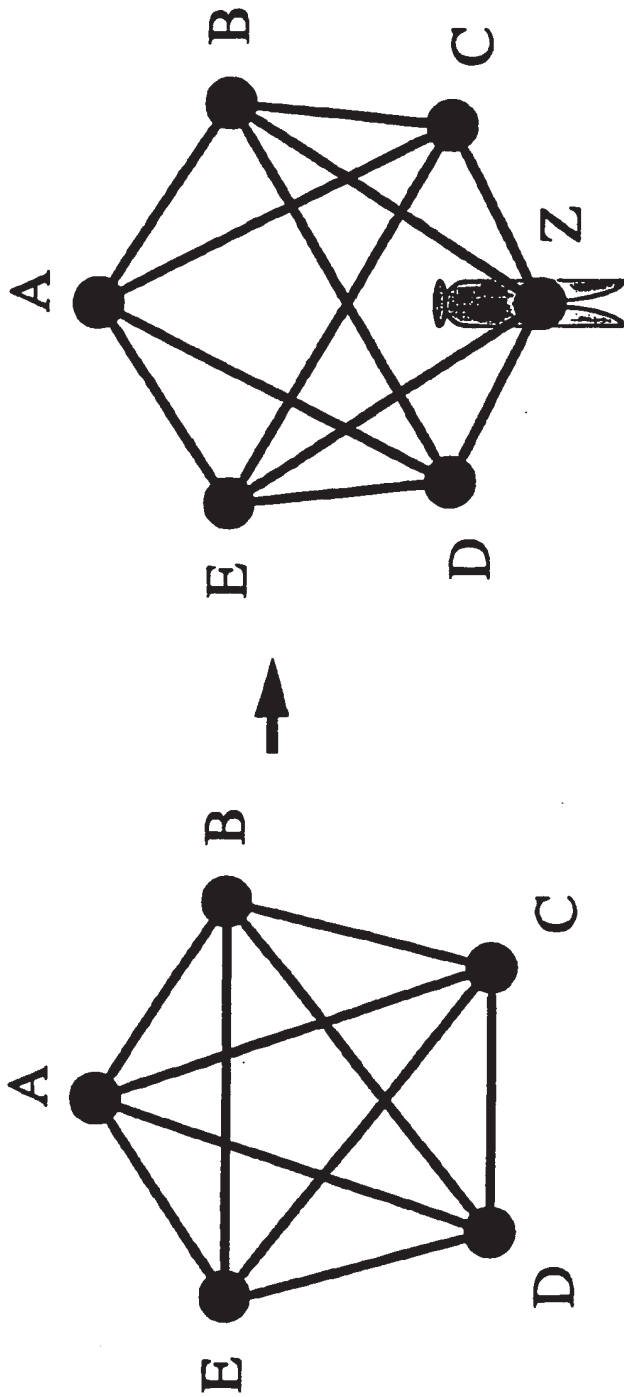


Fig 3B

Fig 3A

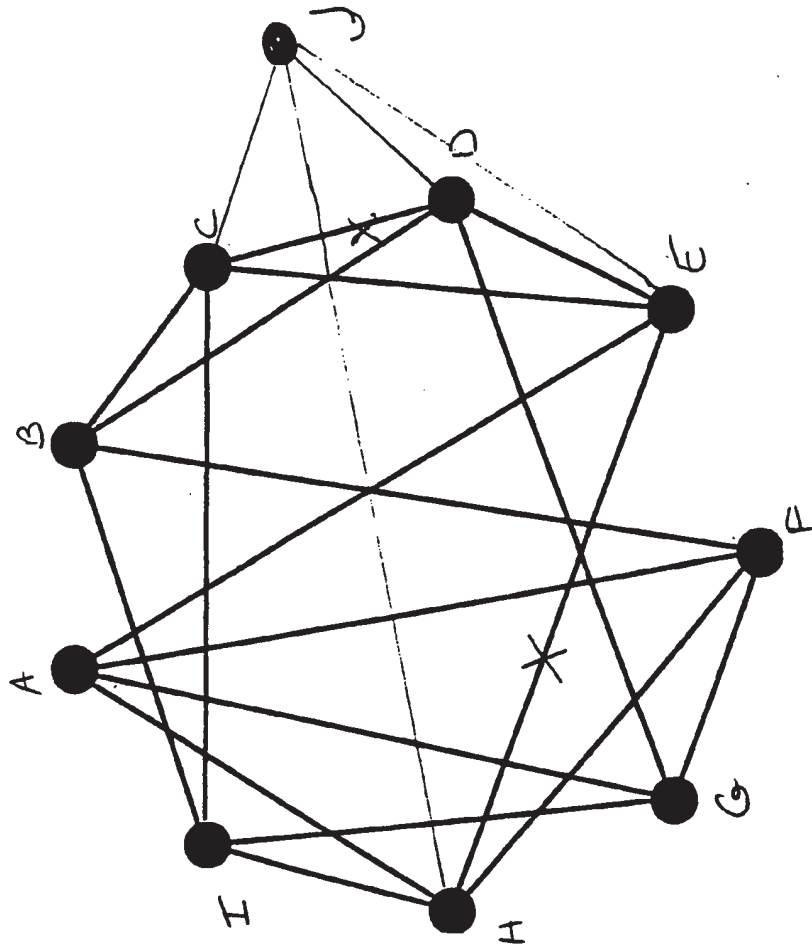


Fig 4A

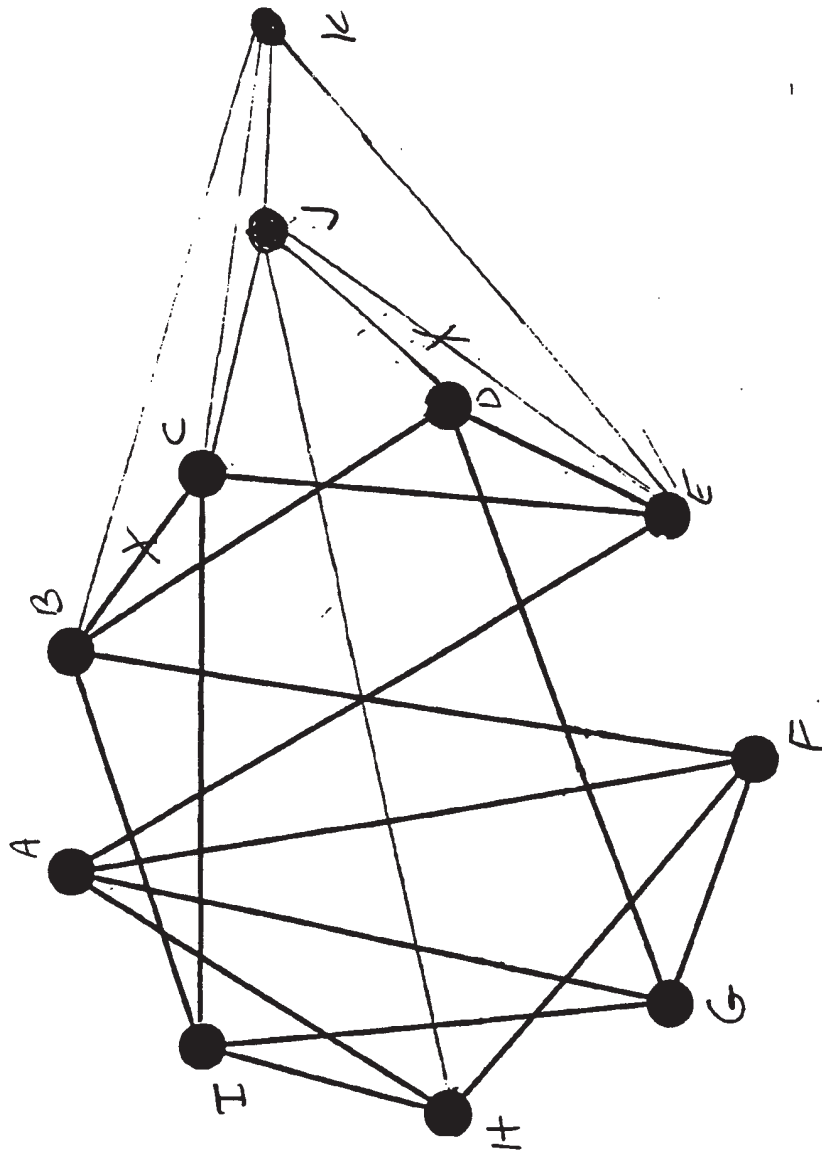


FIG 4B

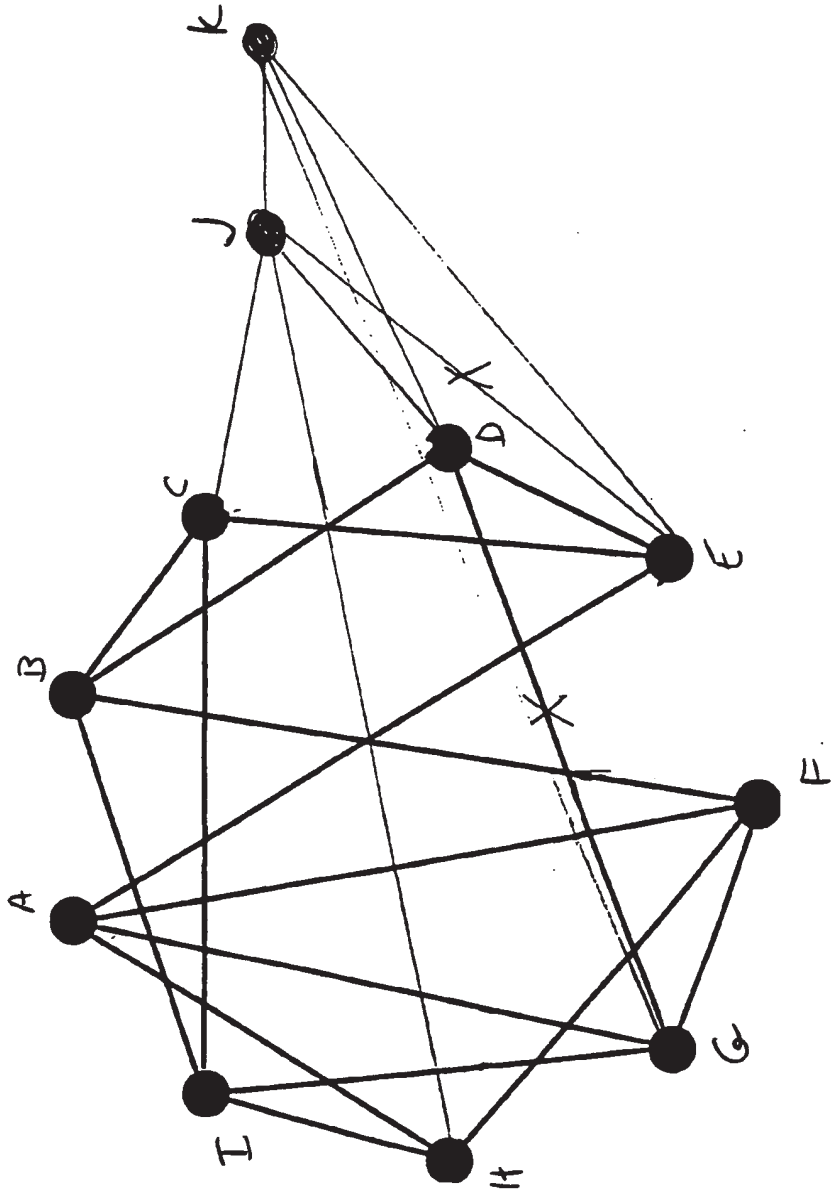


Fig 4C

11

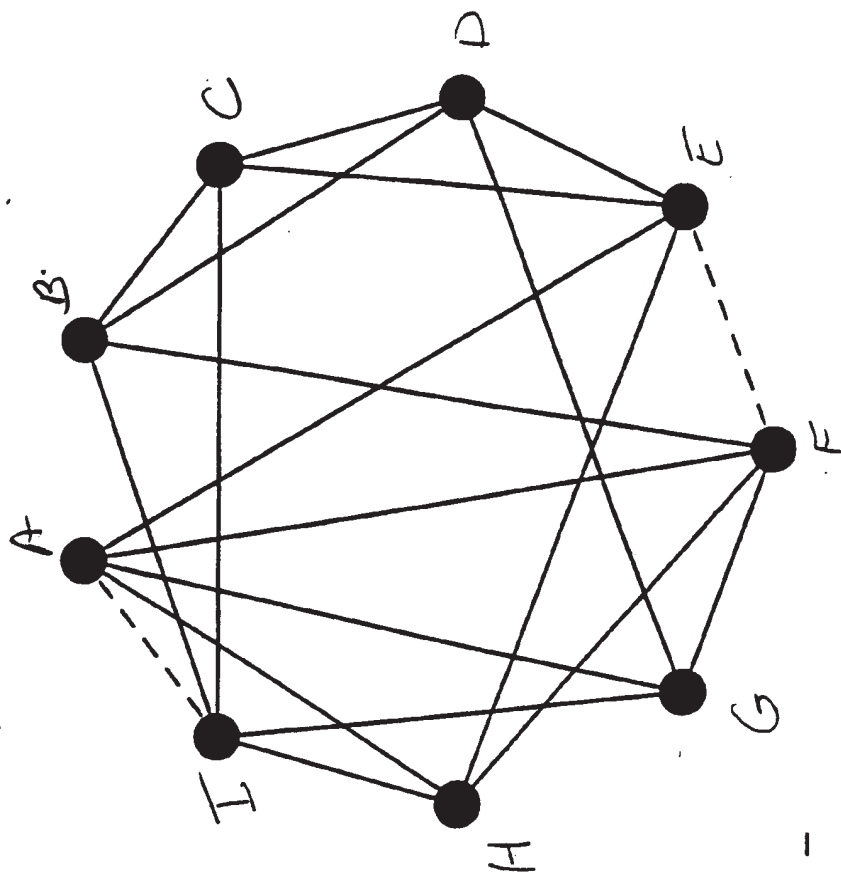


Fig 5A



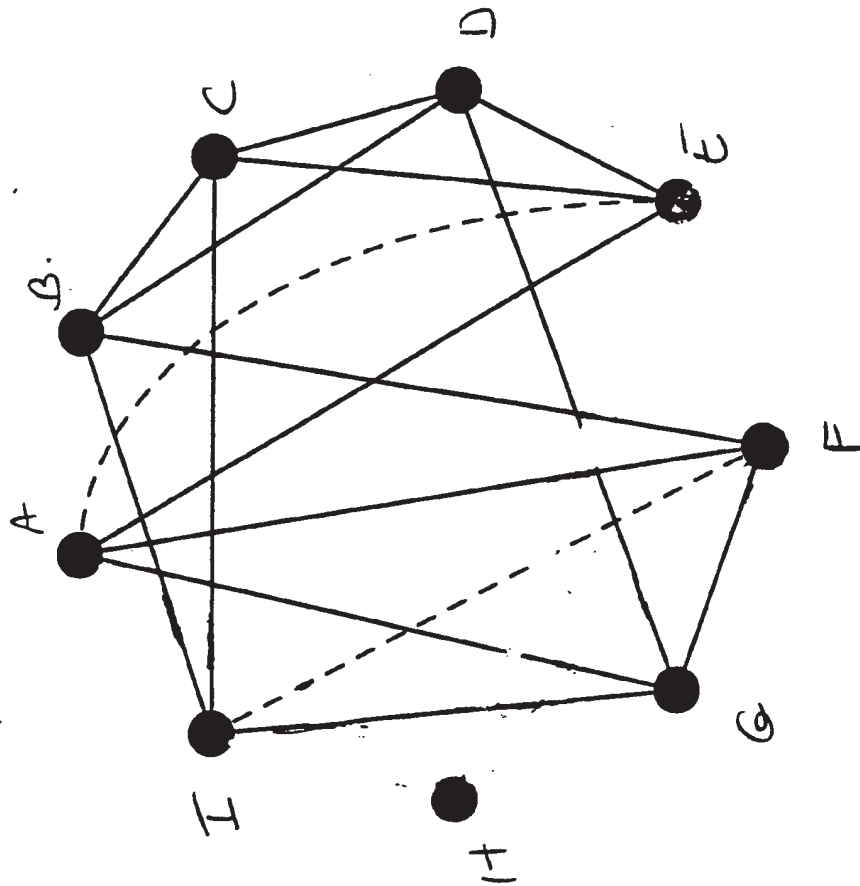


Fig 5B

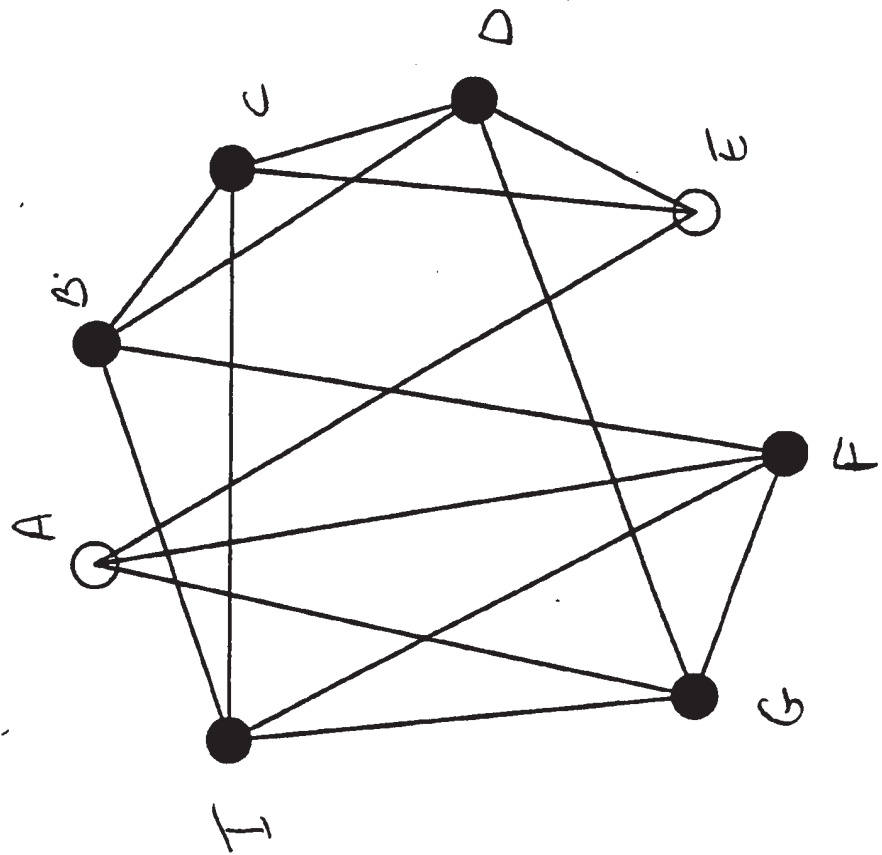


FIG 5C

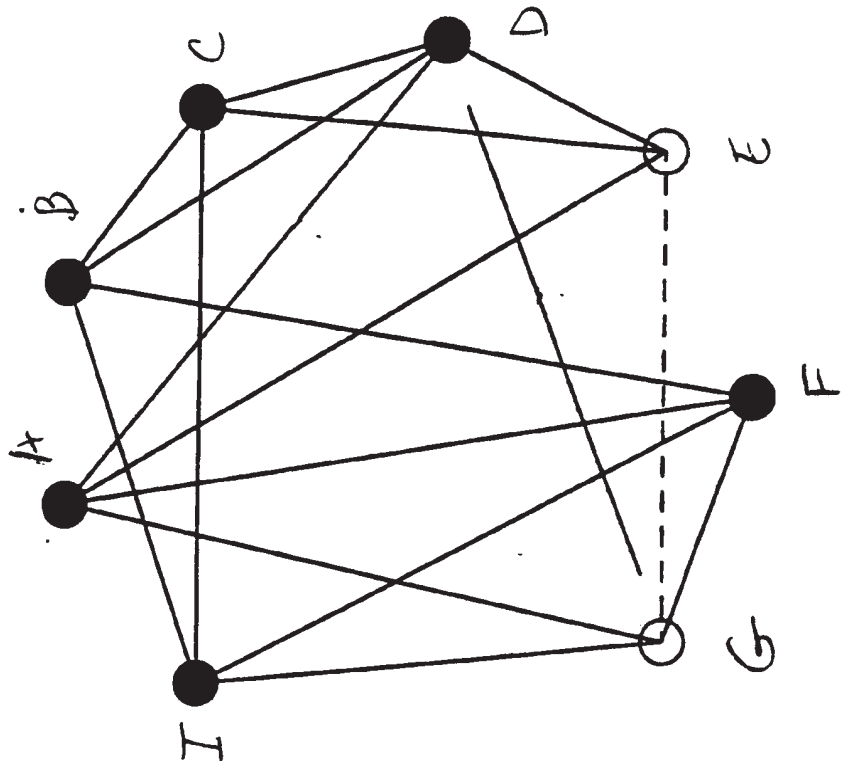


Fig 5D

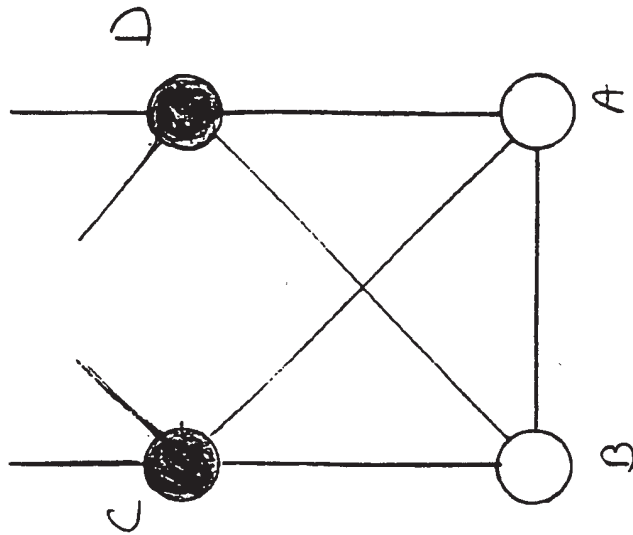


FIG 5F

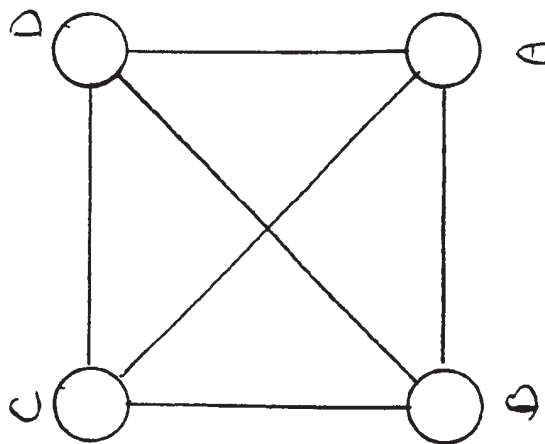
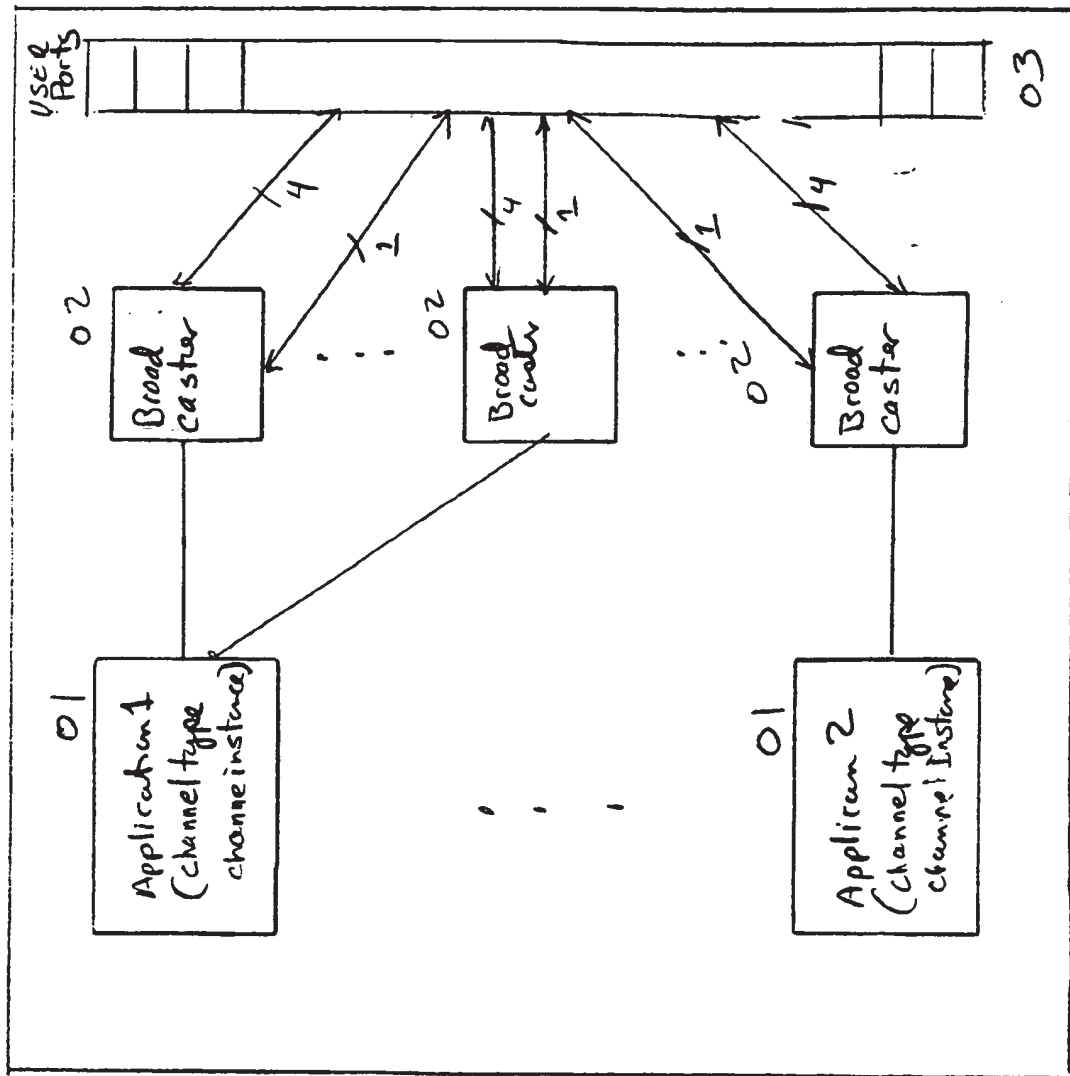


FIG 5E

00



F.84

00

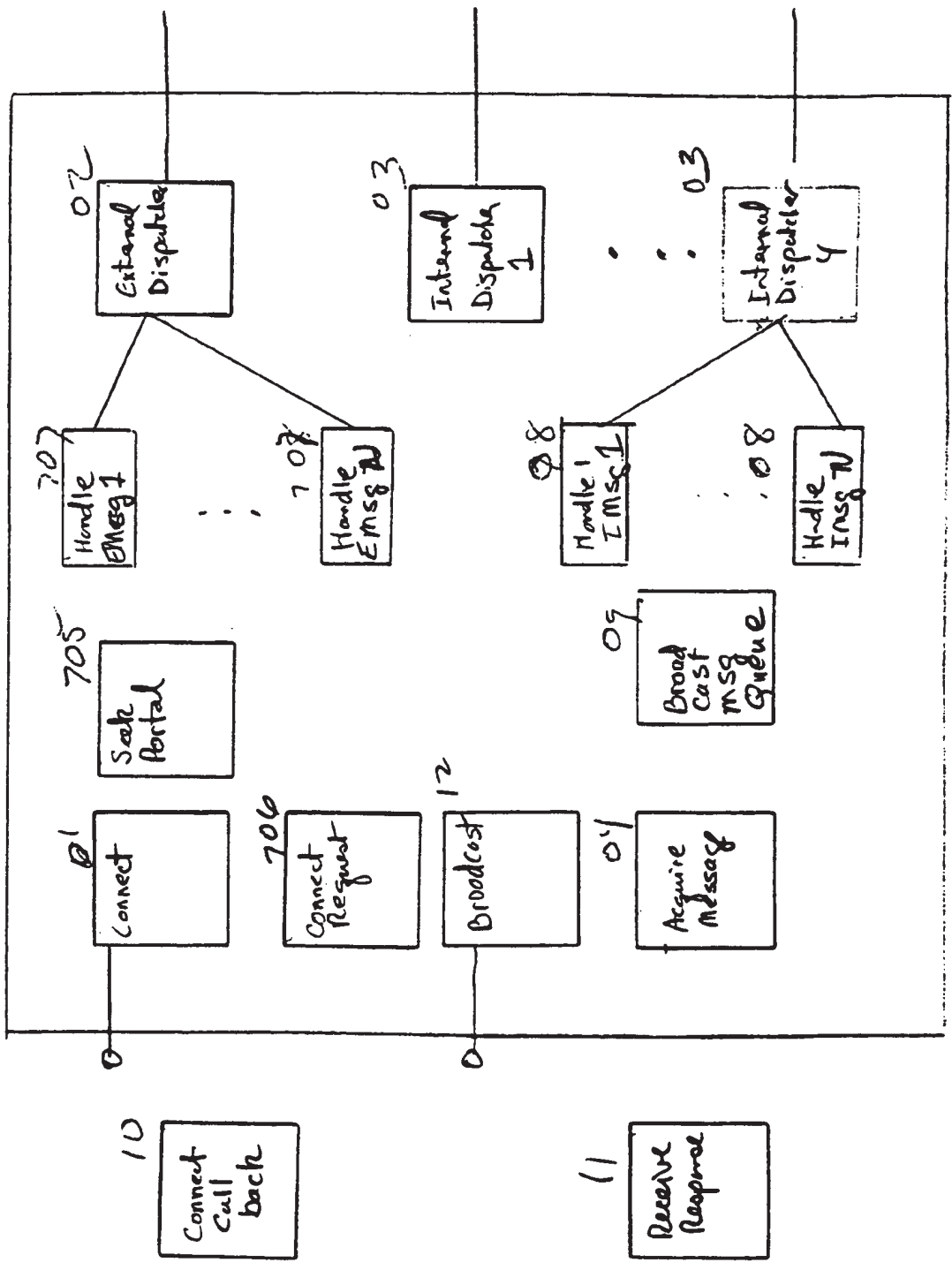
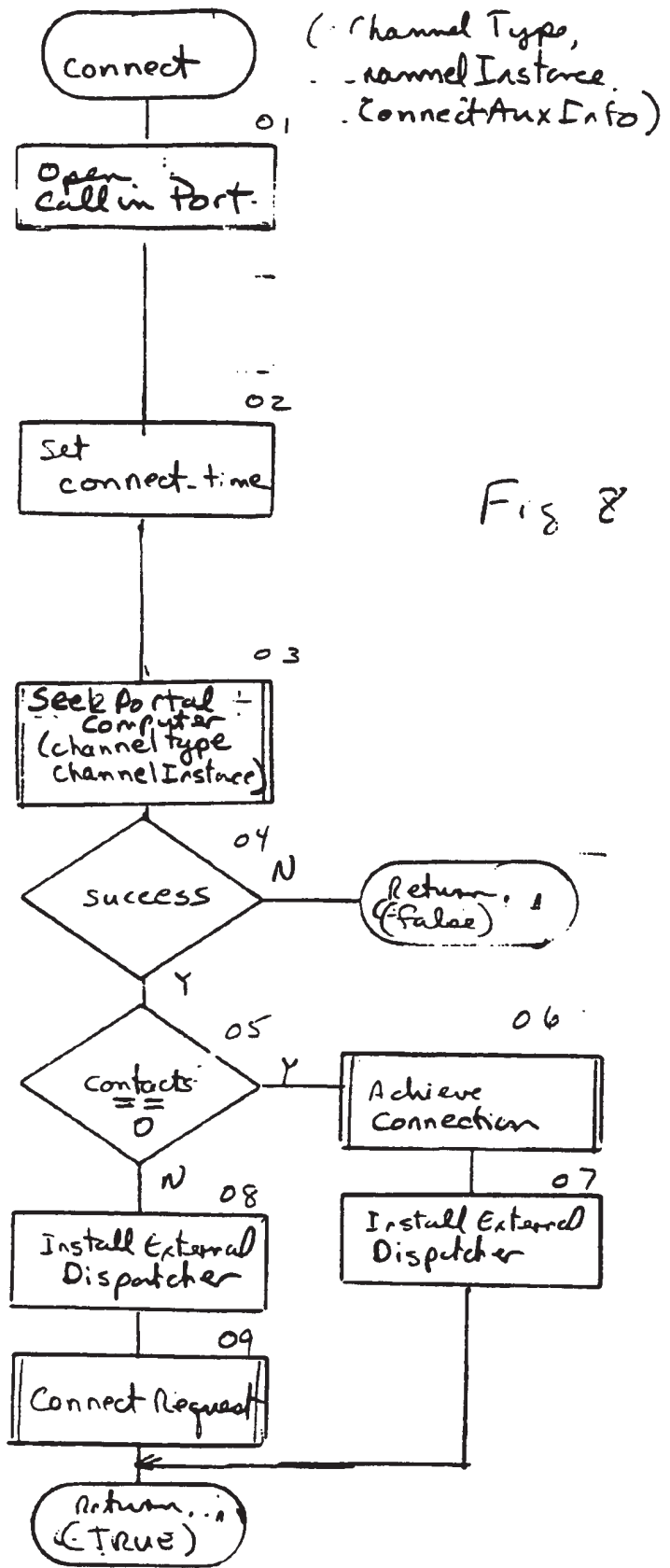


Figure 7

P--



P4  
quest

channel type  
channel instance

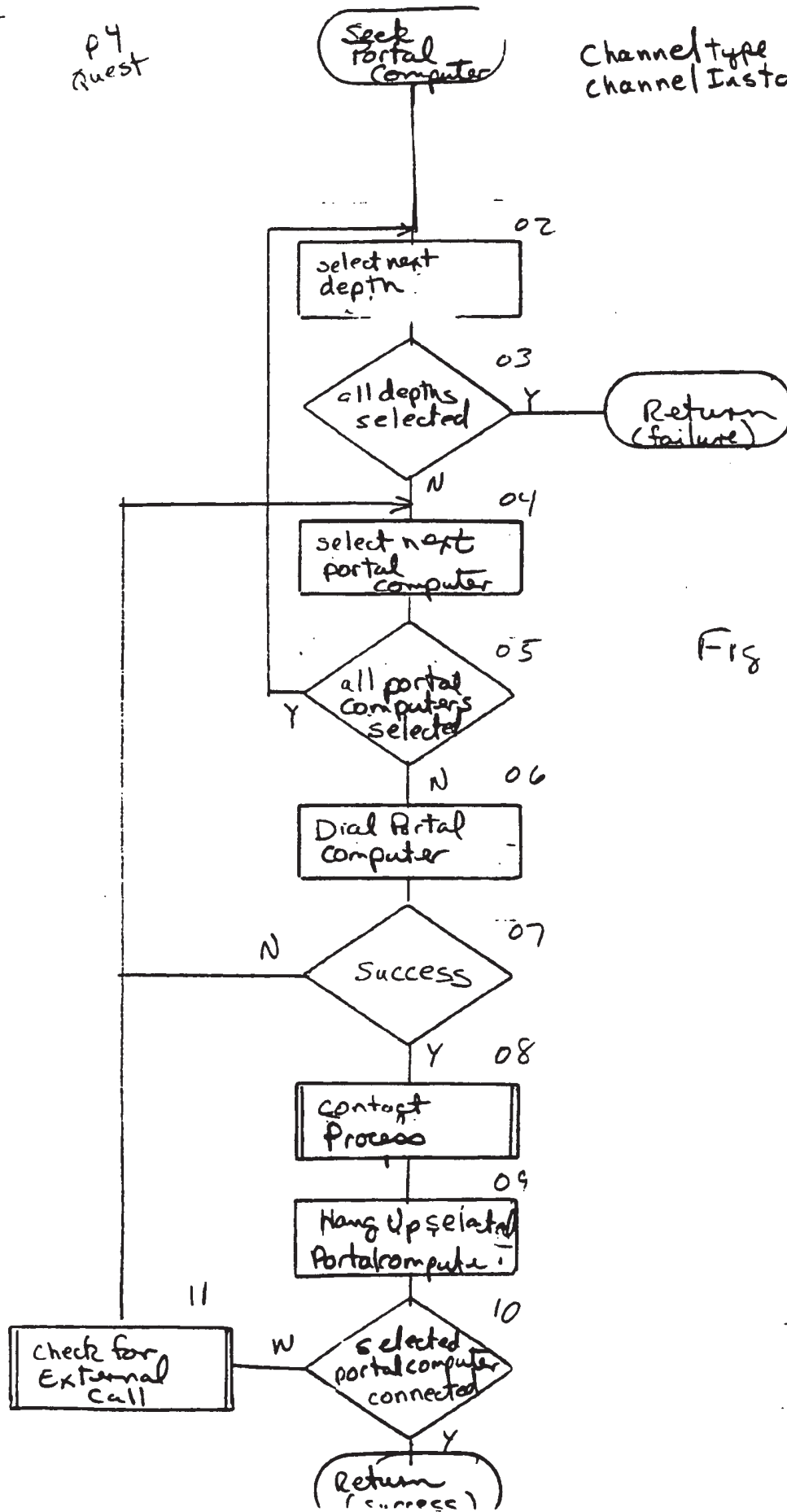


Fig 9



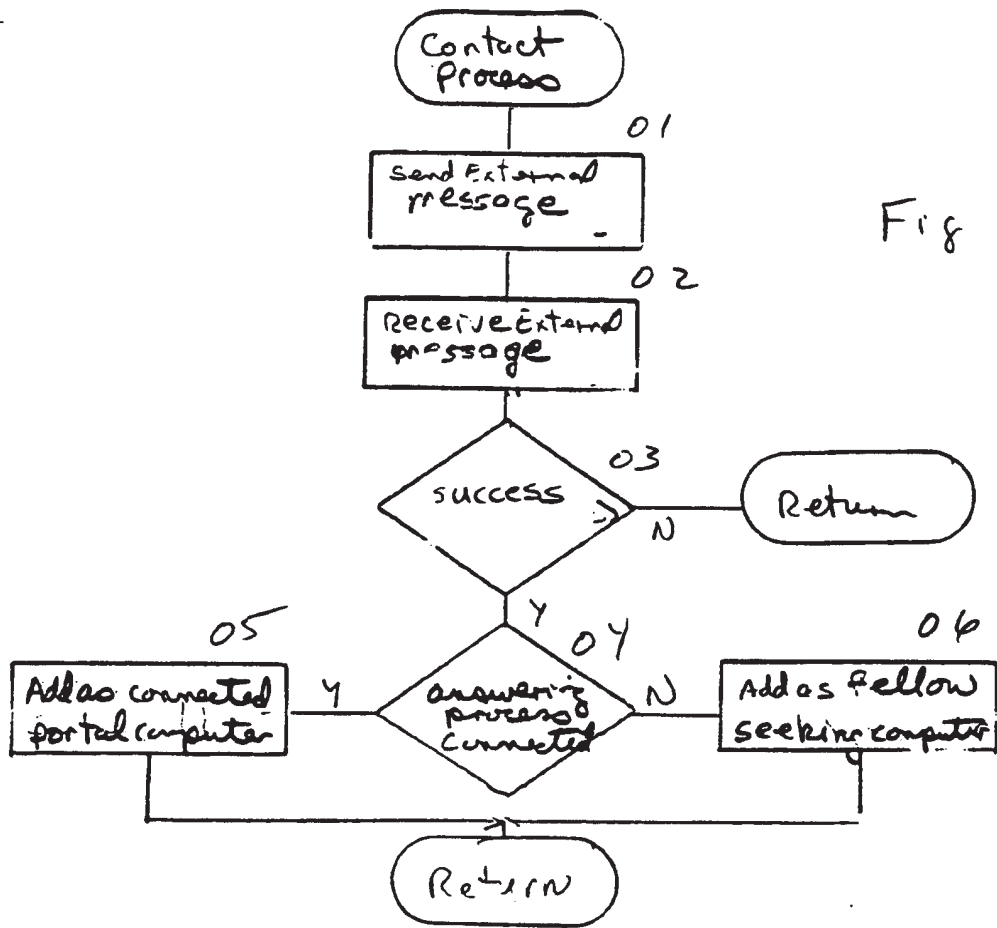
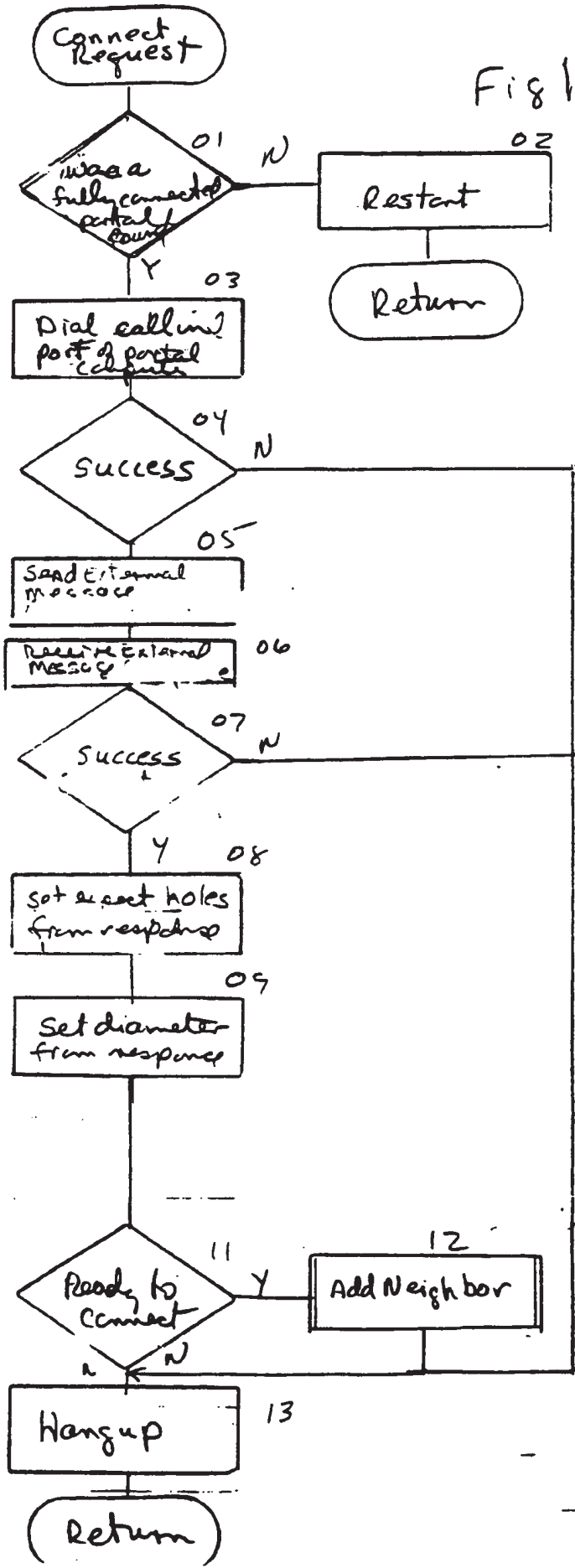


Fig 10

p23

Fig 11



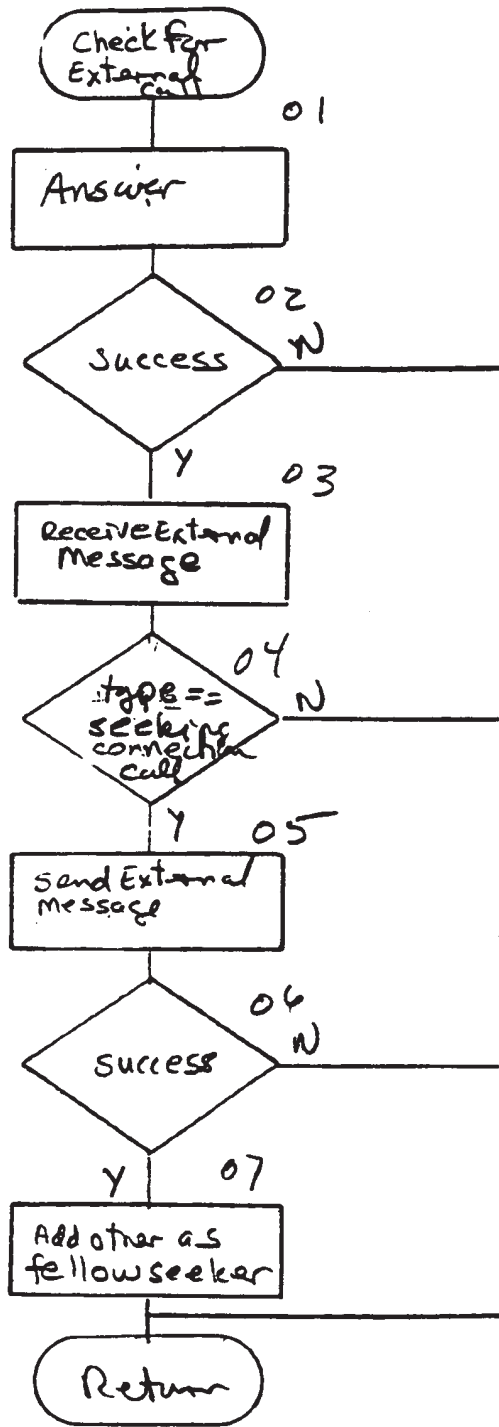


Fig 12

Achieve  
N/A  
22

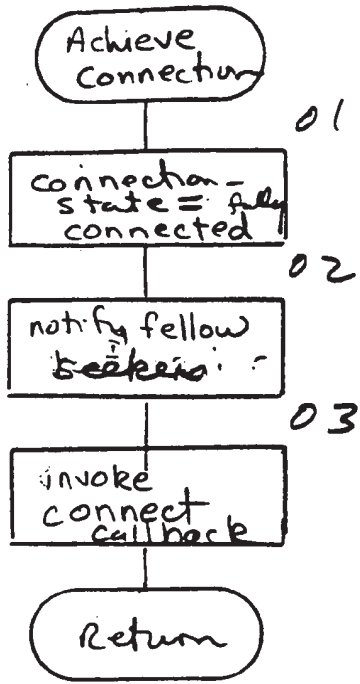
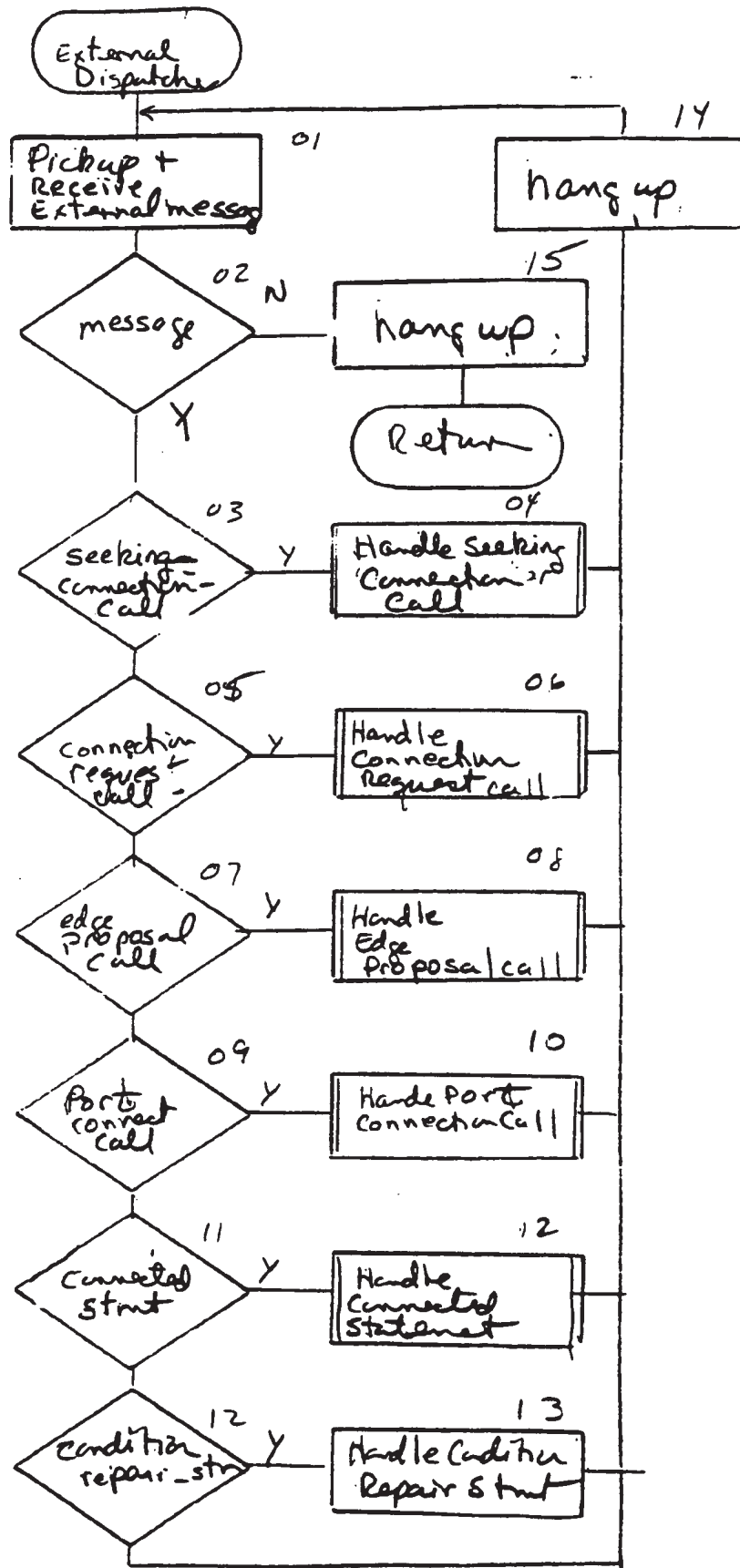


Fig 13

p7

Fig 14



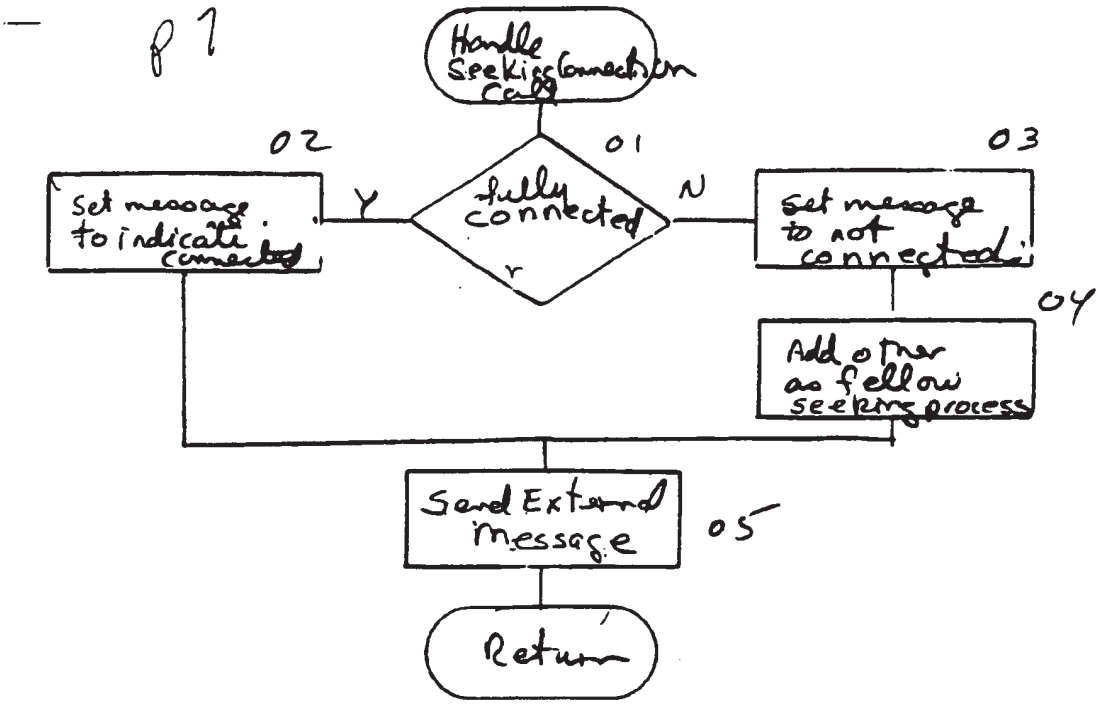
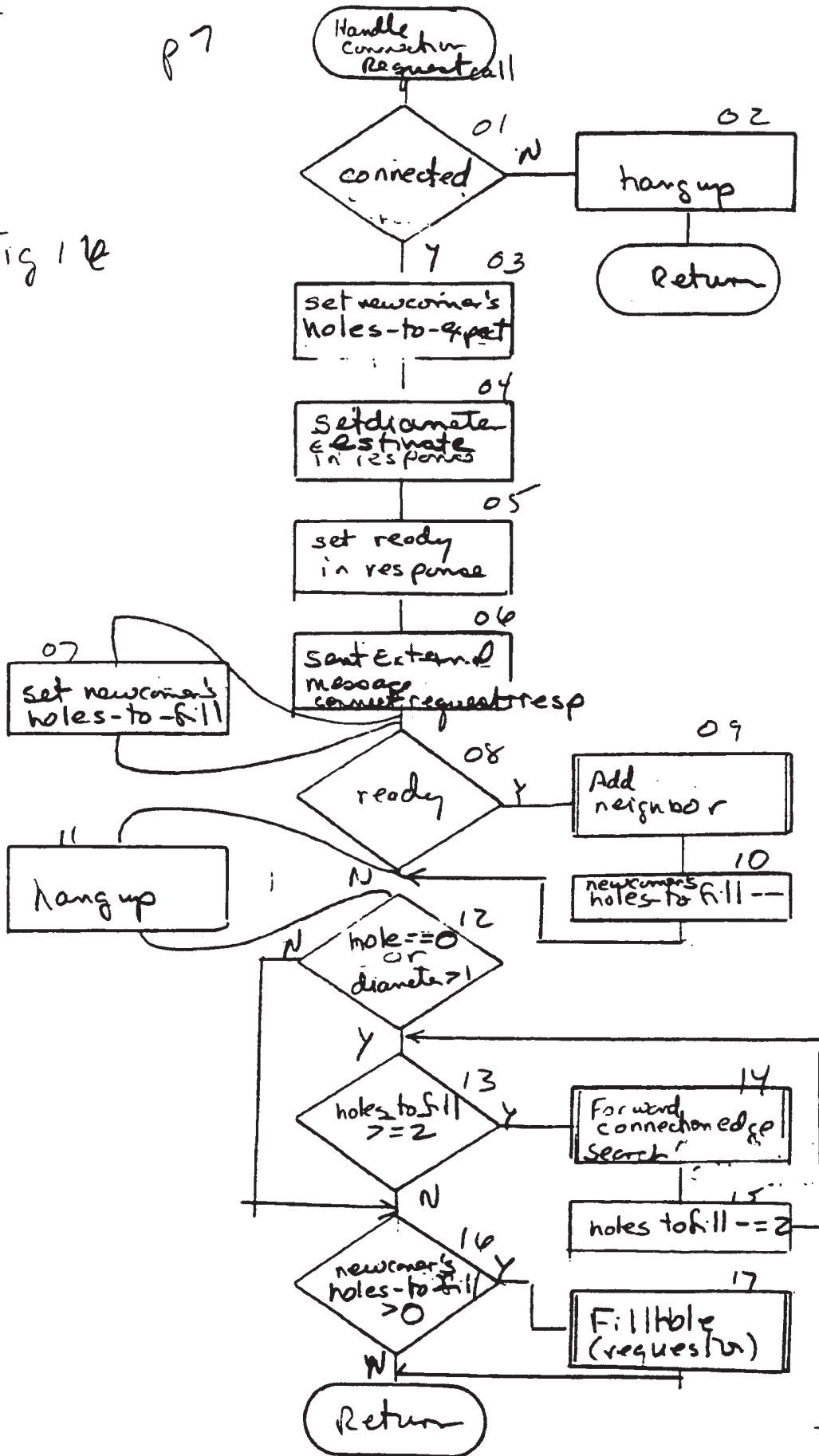


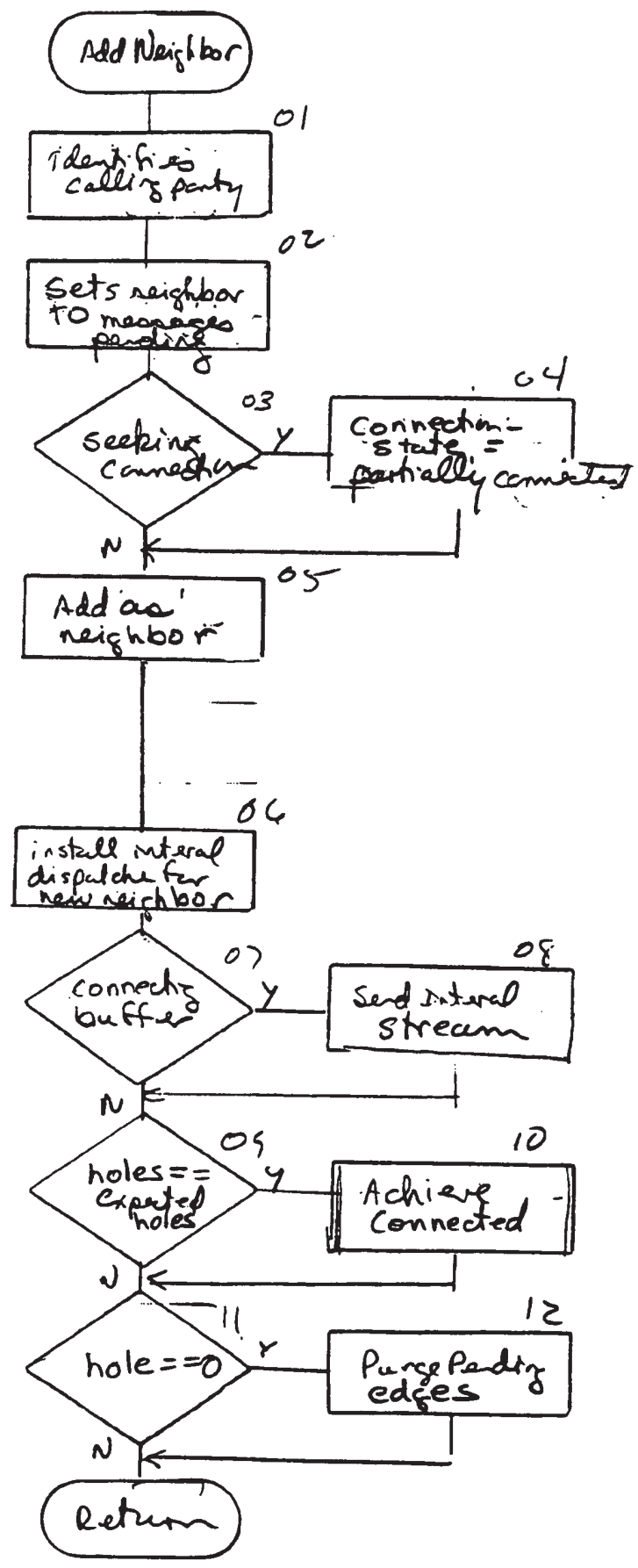
Fig 15

Fig 12



p9

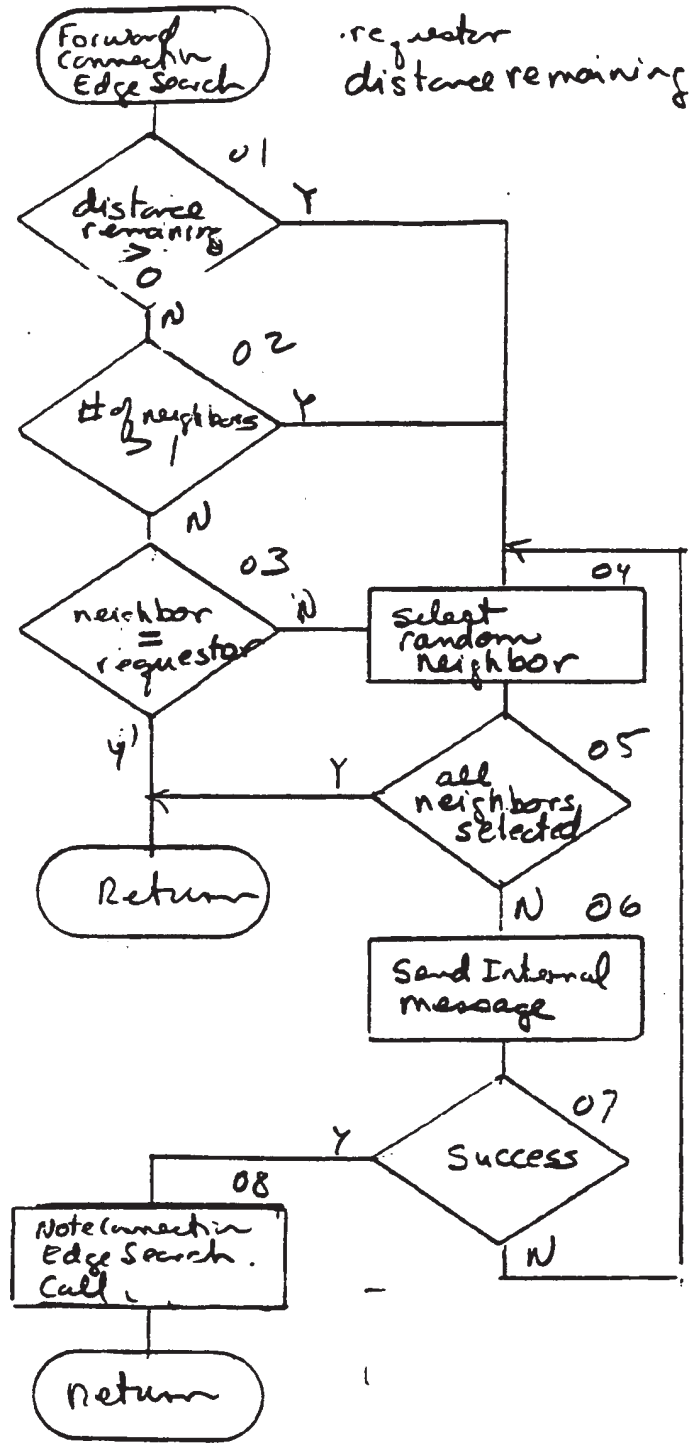
Fig 17





15

Fig 18



requester distance remaining

p. 8

inMessage  
outMessage

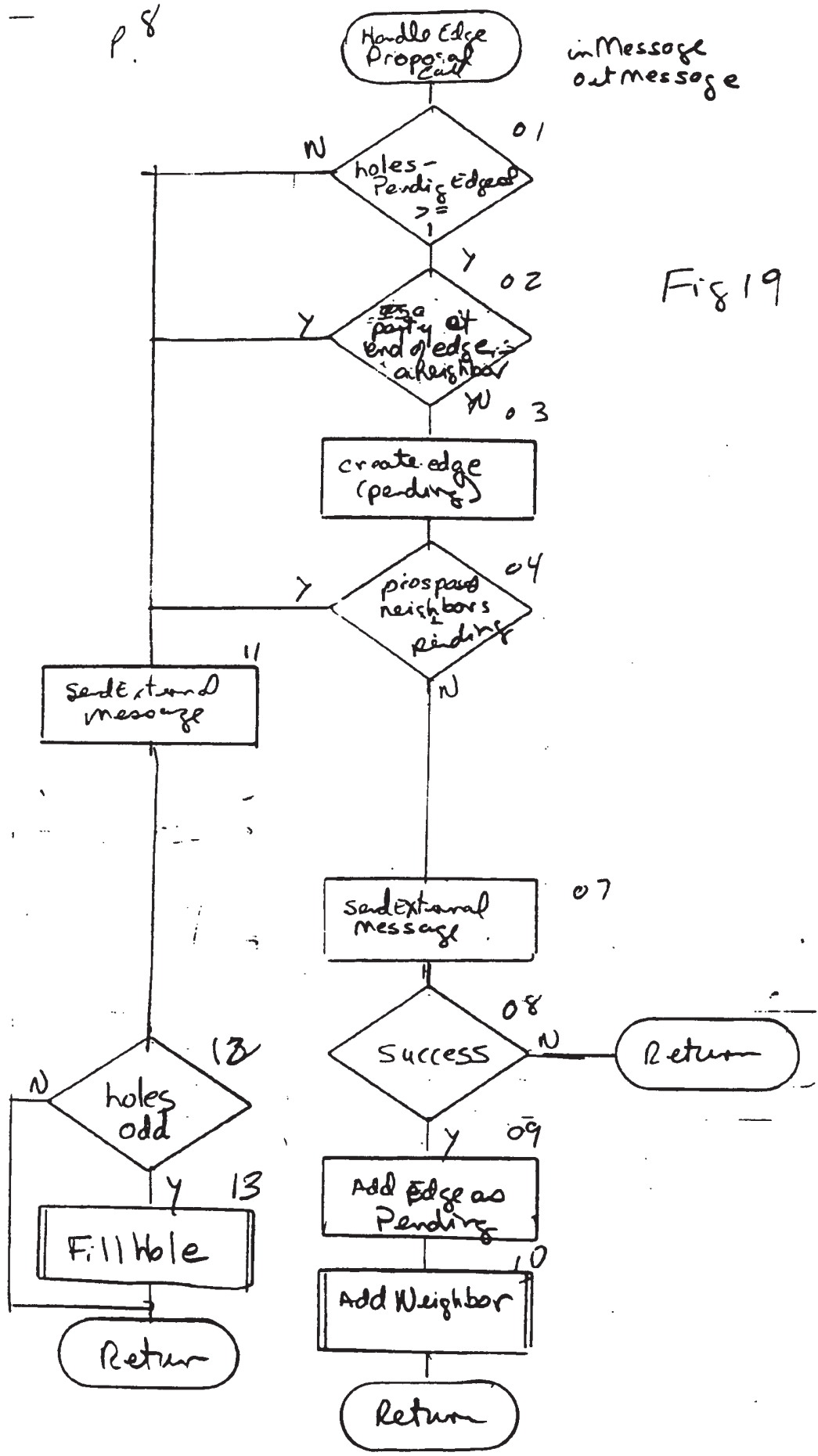


Fig 19

Fig 20

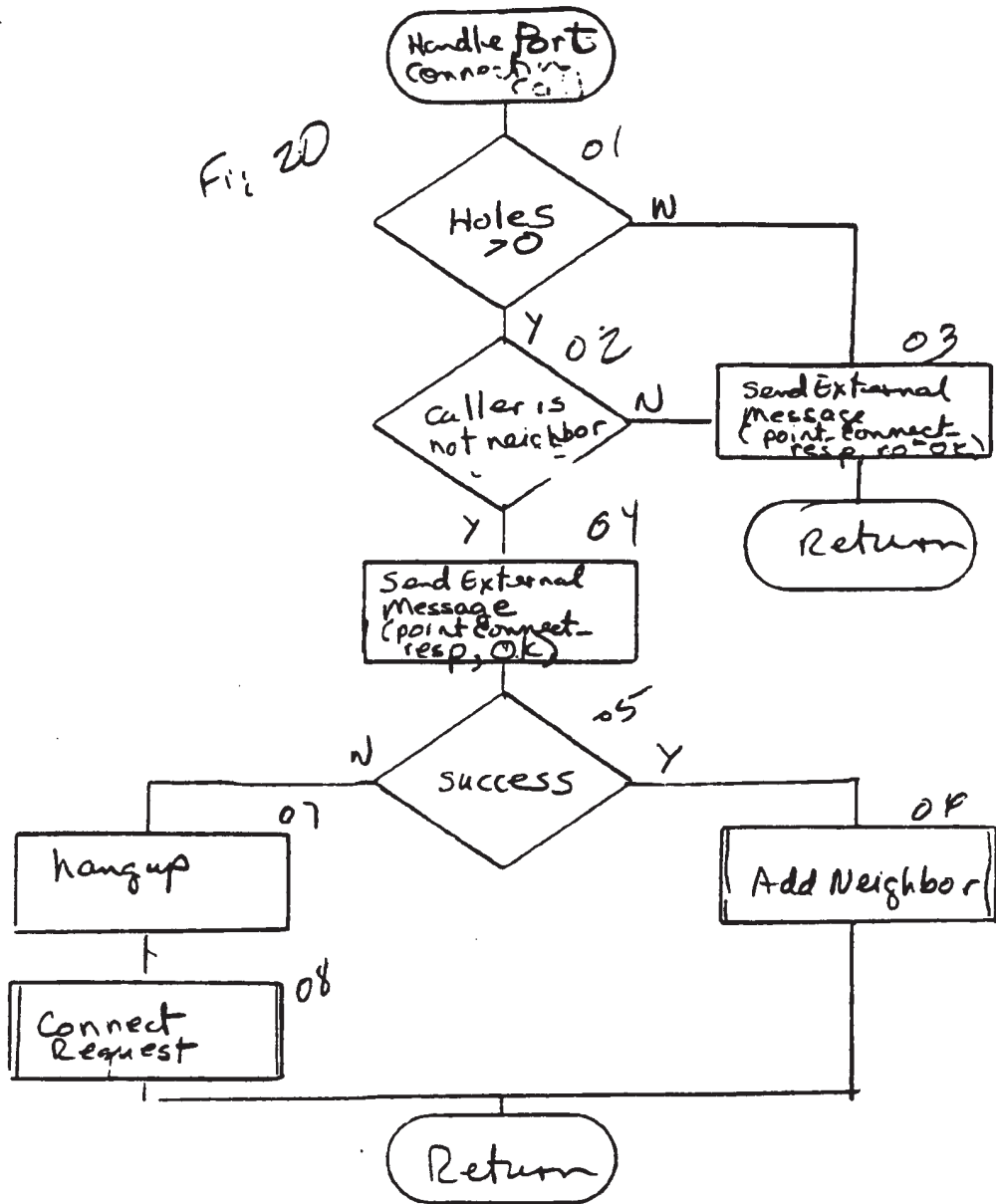
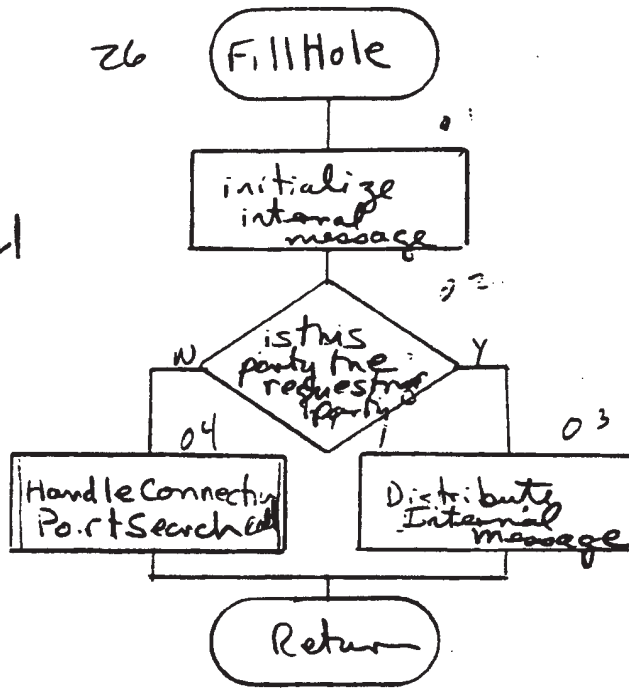


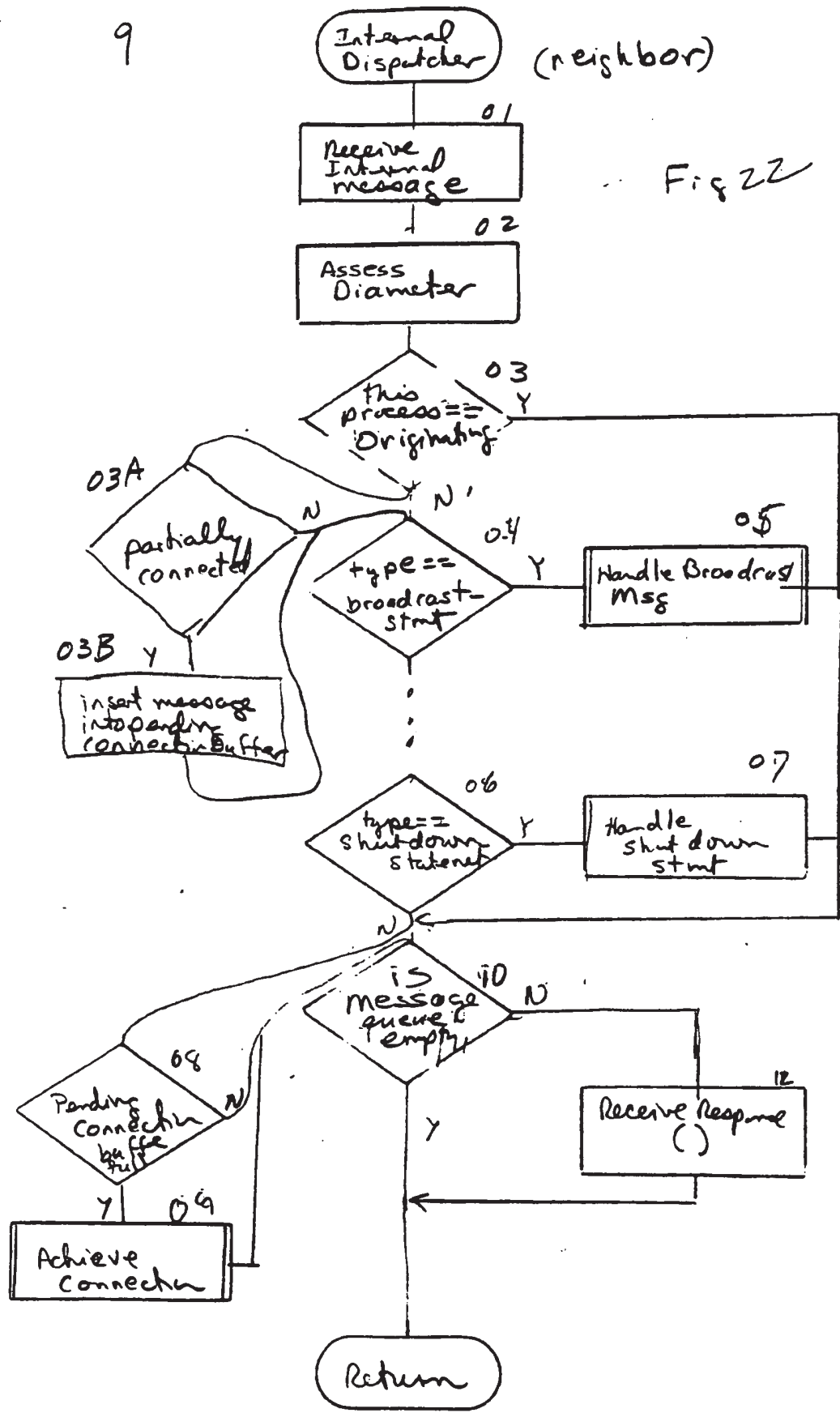
FIG 21

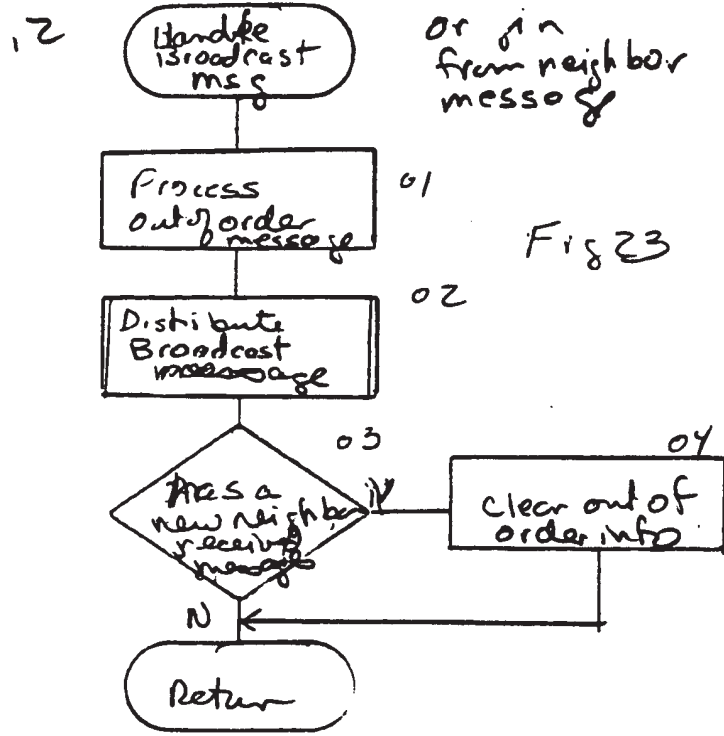


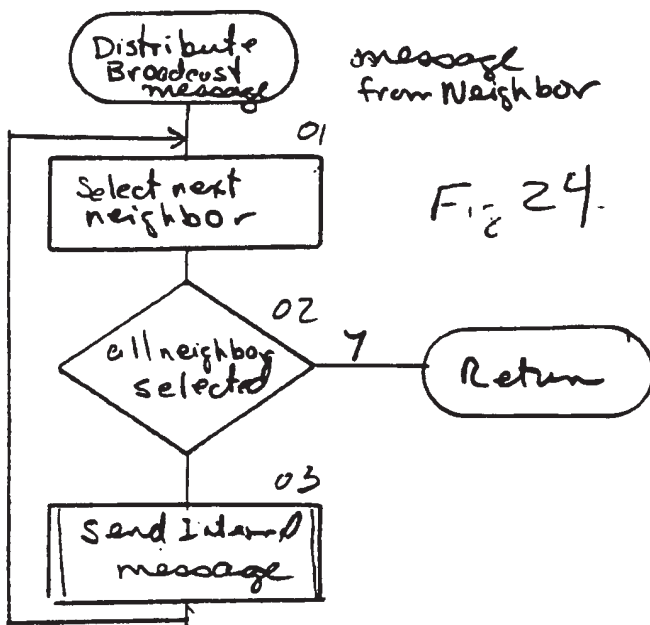
9

Internal Dispatcher (neighbor)

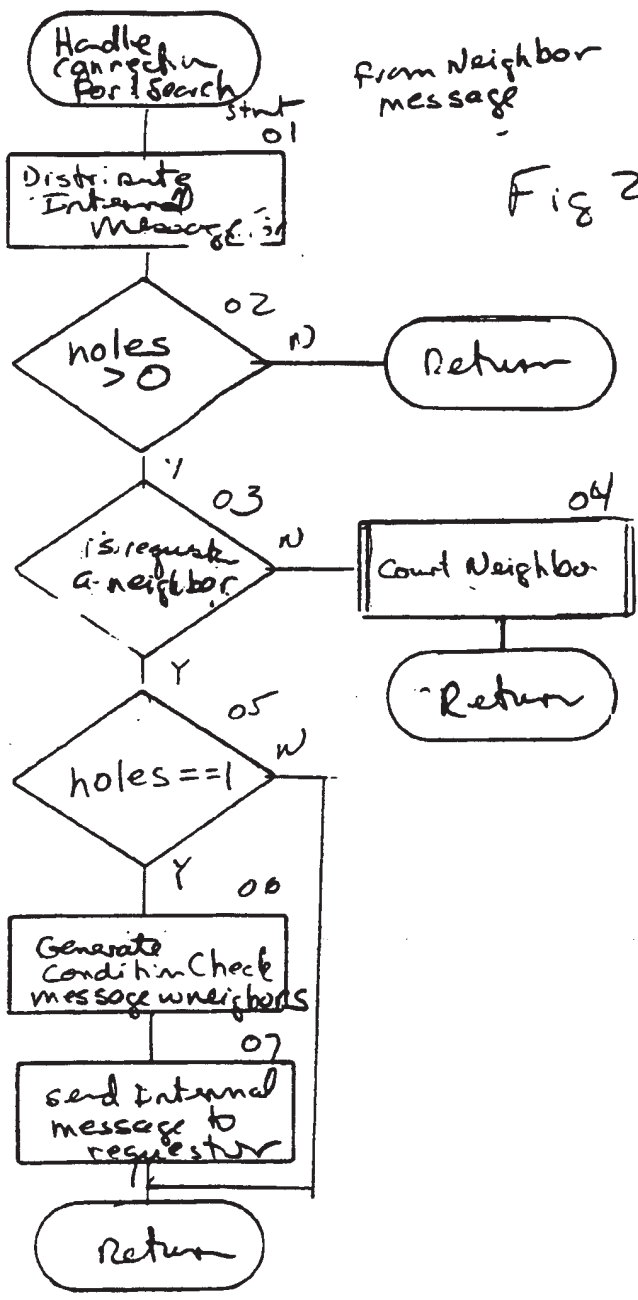
Fig 22







13



from neighbor message

Fig 26

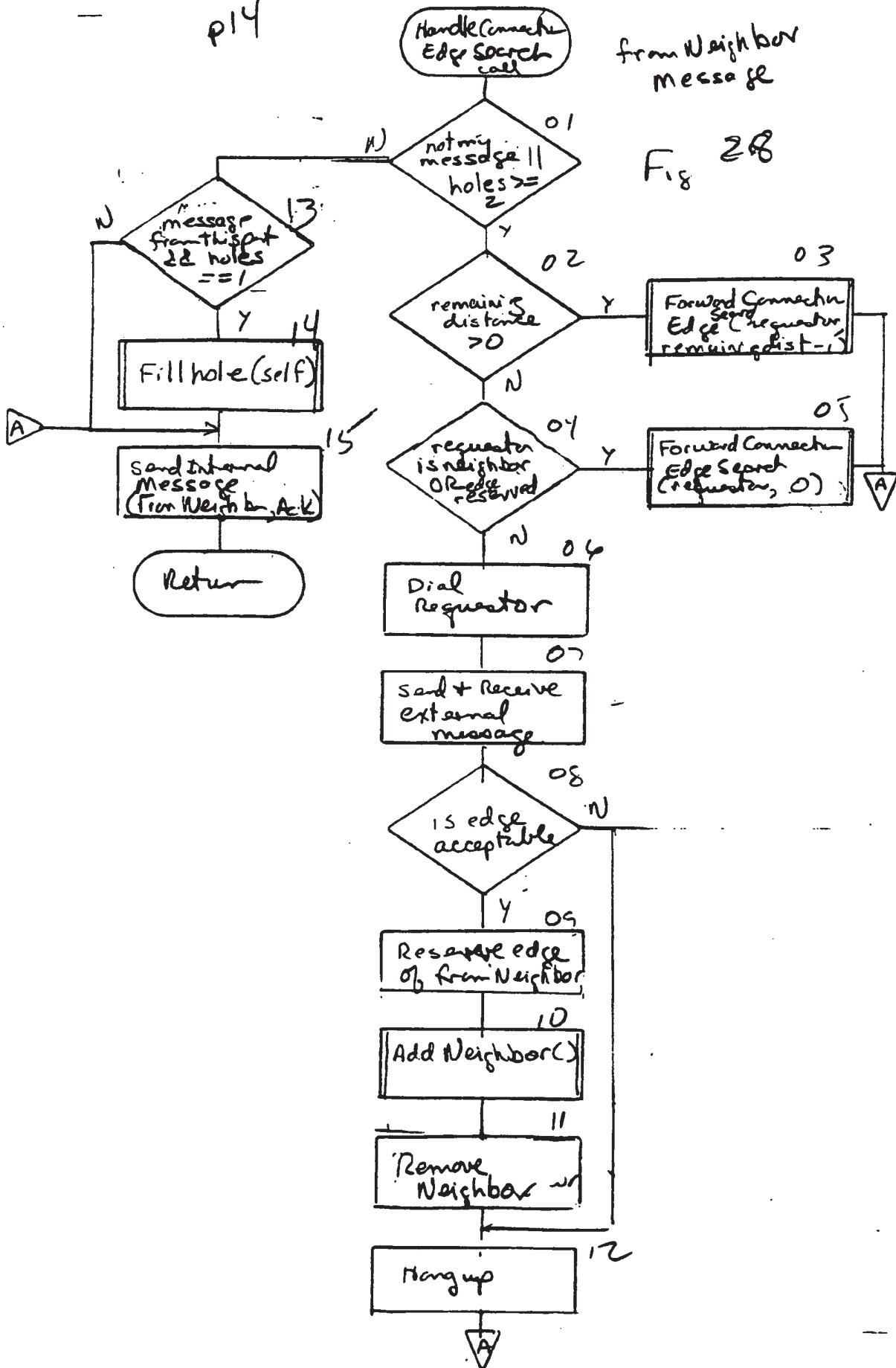
21

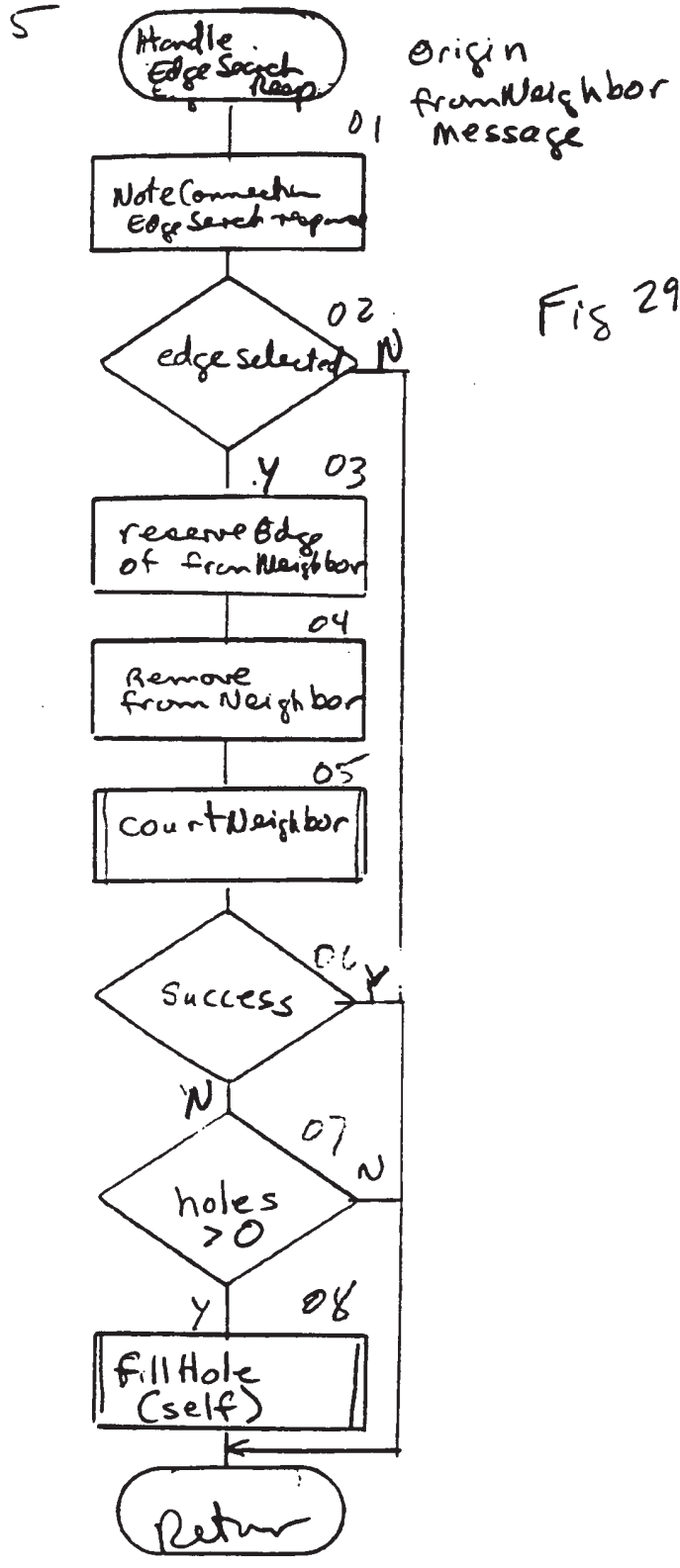


p14

from Neighbor  
message

Fig 28





24

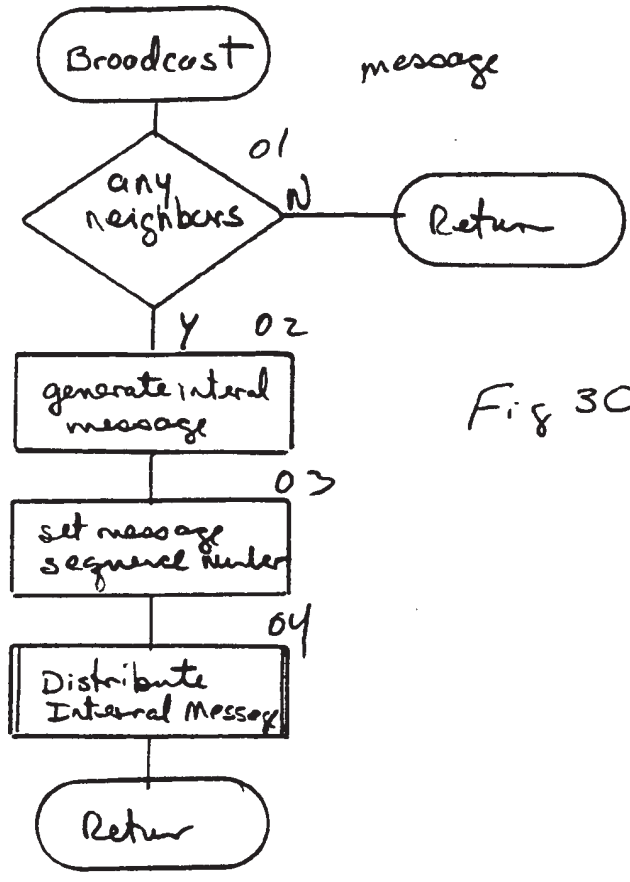
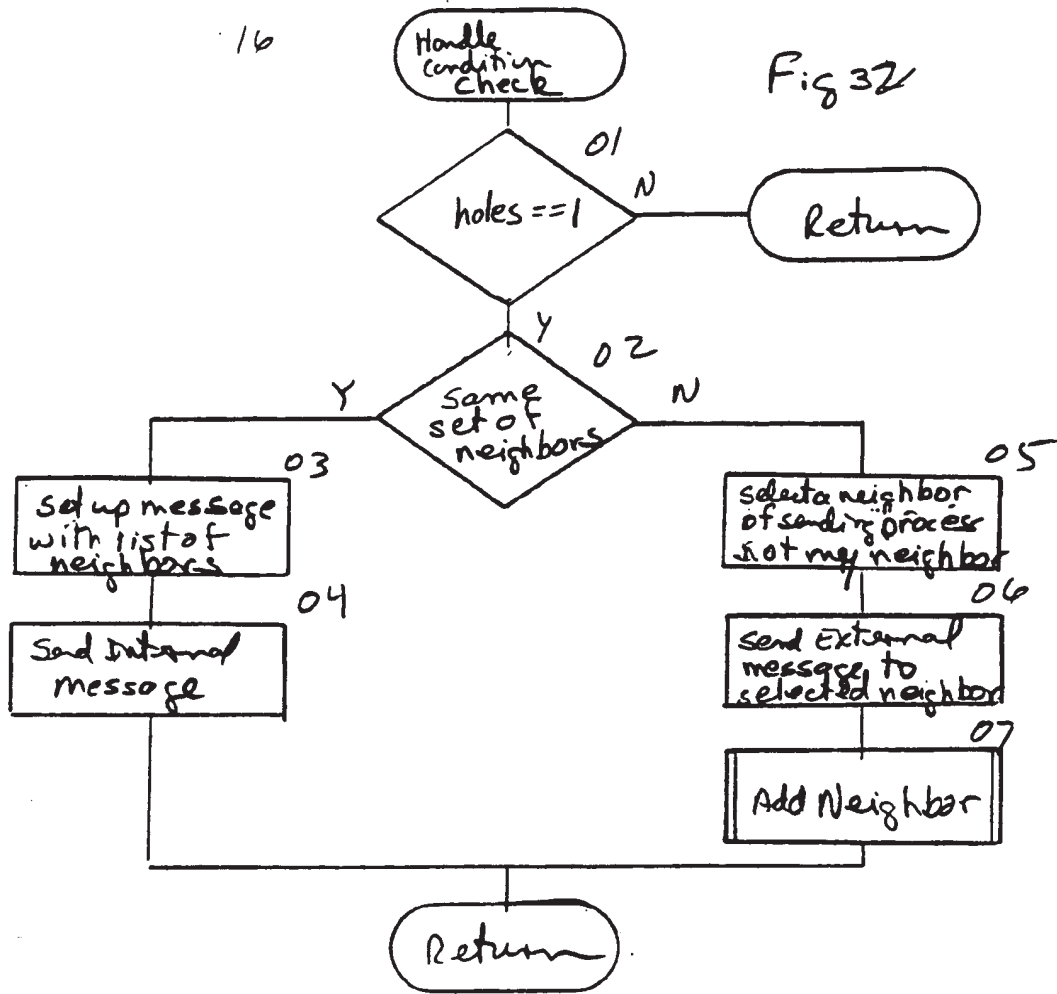


Fig 30

16

Fig 32



3

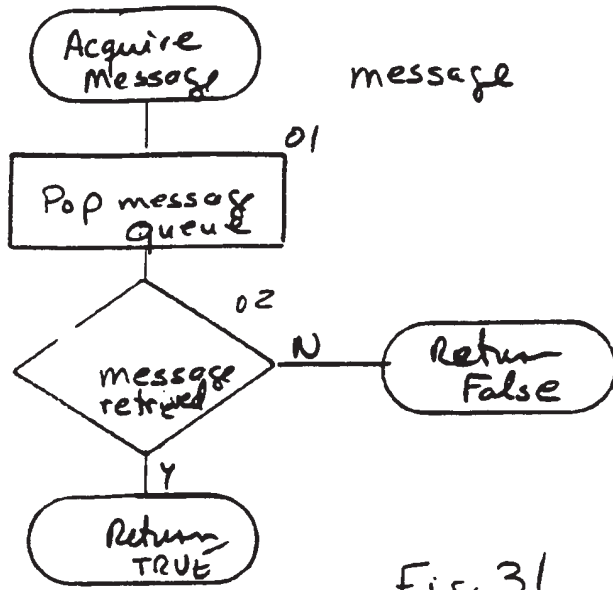
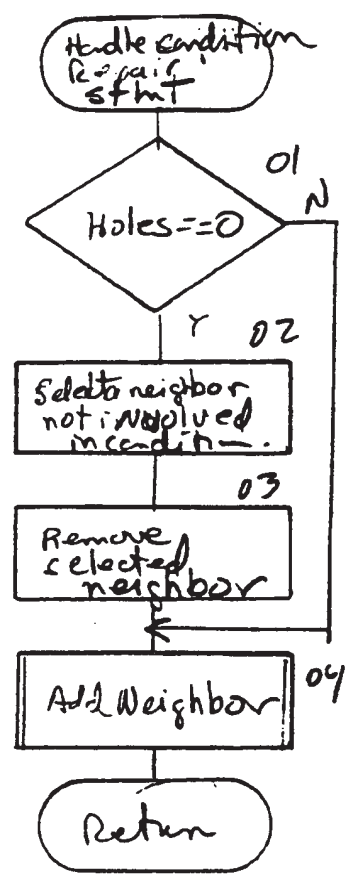
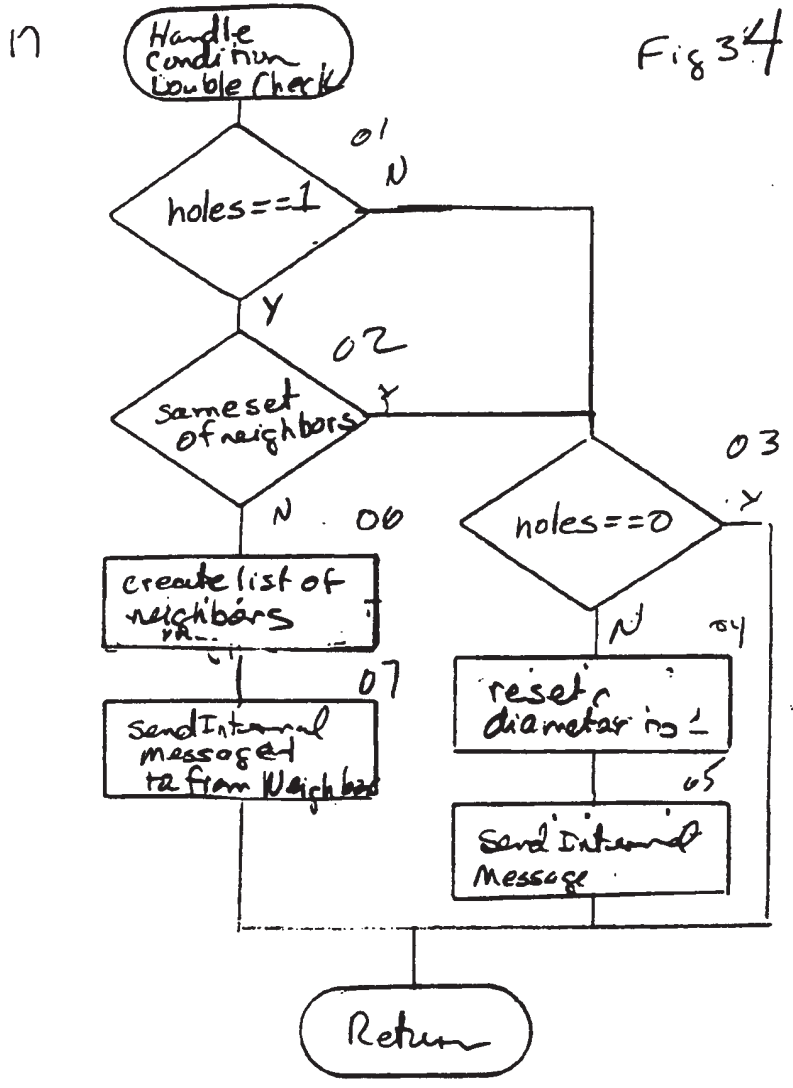


Fig 31

23

Fig 33





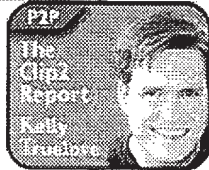


macromedia  
**COLDFUSION 5**

Get there faster. Write less code.

Best Available Copy

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.

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.

### 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](#)



## 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 malformed 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](http://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, outside, 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](#).

**oreillynet.com** Copyright © 2000 O'Reilly & Associates, Inc.

# 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.*

Best Available Copy

## 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 sen-

cially 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.

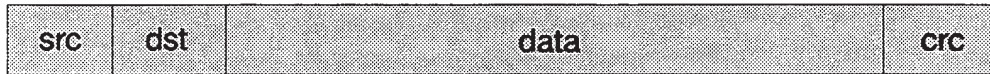


Figure 1.

sors, 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.

*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.*

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

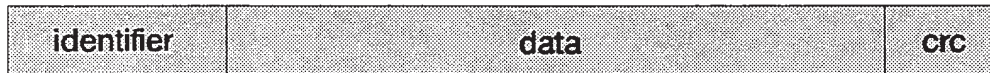


Figure 2.

*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 typi-

caly smarter, the extra diagnostic and configuration data can absorb considerable bandwidth.

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/con-

sumer 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 accommodates the future ☺

---

*Patricia A. Murphy is manager for Emerging Technology and Standards at Rockwell Automation Control Systems.*

*Murphy's responsibilities include:*

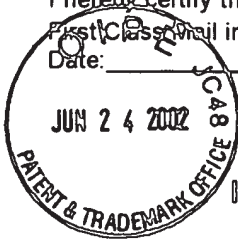
- *Integrating advanced technology into Rockwell Automation Control System's Communications Business.*
- *Coordinating advanced technology projects.*
- *Participating in industry consortia such as Fieldbus Foundation, ODVA and ControlNet International.*
- *Leading business standards activity and participating 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.*

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, Washington, D.C., 20231, on:

Date: 6/17/02

By: Jeanne Connelly  
Jeanne Connelly



PATENT

*[Handwritten signature]*

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

IN RE APPLICATION OF:

ART UNIT No. 2744

FRED B. HOLT AND VIRGIL E. BOURASSA

APPLICATION No.: 09/629,570

FILED: JULY 31, 2000

FOR: JOINING A BROADCAST CHANNEL

RECEIVED

JUN 27 2002

Technology Center 2600

Supplemental Information Disclosure Statement Within Three Months of Application Filing or Before First Action - 37 CFR 1.97(b)

Commissioner for Patents  
Washington, D.C. 20231

Sir:

1. Timing of Submission

This information disclosure is being filed within three months of the filing date of this application or date of entry into the national stage of an international application or before the mailing date of a first Office action on the merits, whichever occurs last [37 CFR 1.97(b)]. The references listed on the enclosed Form PTO/SB/08A (modified) may be material to the examination of this application; the Examiner is requested to make them of record in the application.

2. Cited Information

- Copies of the following references are enclosed:
  - All cited references
  - References marked by asterisks
  - The following:
- Copies of the following references can be found in parent application Ser. No.
  - All cited references
  - References marked by asterisks
  - The following:
- The following references are not in English. For each such reference, the undersigned has enclosed (i) a translation of the reference; (ii) a copy of a communication from a foreign patent office or International Searching Authority citing the reference, (iii) a copy of a reference which appears to be an English-language counterpart, or (iv) an English-language abstract for the reference prepared by a third party. Applicant has not verified that the





translation, English-language counterpart or third-party abstract is an accurate representation of the teachings of the non-English reference, though, and reserves the right to demonstrate otherwise.

- All cited references
- References marked by ampersands
- The following:

3. Effect of Information Disclosure Statement (37 CFR 1.97(h))

This Information Disclosure Statement is not to be construed as a representation that: (i) a search has been made; (ii) additional information material to the examination of this application does not exist; (iii) the information, protocols, results and the like reported by third parties are accurate or enabling; or (iv) the cited information is, or is considered to be, material to patentability. In addition, applicant does not admit that any enclosed item of information constitutes prior art to the subject invention and specifically reserves the right to demonstrate that any such reference is not prior art.

4. Fee Payment

No fees are believed due. However, should the Commissioner determine that fees are due in order for this Information Disclosure Statement to be considered, the Commissioner is hereby authorized to charge such fees to Deposit Account No. 50-0665.

5. Patent Term Adjustment (37 CFR 1.704(d))

- The undersigned states that each item of information submitted herewith was cited in a communication from a foreign patent office in a counterpart application and that this communication was not received by any individual designated in 37 C.F.R. § 1.56(c) more than thirty days prior to the filing of this statement. 37 C.F.R. § 1.704(d).

Respectfully submitted,  
Perkins Coie LLP

  
Maurice J. Piro

Registration No. 33,273

Date: 6/13/02

Correspondence Address:

Customer No. 25096  
Perkins Coie LLP  
P.O. Box 1247  
Seattle, Washington 98111-1247  
Phone: (206) 583-8888

# RELIABLE BROADCAST IN MOBILE WIRELESS NETWORKS<sup>1</sup>

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,

<sup>1</sup>This research was supported in part by NSF under Grant N . CRR-9110177, by the Texas Advanced Technology Program under Grant No. 9741-036, and by a grant from Electrospace Systems, Inc., a Chrysler Company.

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  $2 \leq \mu \leq 5$  [12] to denote the power law relationship between distance and received power. Based on both analysis and measured results,  $\mu=4$  for the microcell propagation environment beyond a characteristic distance  $R_0$ . The power law model takes the form [13]:

$$PL(R) = PL(R_0) + 10\mu \log(R/R_0) + X_0 \quad (1)$$

where  $PL(R_0)$  gives the power loss at the characteristic distance  $R_0$  and  $X_0$  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-capture 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

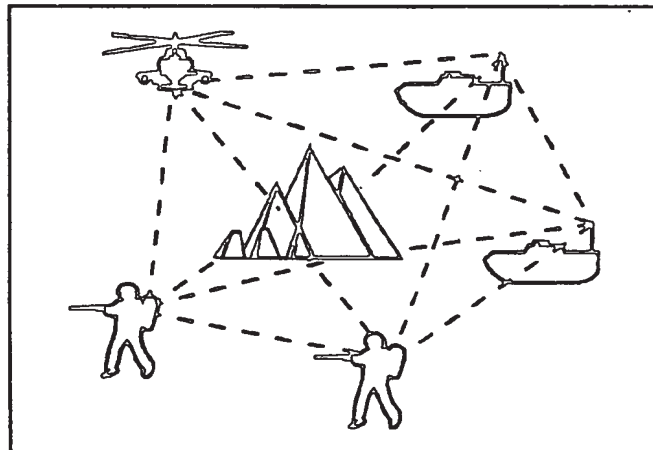


Figure 1. Model of a wireless computer network that experiences link failures due to range limitations and terrain impairments.

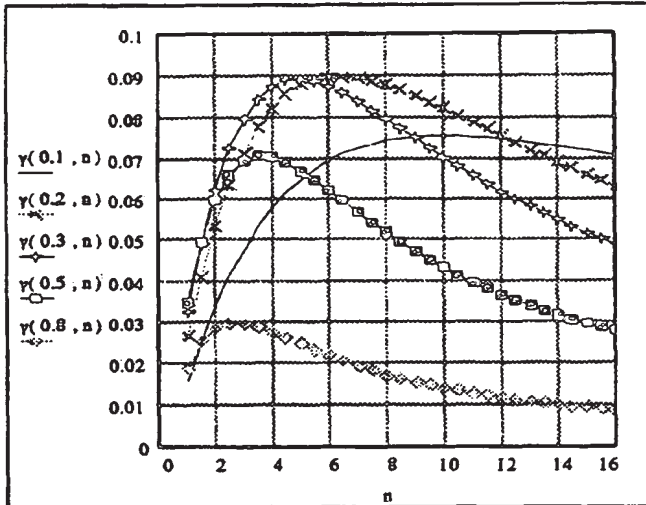


Figure 2. Normalized throughput for perfect capture with the slotted ALOHA protocol versus the average number of nearest neighbors for different probabilities that a node is busy.

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

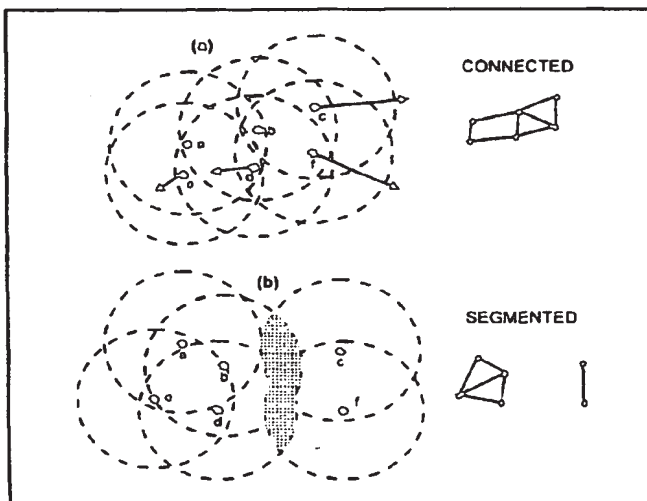


Figure 3. Examples of (a) a fully connected mobile wireless network and (b) a decomposed network due to mobility of hosts *c* and *f*. The shaded region indicates the common area within the range of hosts *c* and *f* and the rest of the network.

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.

**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  $h_i$  has received messages 1 and 4, but not

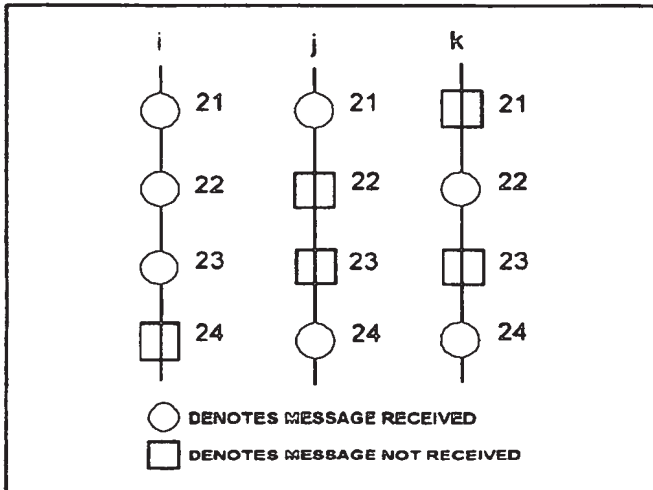


Figure 4. A snapshot of the status of the  $HISTORY_i$  showing messages stored in  $h_i$ ,  $h_j$  and  $h_k$ .

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.

**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  $h_j$  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  $h_k$ .  $h_j$  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  $k^{th}$  component of  $HISTORY_j$  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., Yokota, 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 personal 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.

PATENT COOPERATION TREATY

DOCKETED

0.5004. 0001 WC  
WTP/JC

#6

From the INTERNATIONAL SEARCHING AUTHORITY

JUN 11 2002

PCT

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

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 030048001W0	<b>FOR FURTHER ACTION</b> 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  
Facsimile 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 90bis.1 and 90bis.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 the provisions 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 publication. 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 application 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.

PATENT COOPERATION TREATY

PCT

INTERNATIONAL SEARCH REPORT

(PCT Article 18 and Rules 43 and 44)

Applicant's or agent's file reference <b>030048001W0</b>	<b>FOR FURTHER ACTION</b> see Notification of Transmittal of International Search Report (Form PCT/ISA/220) as well as, where applicable, item 5 below.	
International application No. <b>PCT/US 01/ 24240</b>	International filing date (day/month/year) <b>31/07/2001</b>	(Earliest) Priority Date (day/month/year) <b>31/07/2000</b>
Applicant <b>THE BOEING COMPANY</b>		

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 readable 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.

as suggested by the applicant.

because the applicant failed to suggest a figure.

because this figure better characterizes the invention.

1  
 None of the figures.