TW 7535866

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office

June 29, 2015

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

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

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

EXHIBIT
Ex. 1002
Volume 4

M. TARVER
Certifying Officer
PART (4) OF (6) PART(S)

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

M-Regular

In the embodiment described above, each fully connected computer has four internal connections. The broadcast technique can be used with other numbers of internal connections. For example, each computer could have 6, 8, or any even number of internal connections. As the number of internal connections increase, the diameter of the broadcast channel tends to decrease, and thus propagation time for a message tends to decrease. The time that it takes to connect a seeking computer to the broadcast channel may, however, increase as the number of internal connections increases. When the number of internal connectors is even, then the broadcast channel can be maintained as m-regular and m-connected (in the steady state). If the number of internal connections is odd, then when the broadcast channel has an odd number of computers connected, one of the computers will have less than that odd number of internal connections. In such a situation, the broadcast network is neither m-regular nor m-connected. When the next computer connects to the broadcast channel, it can again become m-regular and m-connected. Thus, with an odd number of internal connections, the broadcast channel toggles between being and not being m-regular and m-connected.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 961 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 962 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 963 of 1657

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.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 964 of 1657

EXTERNAL MESSAGES

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

INTERNAL MESSAGES

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 965 of 1657

	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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 966 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 967 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 968 of 1657

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

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 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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 969 of 1657

returns. In block 1203, the routine receives the external message from the external port. In decision block 1204, if the type of the message indicates that a seeking process is calling (*i.e.*, seeking_connection_call), then the routine continues at block 1205, else the routine returns. In block 1205, the routine sends an external message (*i.e.*, seeking_connection_resp) to the other seeking process indicating that this process is also is seeking a connection. In decision block 1206, if the sending of the external message is successful, then the routine continues at block 1207, else the routine returns. In block 1207, the routine adds the other seeking process to a list of fellow seeking processes and then returns. This list may be used if this process can find no process that is fully connected to the broadcast channel. In which case, this process may check to see if any fellow seeking process were successful in connecting to the broadcast channel. For example, a fellow seeking process may become the first process fully connected to the broadcast channel.

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 970 of 1657

request call routine in block 1406, else the routine continues at block 1407. In decision block 1407, if the message type is edge proposal call (*i.e.*, edge_proposal_call), then the routine invokes the handle edge proposal call routine in block 1408, else the routine continues at block 1409. In decision block 1409, if the message type is port connect call (*i.e.*, port_connect_call), then the routine invokes the handle port connection call routine in block 1410, else the routine continues at block 1411. In decision block 1411, if the message type is a connected statement (*i.e.*, connected_stmt), the routine invokes the handle connected statement in block 1112, else the routine continues at block 1212. In decision block 1412, if the message type is a condition repair statement (*i.e.*, condition_repair_stmt), then the routine invokes the handle condition repair routine in block 1413, else the routine loops to block 1414 to process the next message. After each handling routine is invoked, the routine loops to block 1414. In block 1414, the routine hangs up on the external port and continues at block 1401 to receive the next message.

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 971 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 972 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 973 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 974 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 975 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 976 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 977 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 978 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 979 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 980 of 1657

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.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 981 of 1657

CLAIMS

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

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

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

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

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

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

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 982 of 1657

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

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

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

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

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

14. A distributed game system comprising:

a plurality of broadcast channels, each broadcast channel for playing a game;

means for identifying a broadcast channel for a game of interest; and

means for connecting to the identified broadcast channel.

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 983 of 1657

3004-8009 6500

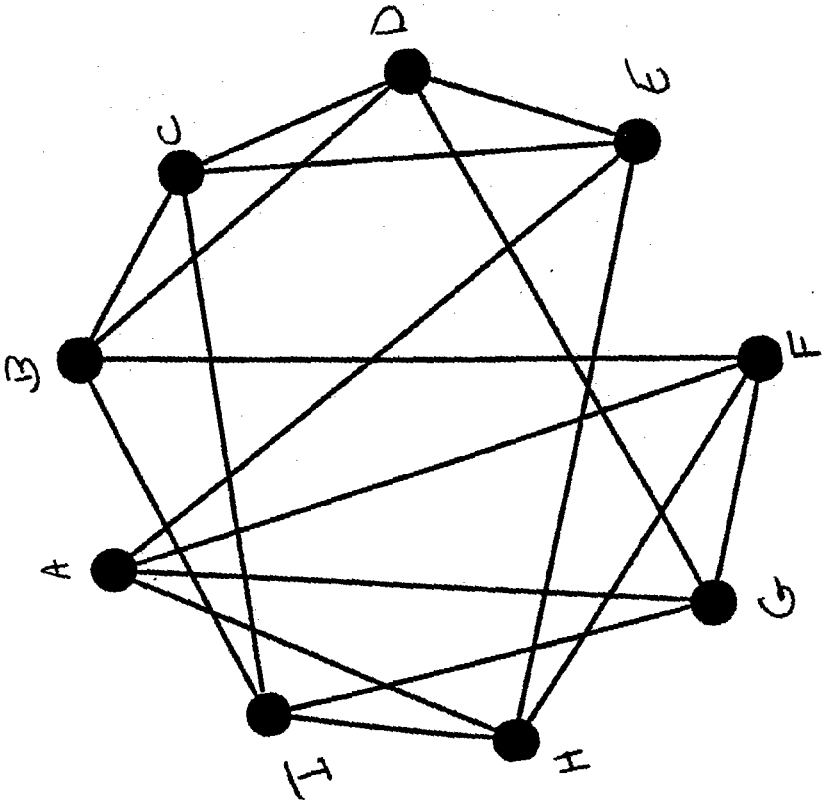


Fig 1

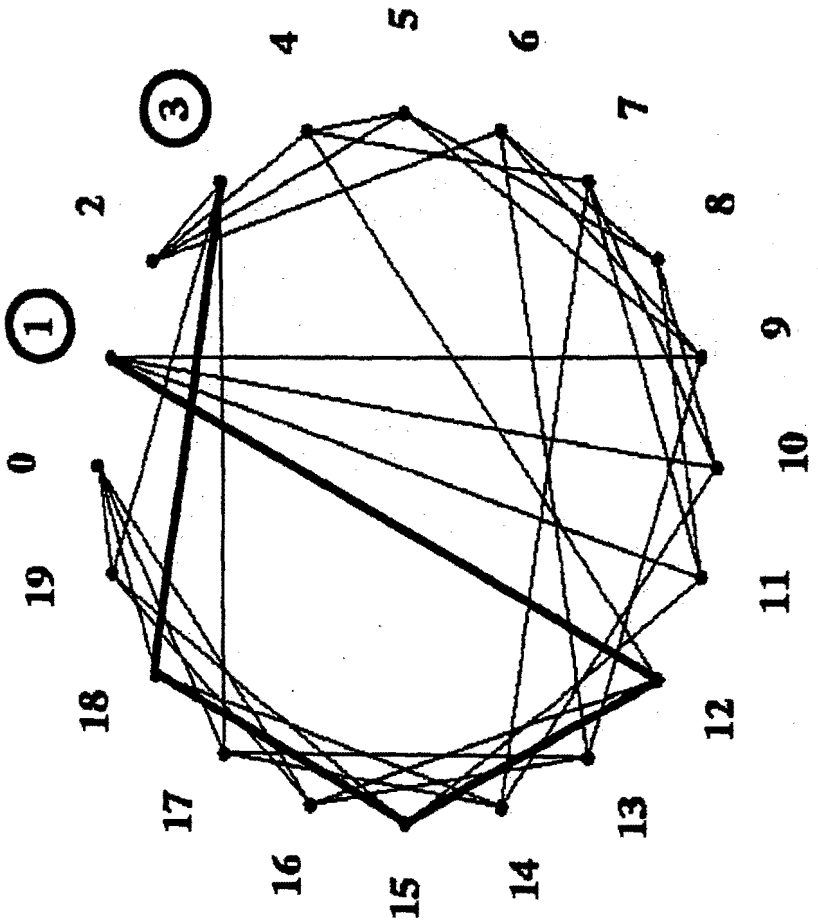


Fig 2

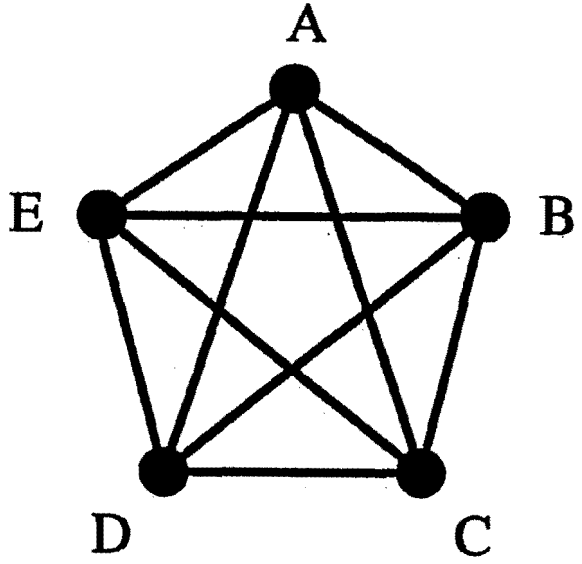


Fig 3A

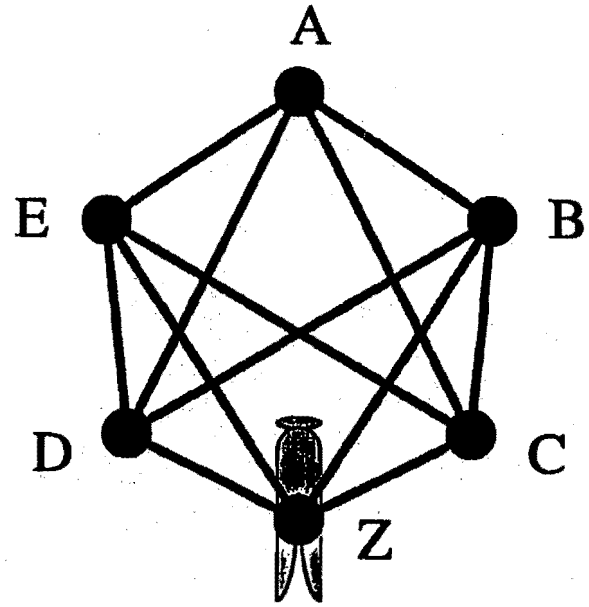


Fig 3B

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 986 of 1657

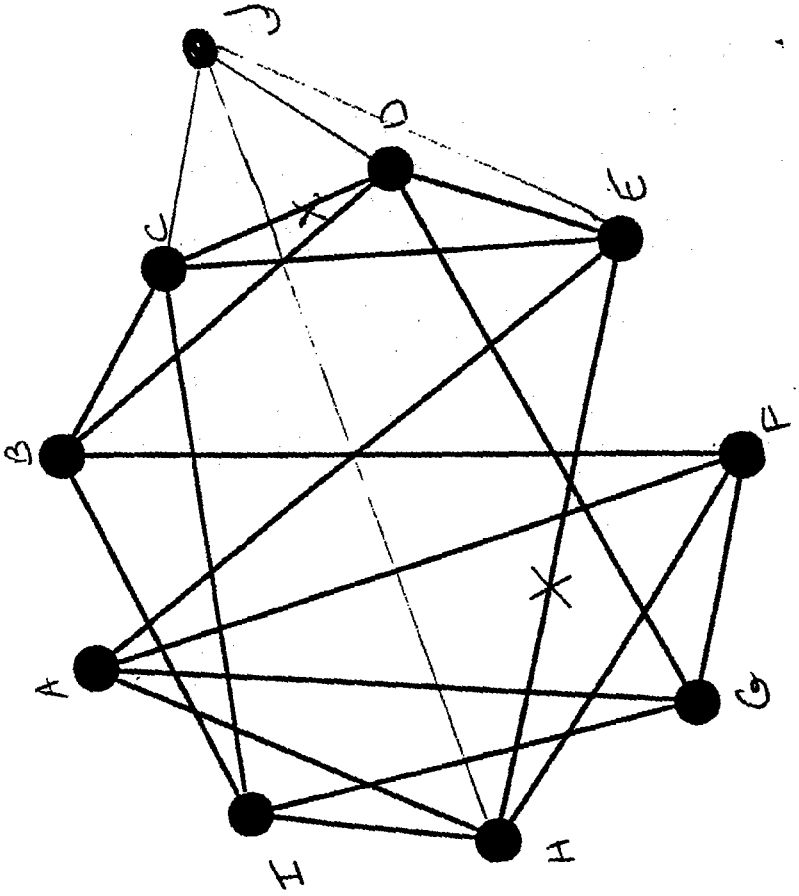


Fig 4A

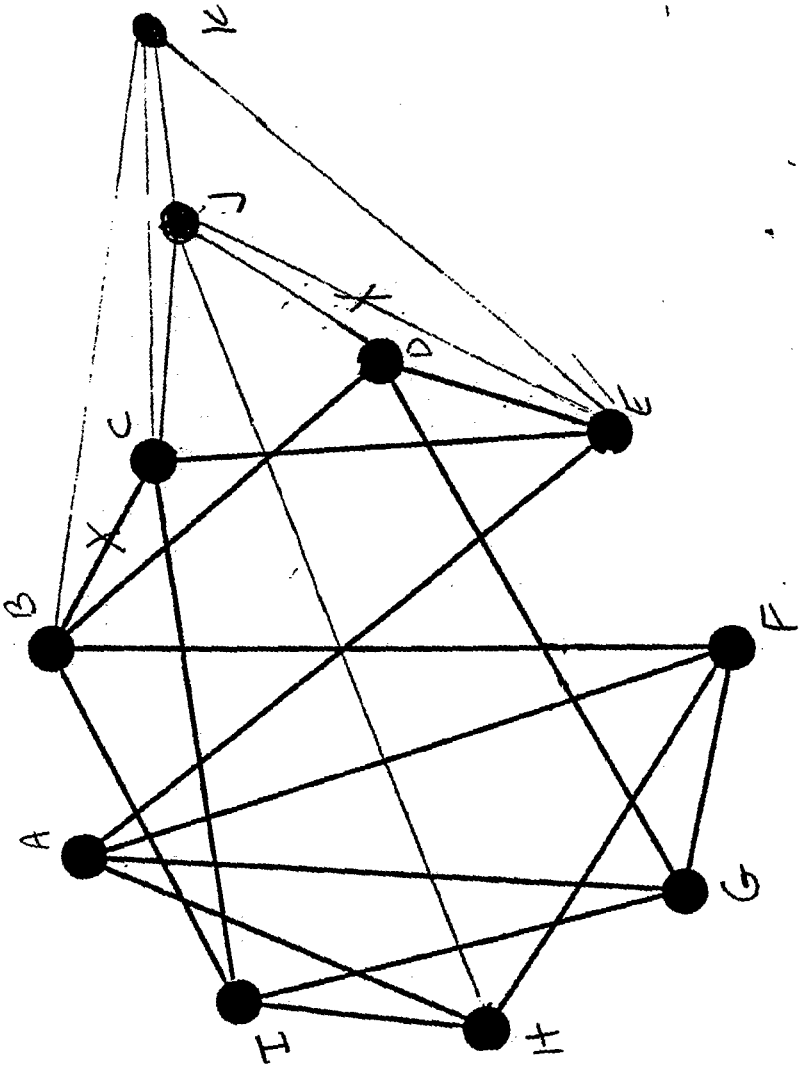


Fig 4B

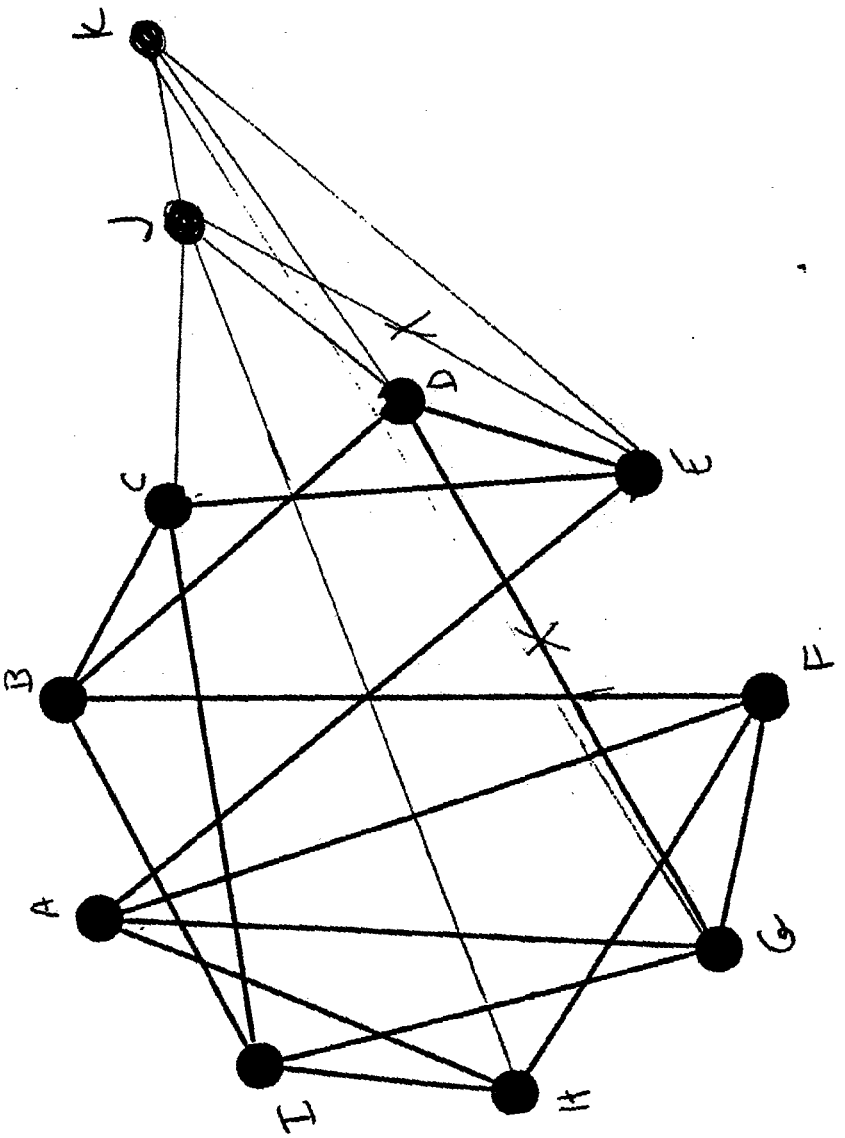


Fig 4V

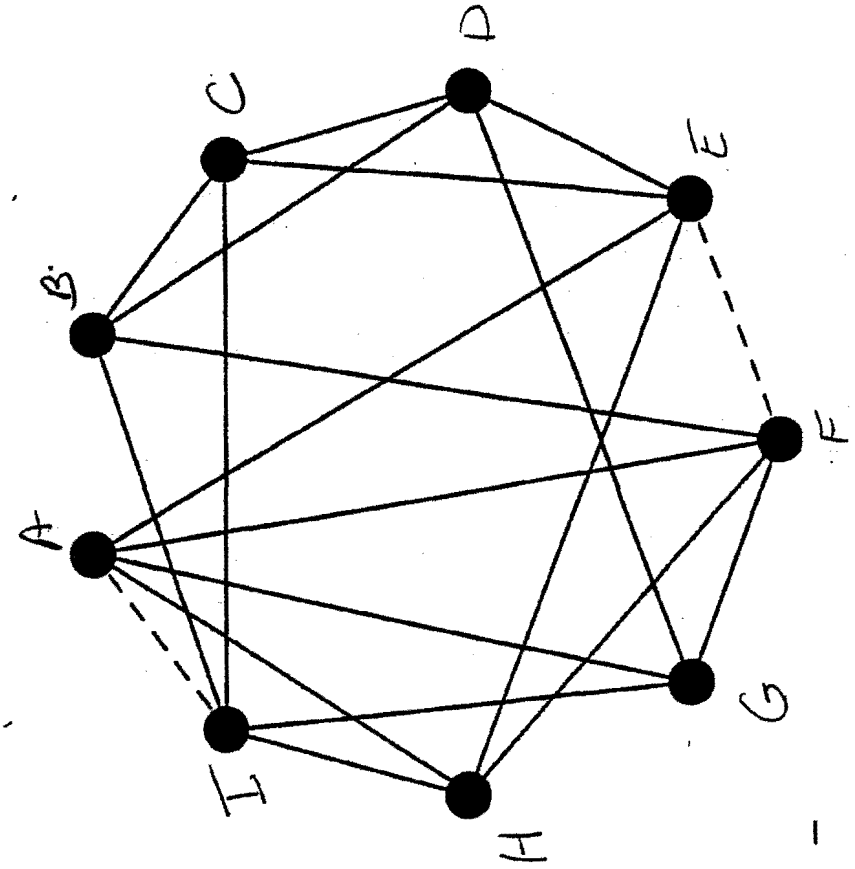


FIG 5A

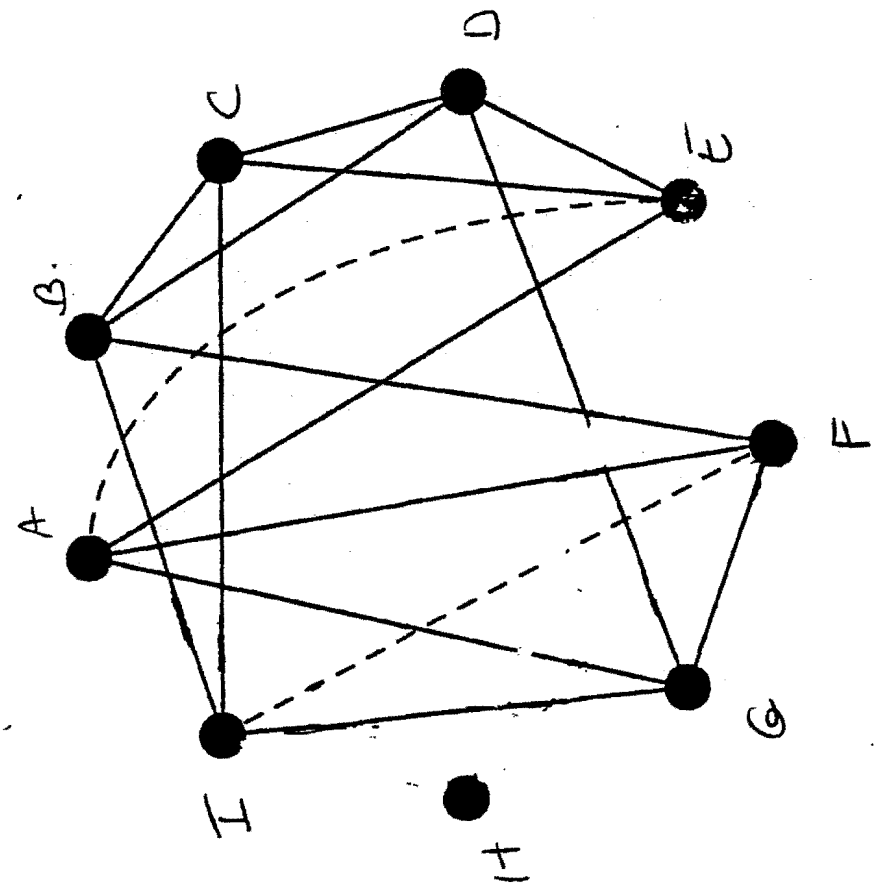


Fig 5B

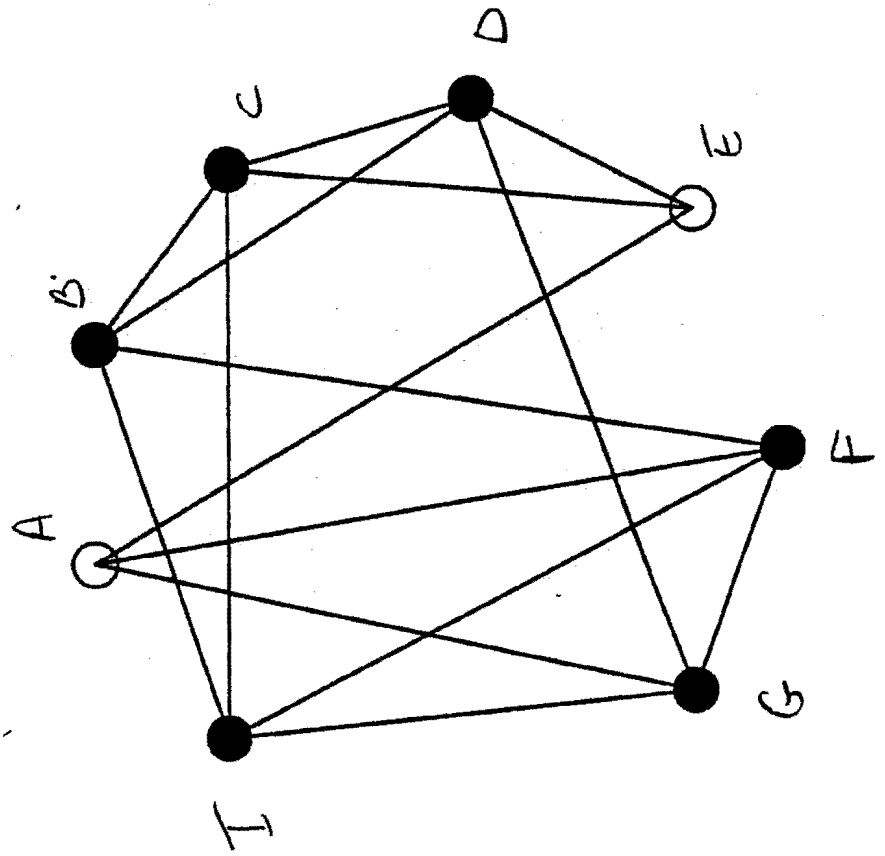


Fig 5C

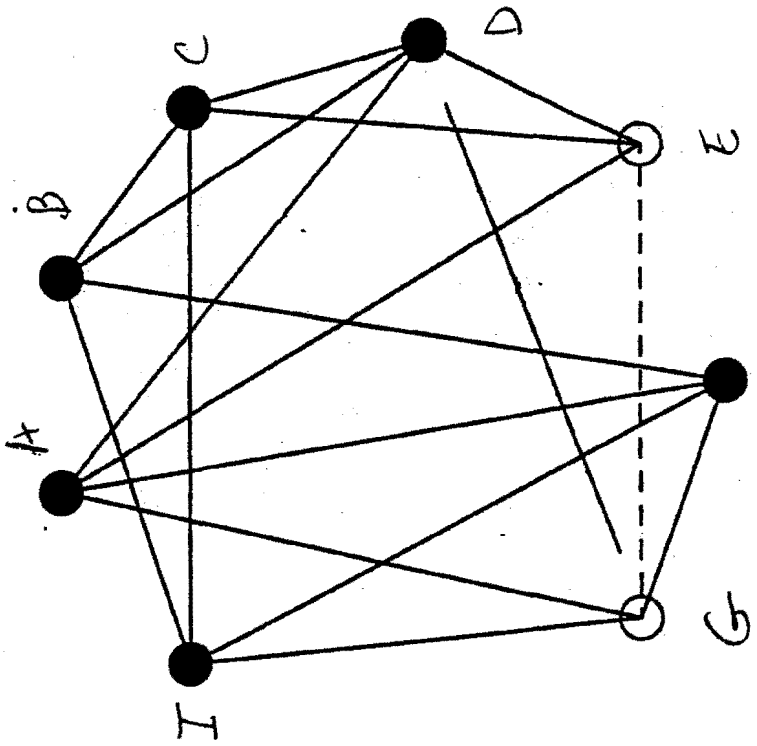


Fig 5D

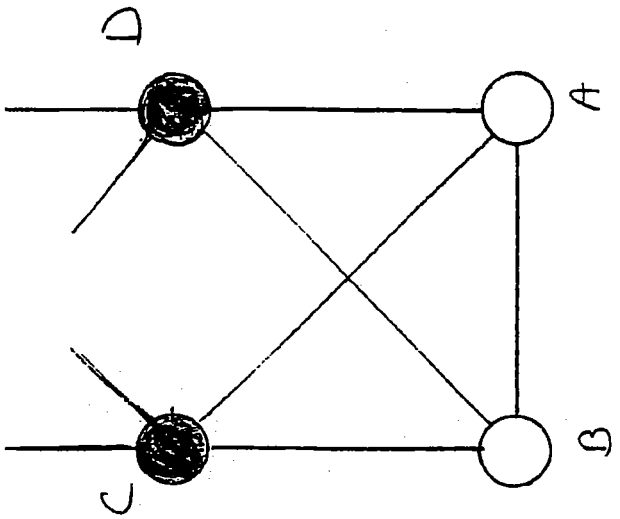


Fig 5F

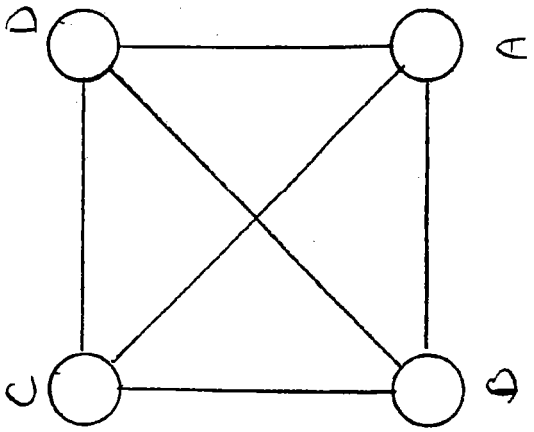


Fig 5E

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 995 of 1657

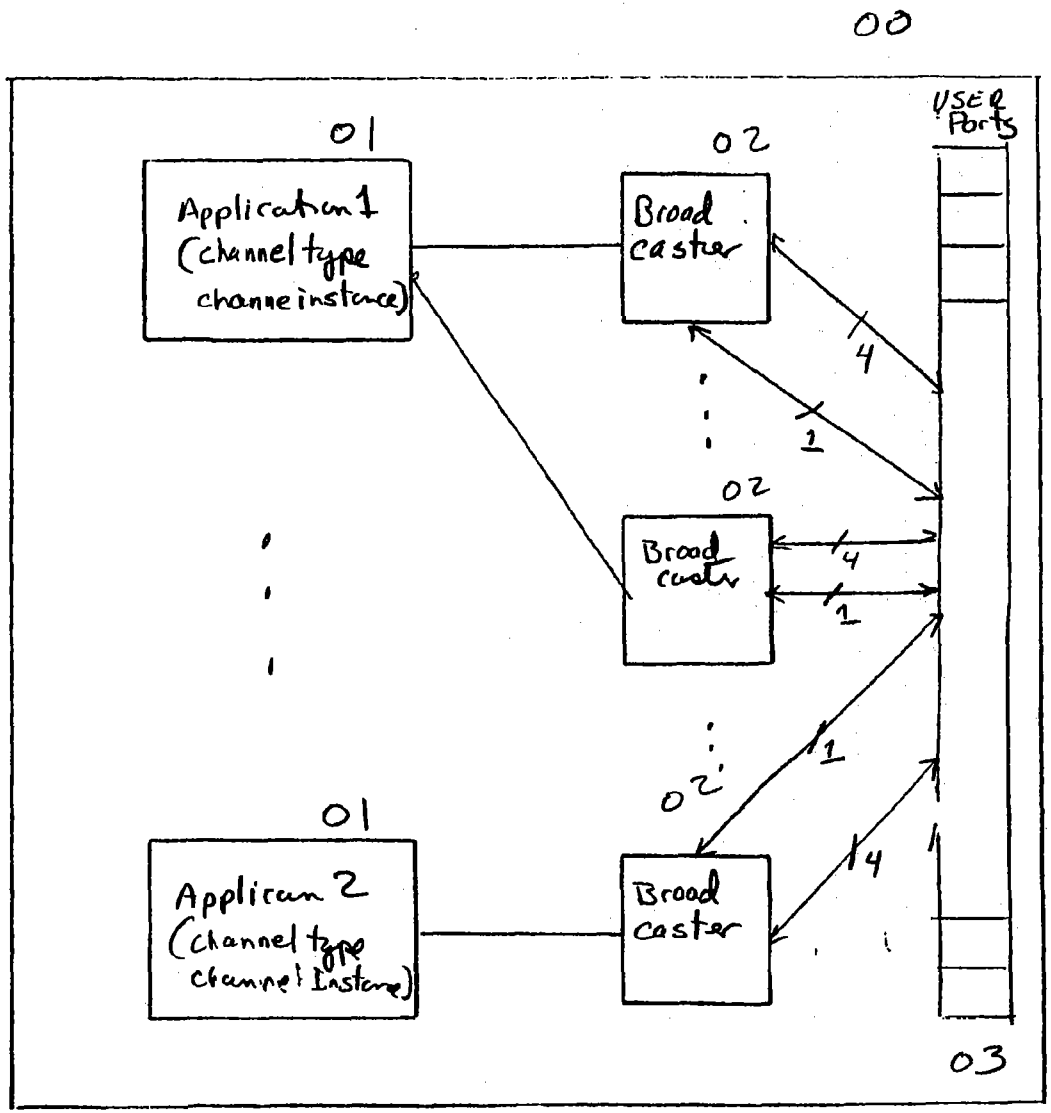


Fig 6

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 996 of 1657

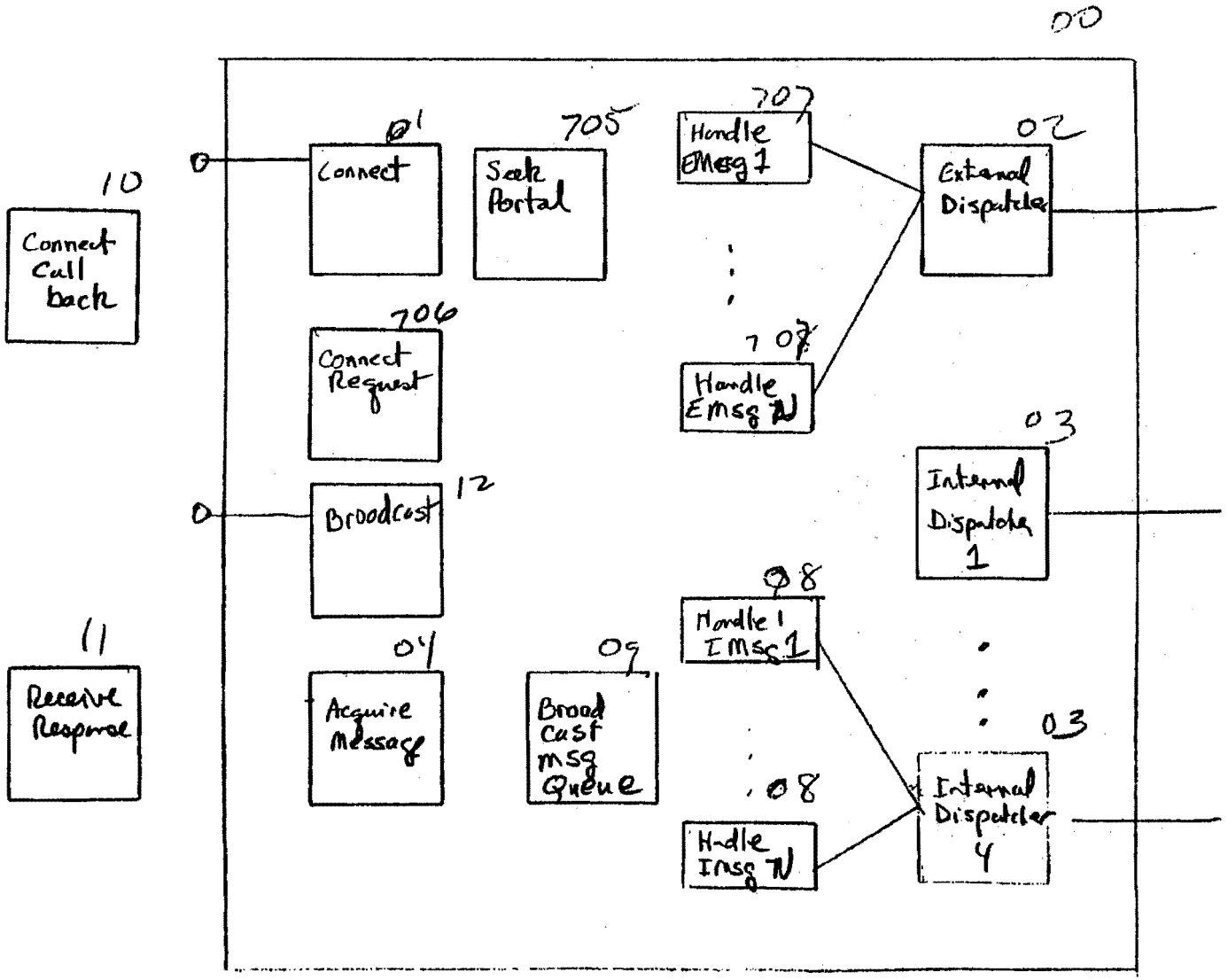


Figure 7

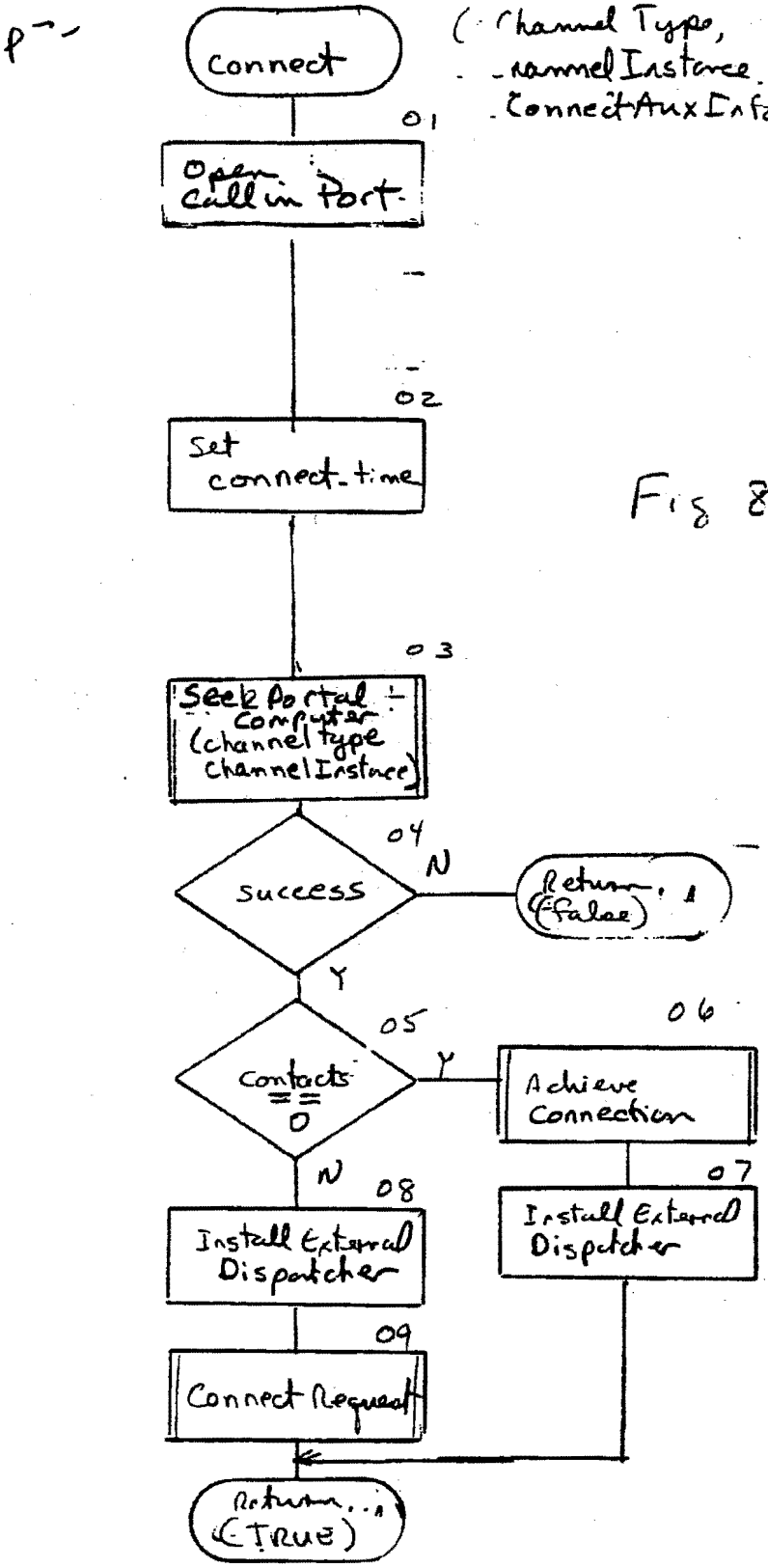

Connect
(Channel Type,
Channel Instance
ConnectAuxInfo)
01
Open Call in Port
02
Set connect_time
03
Seek Portal Computer (channel type channel Instance)
04
success
N
Return (false)
Y
05
Contacts == 0
Y
06
Achieve Connection
07
Install External Dispatcher
N
08
Install External Dispatcher
09
Connect Request
Return (TRUE)

Fig 8

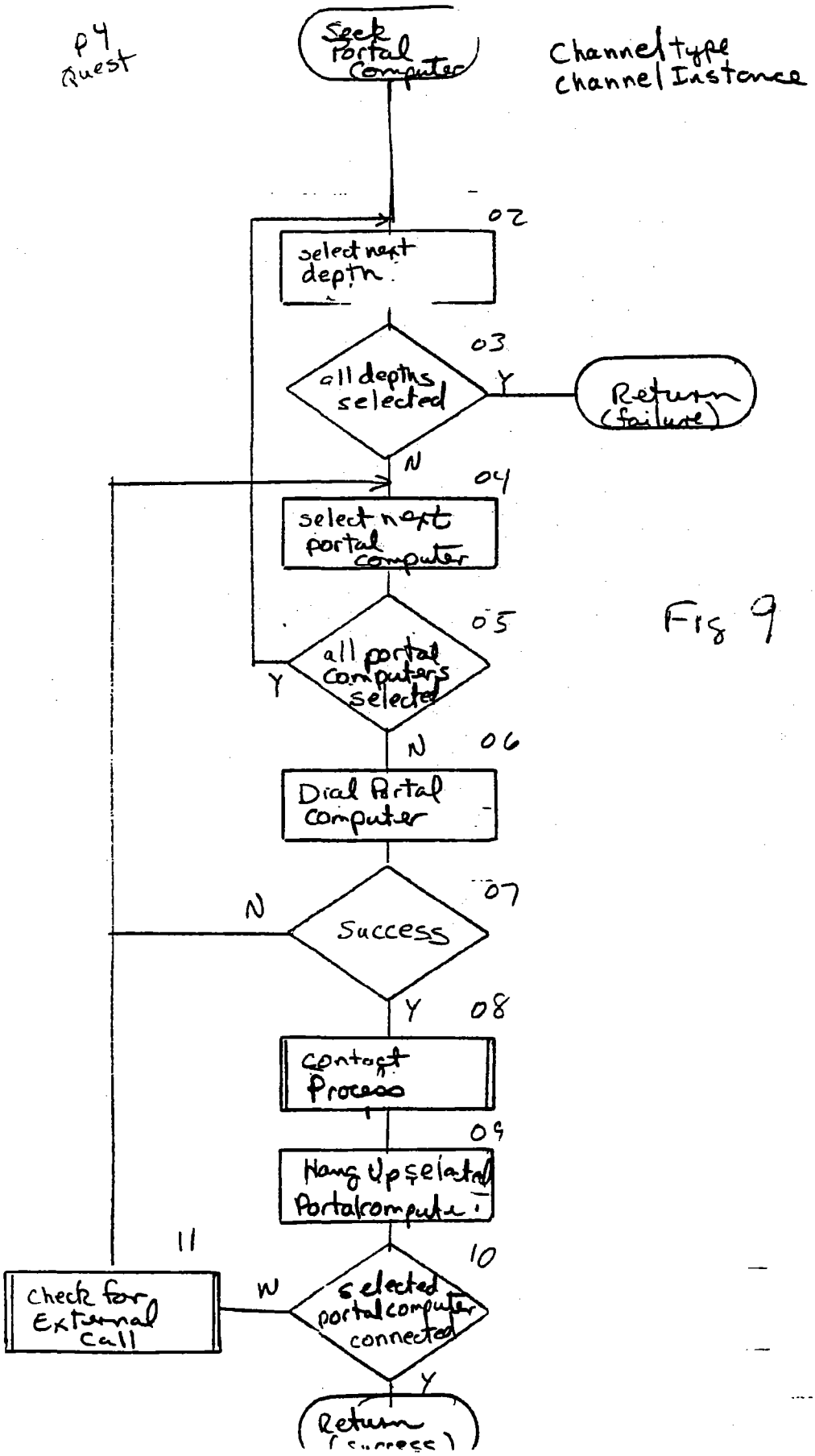

P4
Quest
Seek Portal Computer
Channel type
channel Instance
02
select next depth
03
all depths selected
Y
Return (failure)
N
04
select next portal computer
05
all portal computers selected
Y
N
06
Dial Portal computer
07
Success
N
Y
08
contact Process
09
Hang Up selected Portal computer
10
selected portal computer connected
N
11
Check for External call
Y
Return (success)

Fig 9

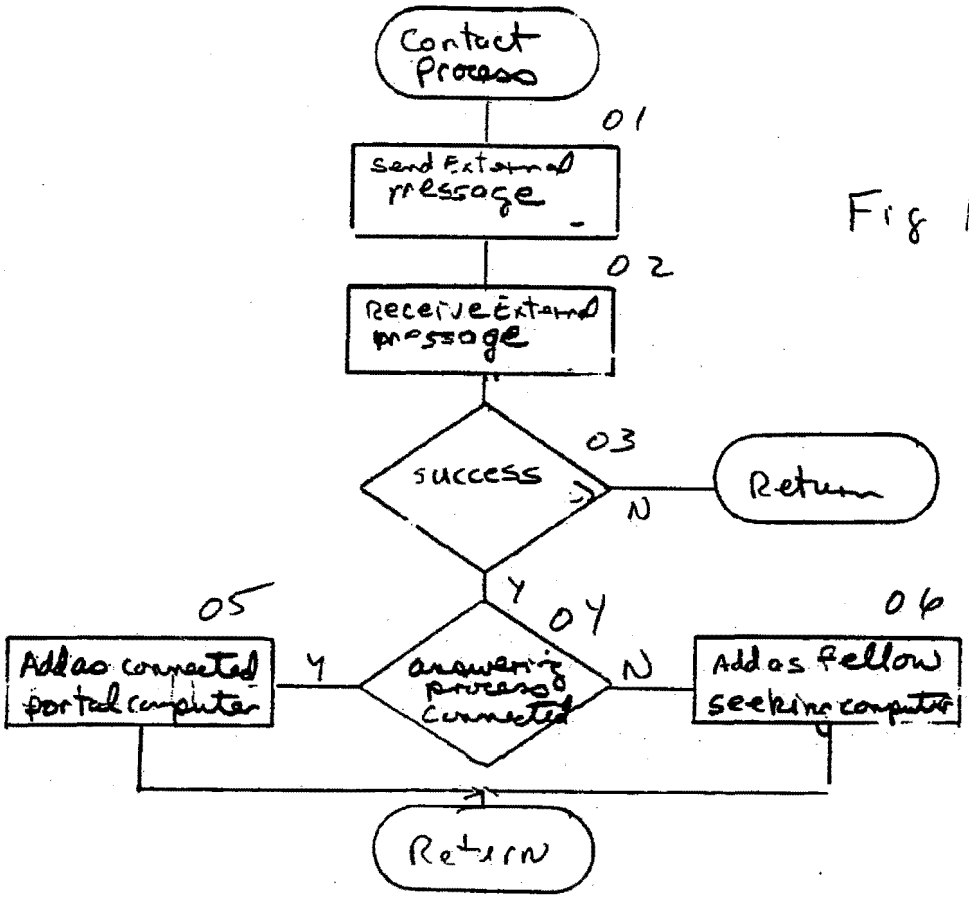
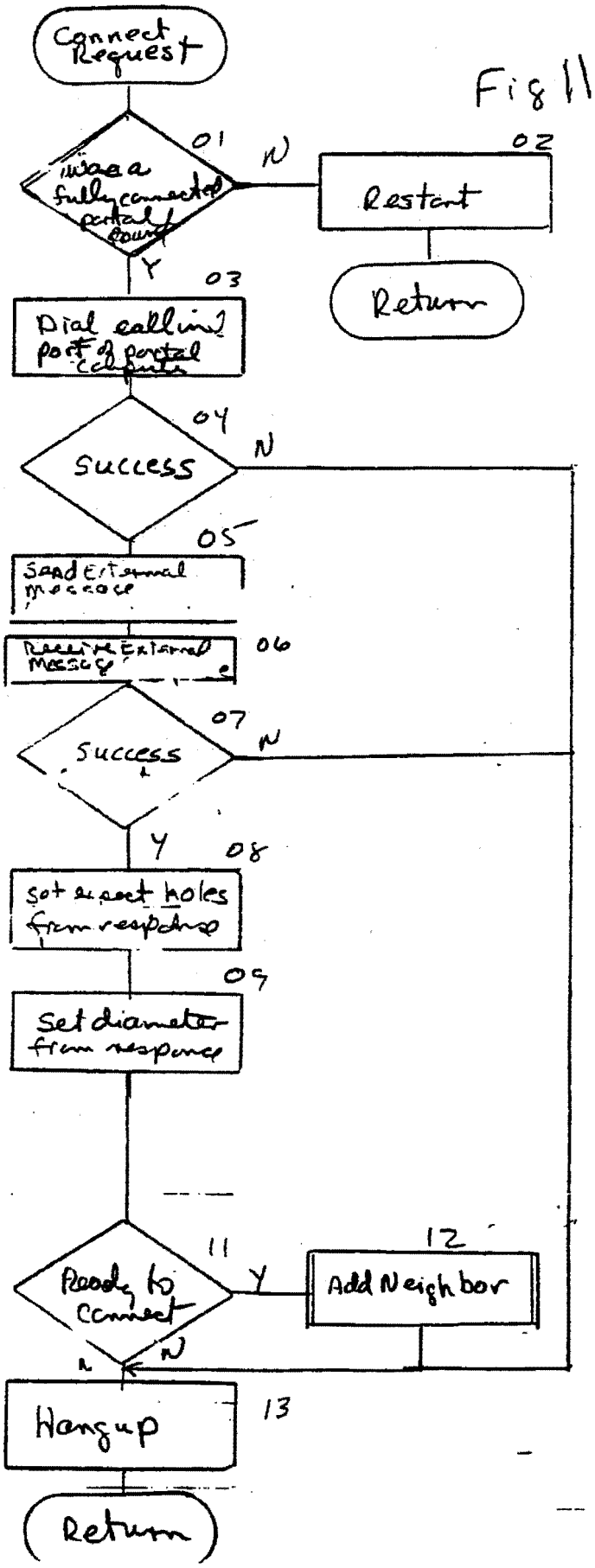

Contact Process
01
Send External message
02
Receive External message
success
03
N
Return
Y
04
05
Add as connected portal computer
Y
answering process connected
N
06
Add as fellow seeking computer
Return

Fig 10

p23

Fig 11


Connect Request
01
N
02
Restart
Return
Y
03
Dial calling port of portal
04
Success
N
05
Send External message
06
Receive External Message
07
Success
N
Y
08
Set aspect holes from response
09
Set diameter from response
11
Ready to Connect
Y
12
Add Neighbor
N
N
Hangup
13
Return

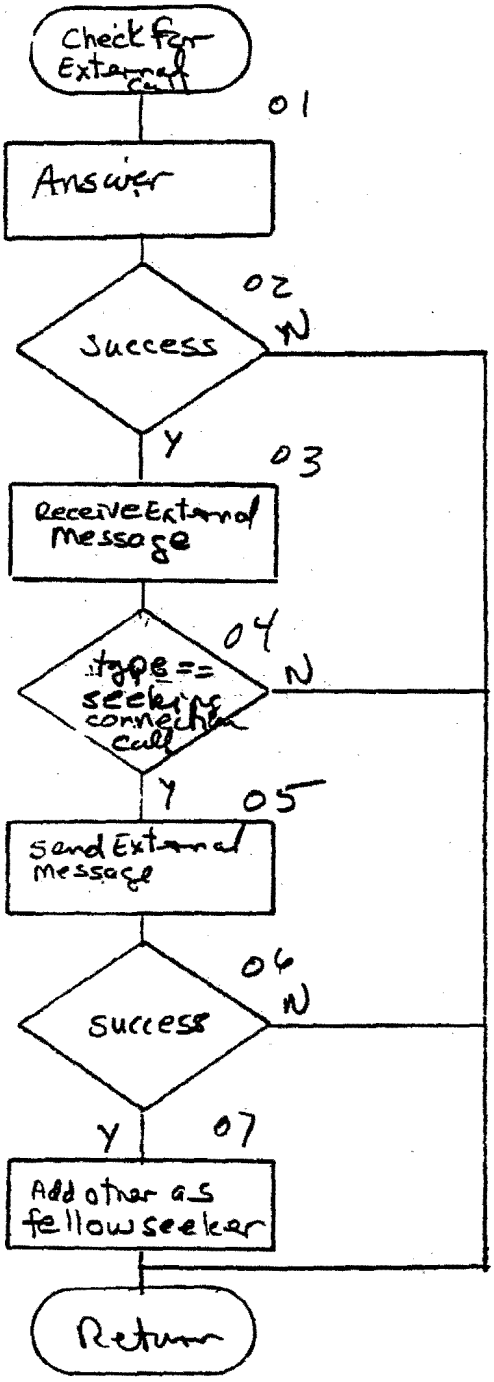

Check for External call
01
Answer
02
Success
N
Y
03
Receive External message
04
type == seeking connection call
N
Y
05
Send External message
06
Success
N
Y
07
Add other as fellow seeker
Return

Fig 12

Achieve Connection

22

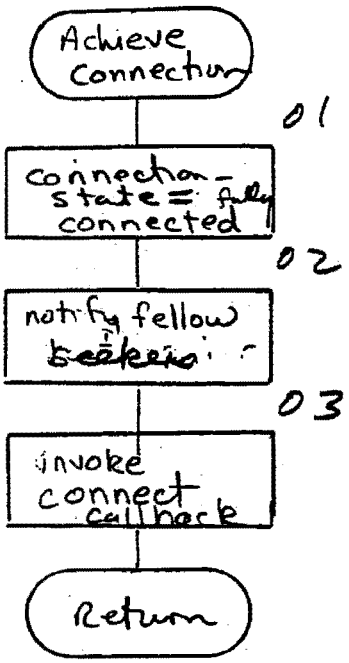
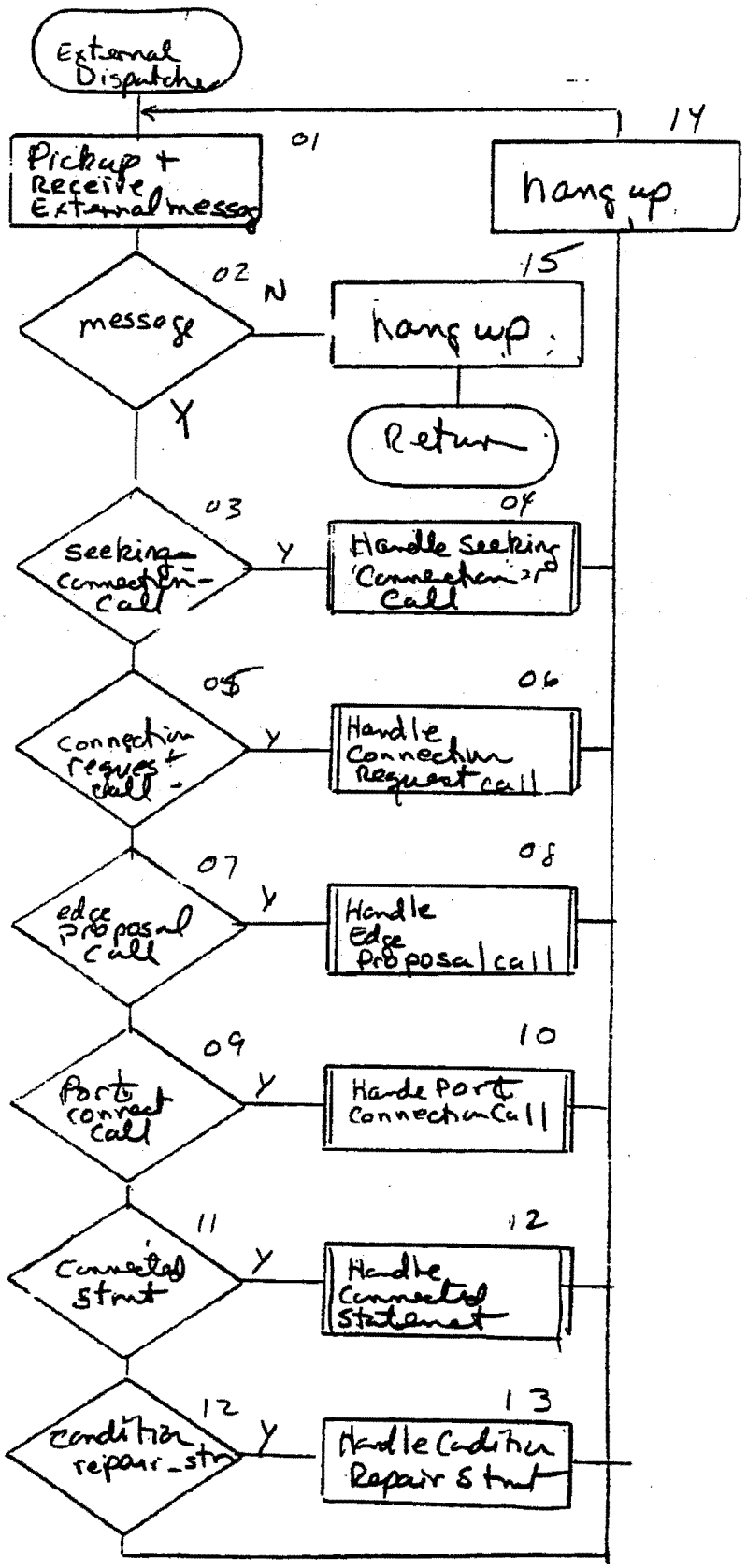


Fig 13

p7

Fig 14


External Dispatcher
Pickup + Receive External messag
01
14
hang up
02
N
message
15
hang up
Y
Return
03
04
seeking connection-call
Y
Handle Seeking Connection Call
05
06
Connection request call
Y
Handle Connection Request call
07
08
edge Proposal Call
Y
Handle Edge Proposal call
09
10
Port connect Call
Y
Handle Port Connection Call
11
12
Connected Stmt
Y
Handle Connected Statement
12
13
condition repair_stm
Y
Handle Condition Repair Stmt

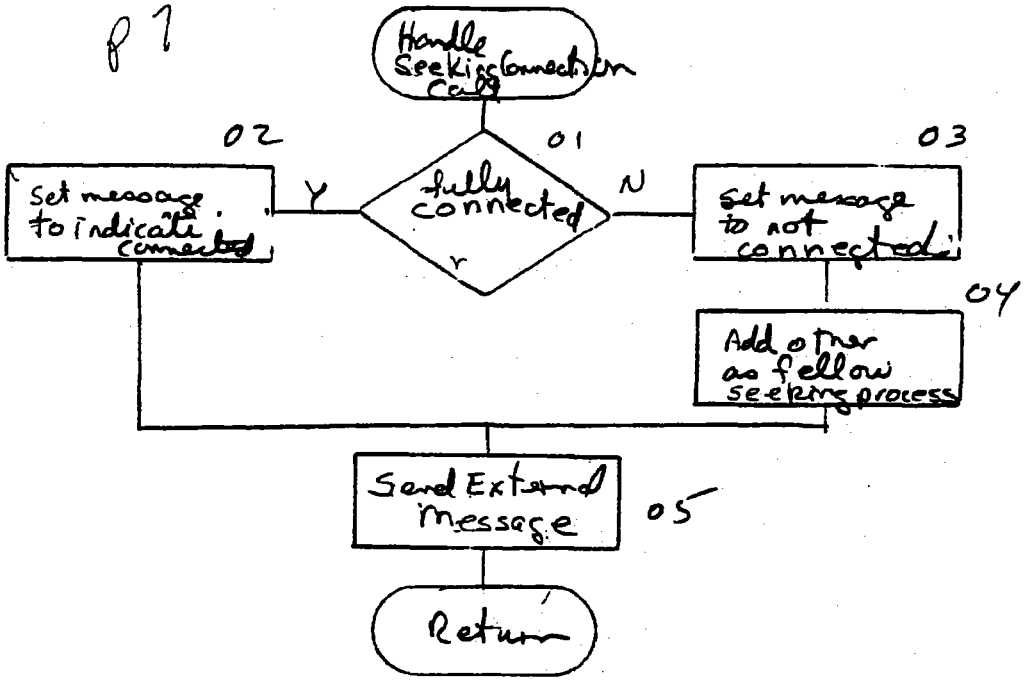
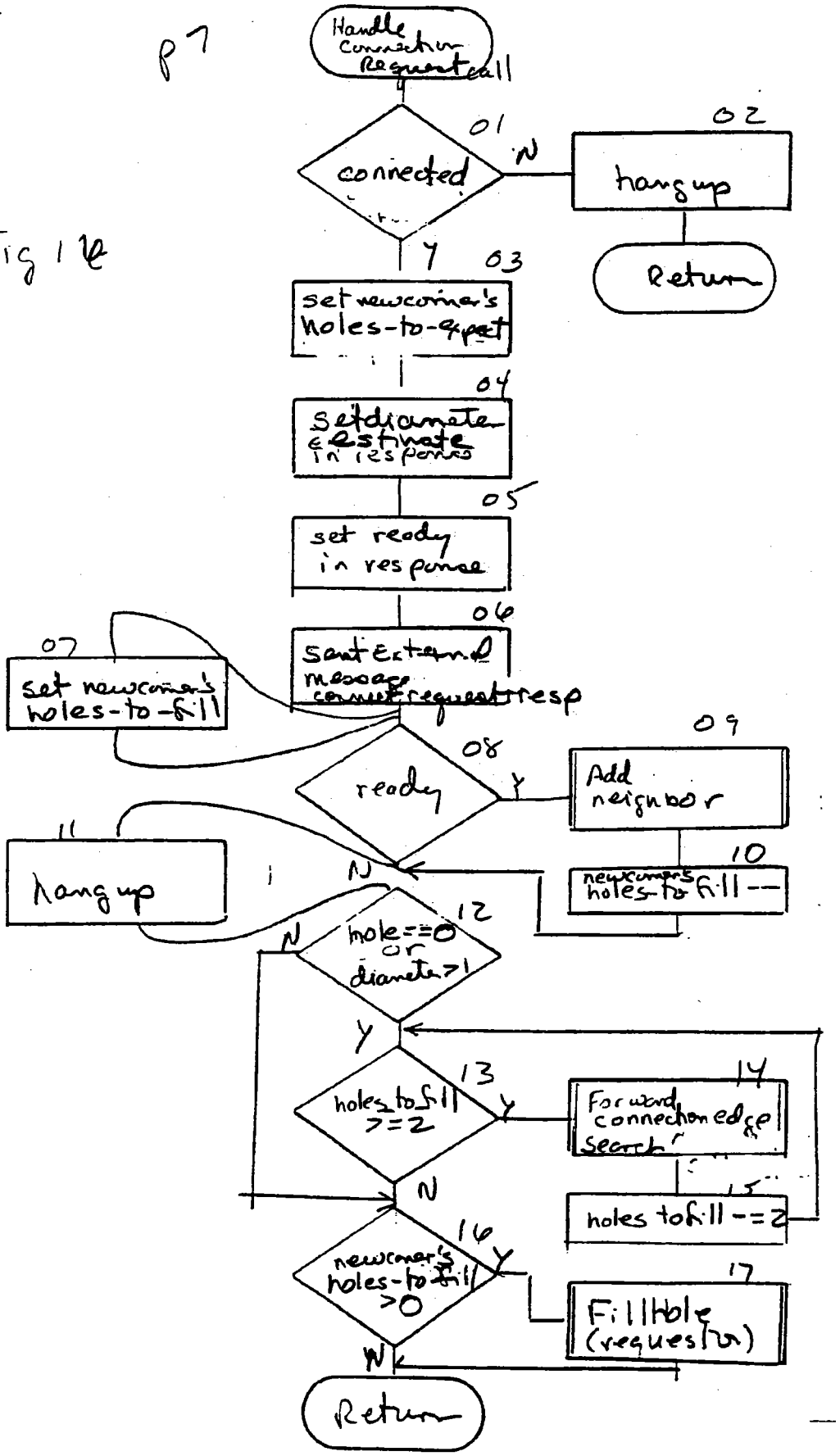

Handle Seeking Connection Call
01
02
03
04
05
fully connected
Y
N
Set message to indicate connected
Set message to not connected
Add other as fellow seeking process
Send Extended message
Return

Fig 15

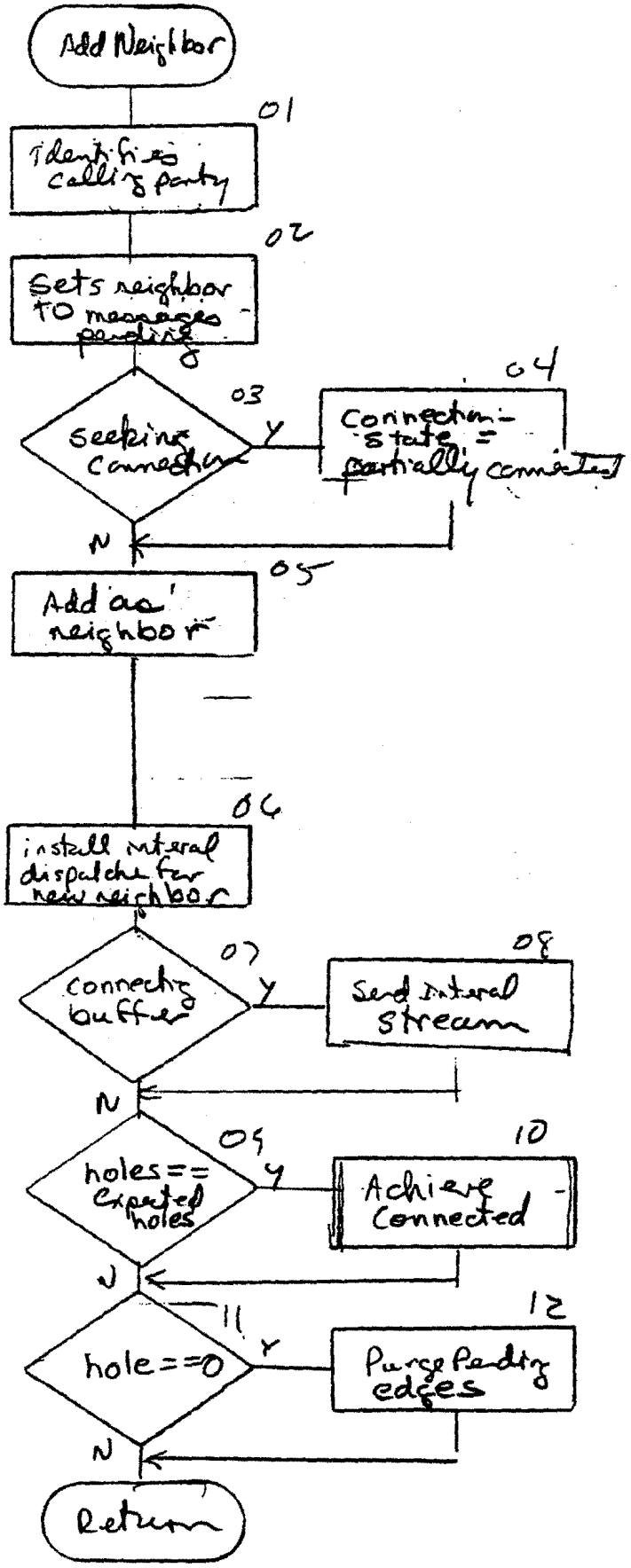
p7

Fig 16


Handle Connection Request call
01
connected
N
02
hangup
Return
Y
03
set newcomer's holes-to-eject
04
set diameter estimate in response
05
set ready in response
06
send extended message connect request resp
07
set newcomer's holes-to-fill
08
ready
Y
09
Add neighbor
10
newcomers holes-to fill --
11
hangup
N
12
hole==0 or diameter>1
N
Y
13
holes_to_fill >=2
Y
14
Forward connection edge search
15
holes to fill --=2
N
16
newcomer's holes-to-fill >0
Y
17
FillHole (requestor)
N
Return

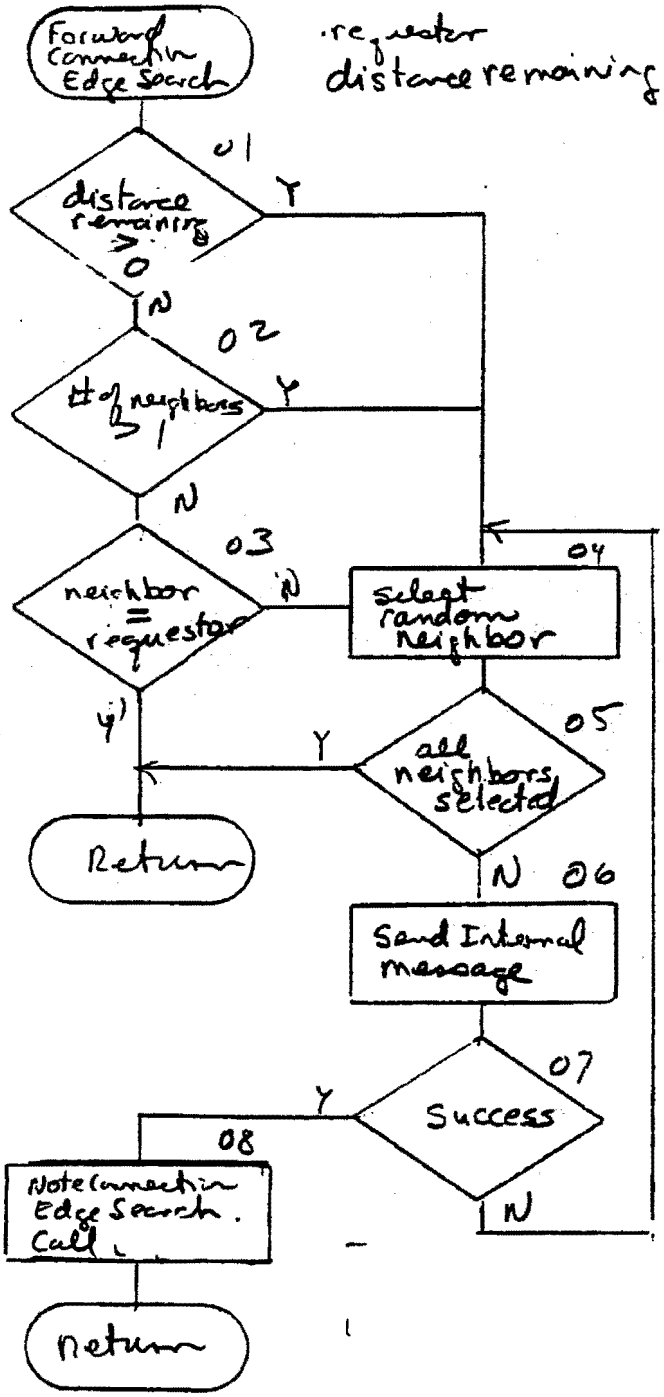
p9

Fig 17


Add Neighbor
01
Identifies calling party
02
Sets neighbor TO messages pending
03
Seeking Connection
Y
04
Connection-state = partially connected
N
05
Add 'aa' neighbor
06
install interal dispatcher for new neighbor
07
connecting buffer
Y
08
Send Interal Stream
N
09
holes == expected holes
Y
10
Achieve Connected
N
11
hole == 0
Y
12
Purge Pending edges
N
Return

15

Fig 18


Forward Connection Edge Search
requester
distance remaining
01
distance remaining > 0
Y
N
02
# of neighbors = 1
Y
N
03
neighbor = requester
N
04
select random neighbor
Y
05
all neighbors selected
Y
N
Return
06
Send Internal message
07
Success
Y
N
08
Note connection Edge Search. Call
return

p. 8

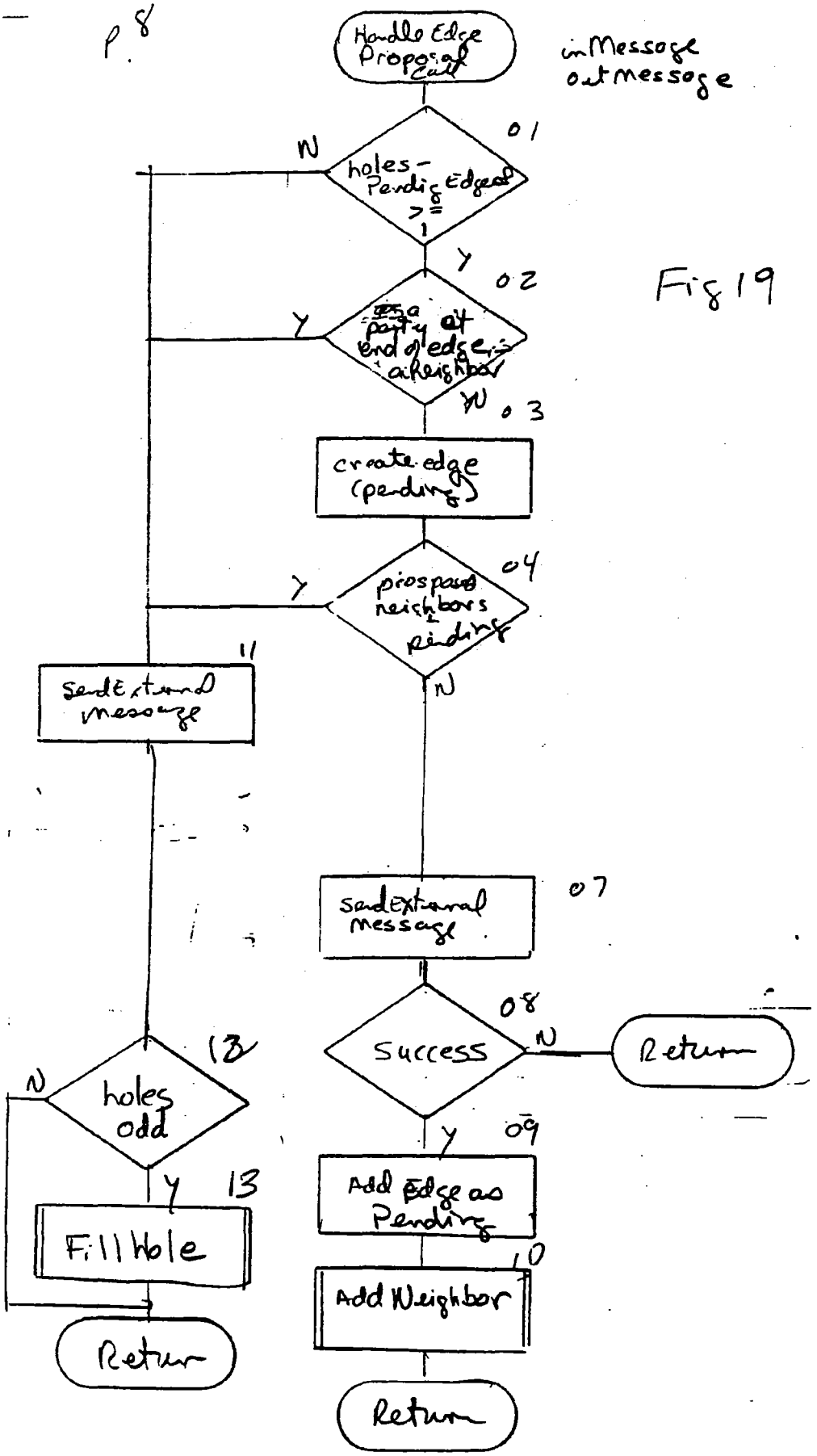

Handle Edge Proposal Call
in Message
out message
01
holes - Pending Edges >= 1
N
Y
02
a party at end of edge a Neighbor
Y
N
03
create edge (pending)
04
prospect neighbors pending
Y
N
11
Send External message
07
Send External message
08
Success
N
Return
Y
09
Add Edge as Pending
10
Add Neighbor
Return
12
holes odd
N
Y
13
Fill Hole
Return

Fig 19

Fig 20

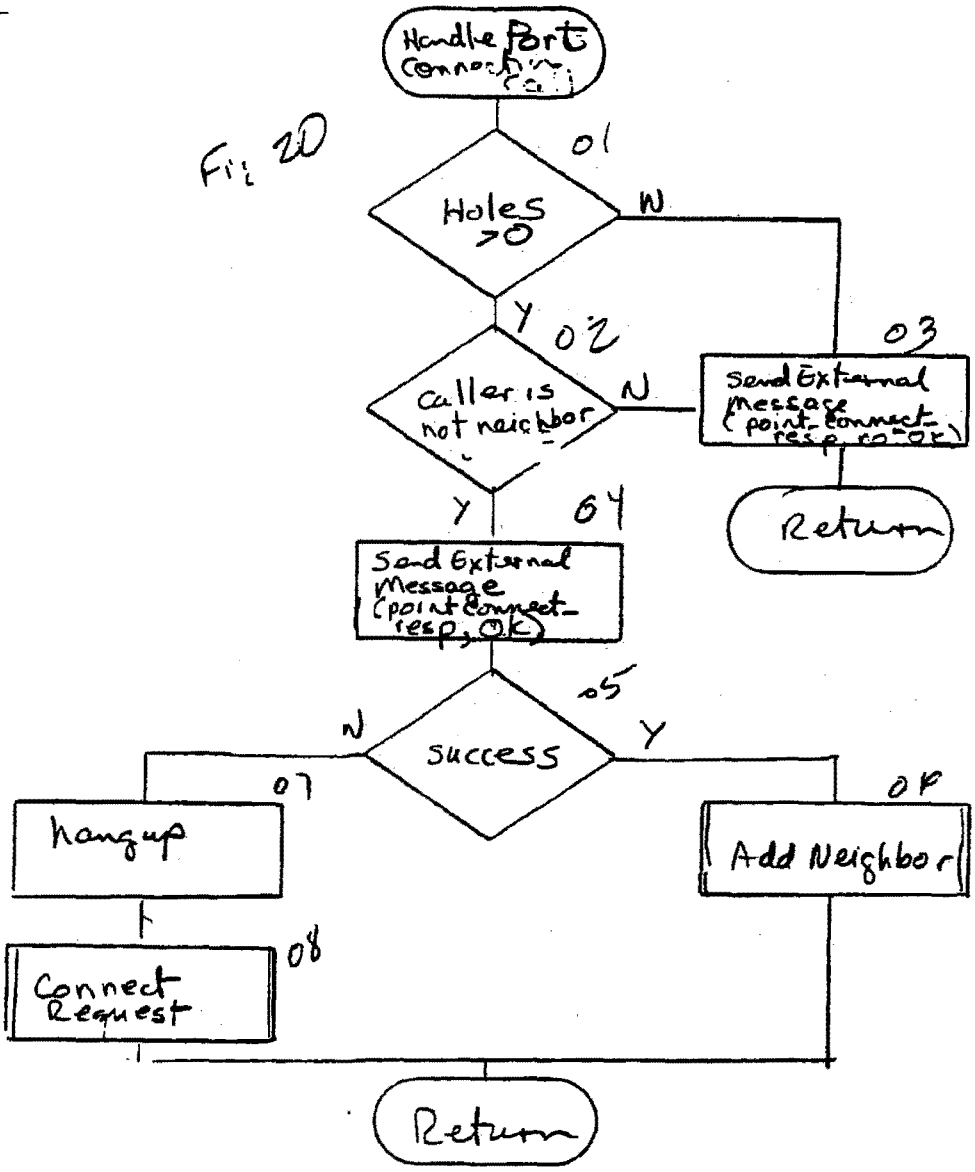
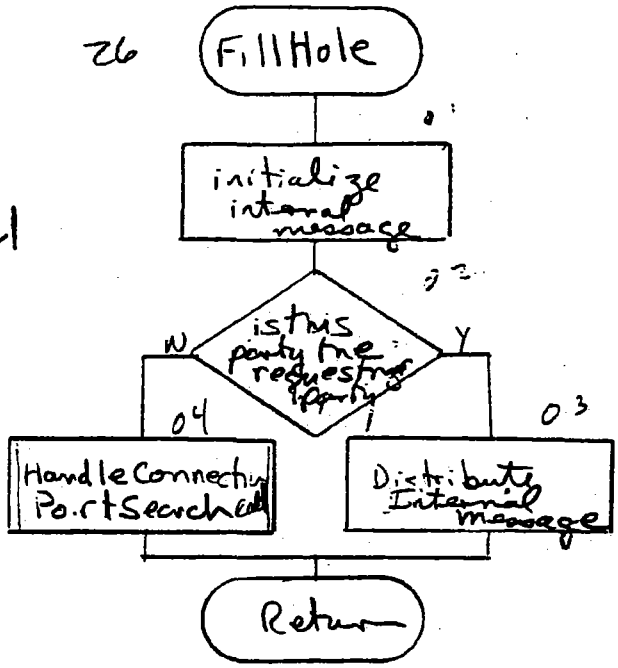

Handle Port Connection
01
Holes >0
N
Y
02
Caller is not neighbor
N
03
Send External message (point_connect_resp, ok=OK)
Return
Y
04
Send External message (point_connect_resp, OK)
05
Success
N
Y
07
hangup
06
Add Neighbor
Connect Request
08
Return

Fig 21


26
FillHole
initialize internal message
is this party the requesting party
N
Y
04
03
HandleConnectionPortSearch
Distribute Internal message
Return

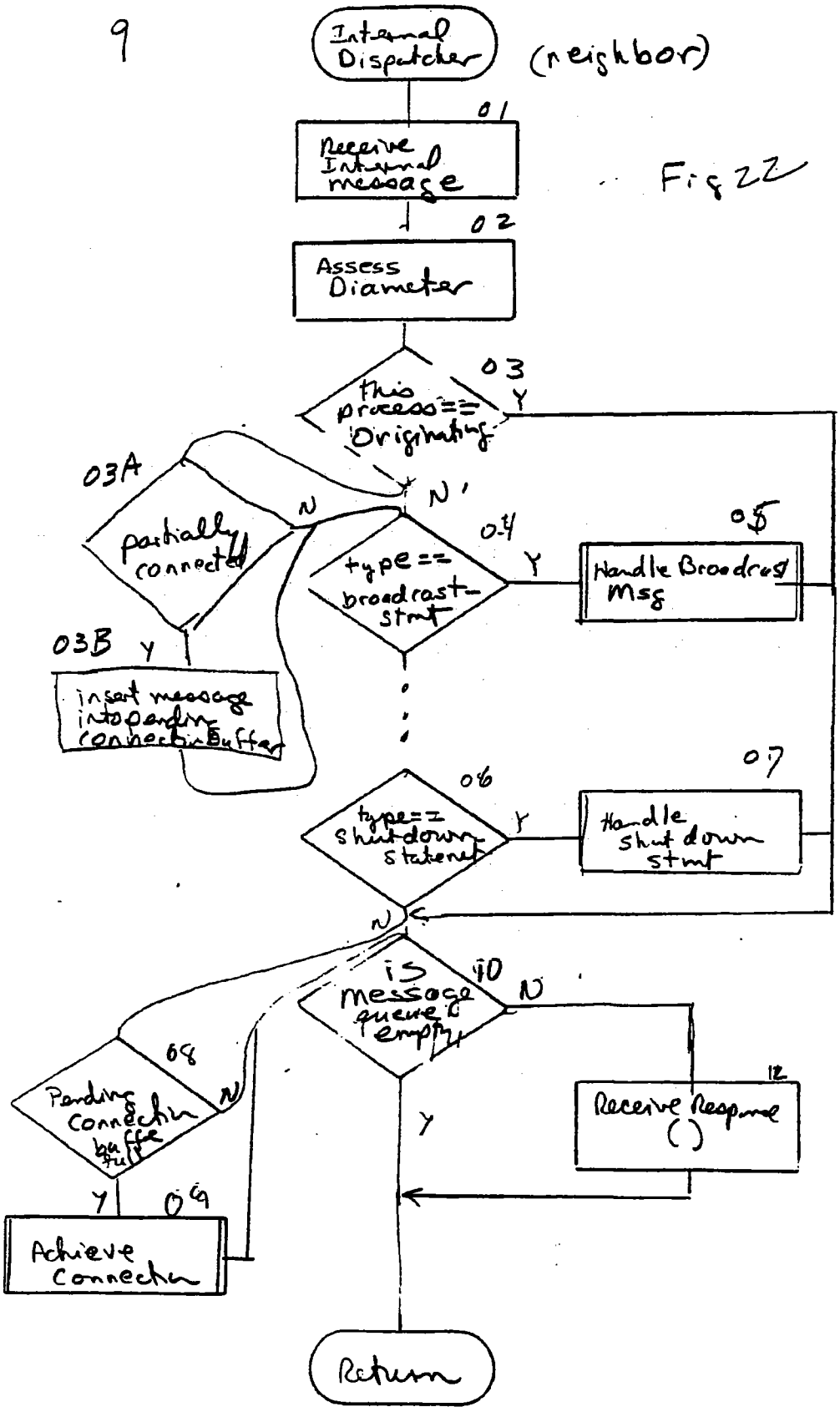

Internal Dispatcher
(neighbor)
01
Receive Internal message
02
Assess Diameter
03
this process == Originating
Y
N'
03A
partially connected
N
03B
Y
insert message into pending connection Buffer
04
type == broadcast-stmt
Y
05
Handle Broadcast Msg
06
type == shut-down statement
Y
07
Handle shut down stmt
N
10
is message queue empty
N
08
Pending Connection buffer full
N
Y
09
Achieve Connection
Y
12
Receive Response ()
Return

Fig 22

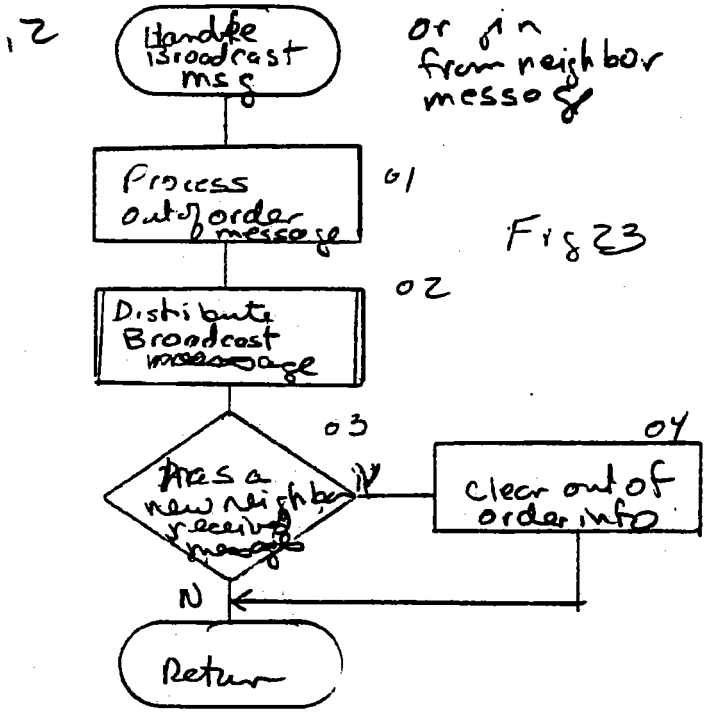

12
Handle broadcast msg
or from neighbor messg
Process out of order message
01
02
Distribute Broadcast message
03
Was a new neighbor receive msg
Y
04
Clear out of order info
N
Return

Fig 23

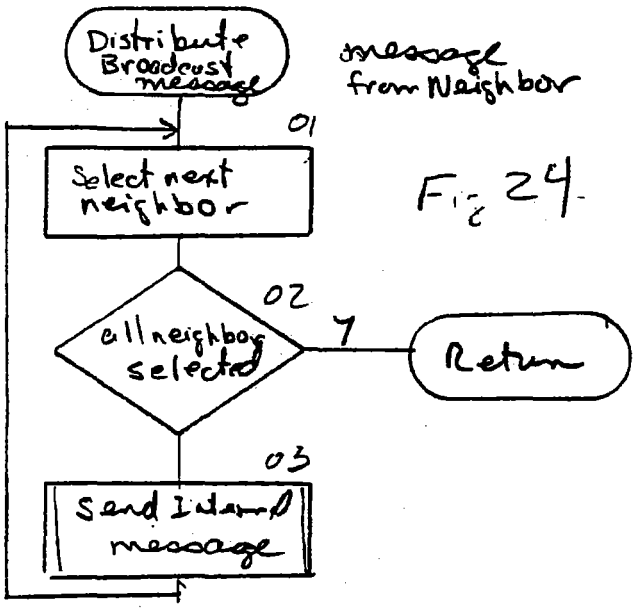

Distribute Broadcast message
message from Neighbor
01
Select next neighbor
02
all neighbors selected
Y
Return
03
Send Inband message

Fig 24.

13

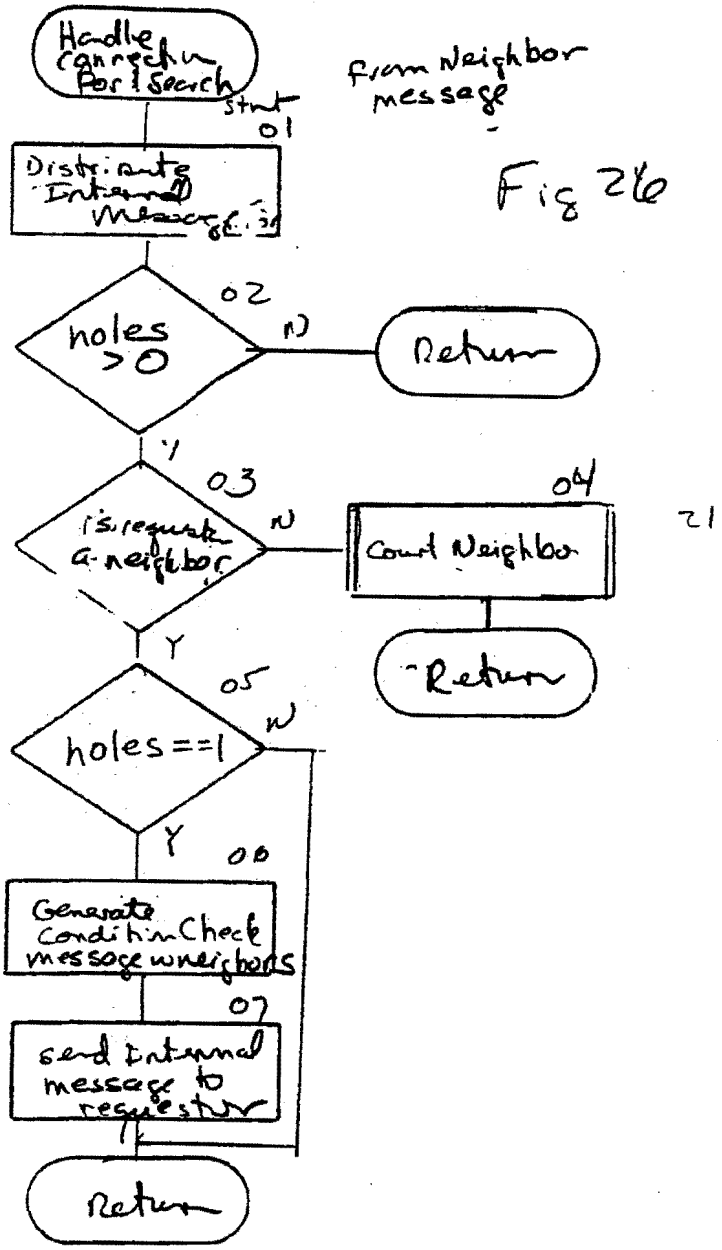

Handle Connection For Search
start
01
Distribute Internal Message
02
holes > 0
N
Return
Y
03
is request a neighbor
N
04
Count Neighbor
Return
Y
05
holes == 1
N
Y
06
Generate Condition Check message to neighbors
07
send Internal message to requestor
Return
from Neighbor message

Fig 26

21

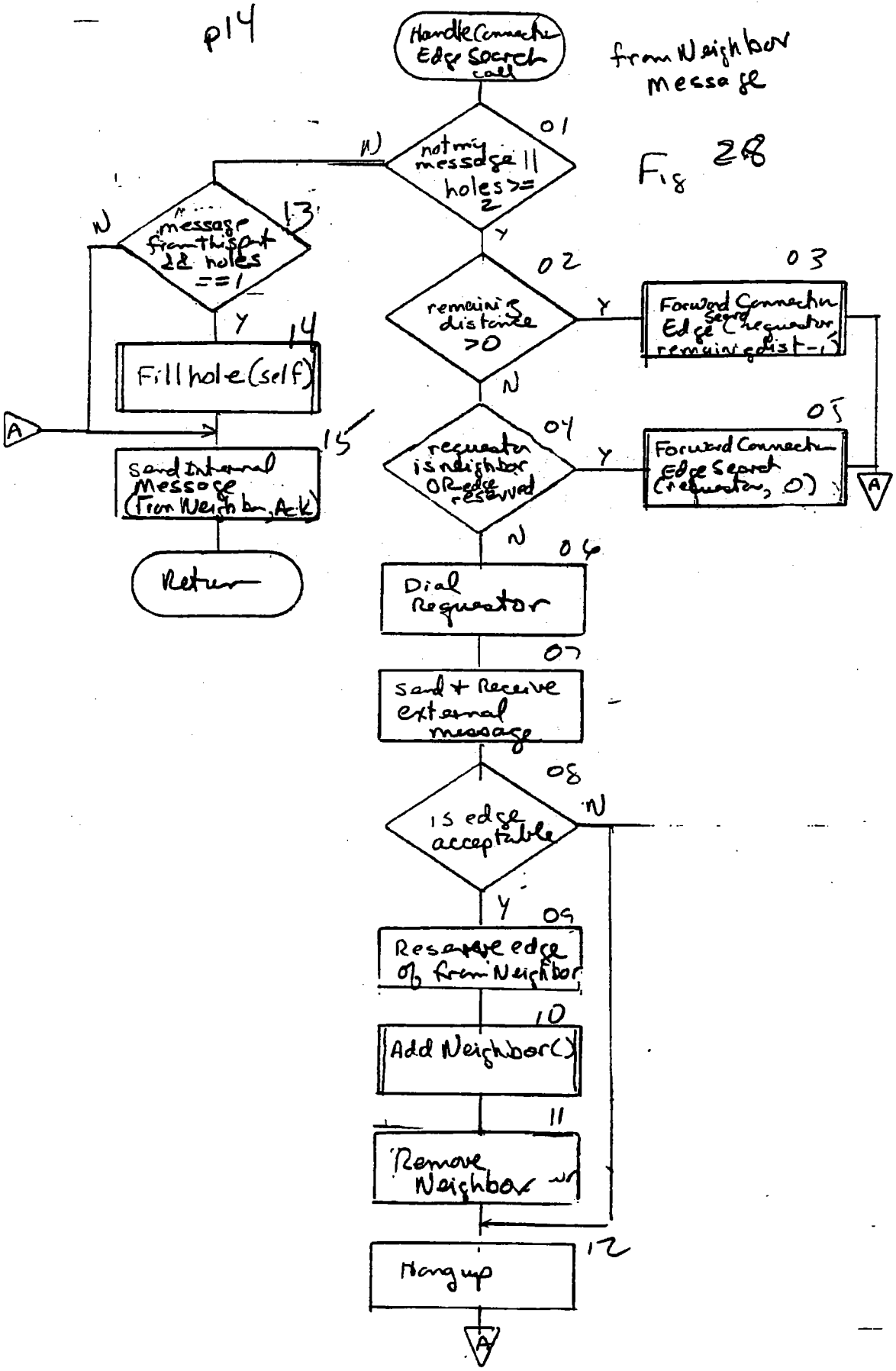

Handle Connection Edge Search call
from Neighbor message
01
not my message || holes >= 2
N
Y
13
message from this pt && holes == 1
N
Y
14
Fill hole (self)
A
15
Send Internal Message (From Neighbor, Ack)
Return
02
remaining distance >0
Y
03
Forward Connection Edge Search (requestor, remaining dist - 1)
N
04
requestor is neighbor OR edge reserved
Y
05
Forward Connection Edge Search (requestor, 0)
A
N
06
Dial Requestor
07
Send + Receive external message
08
is edge acceptable
N
Y
09
Reserve edge 0, from Neighbor
10
Add Neighbor()
11
Remove Neighbor
12
Hang up
A

Fig 2B

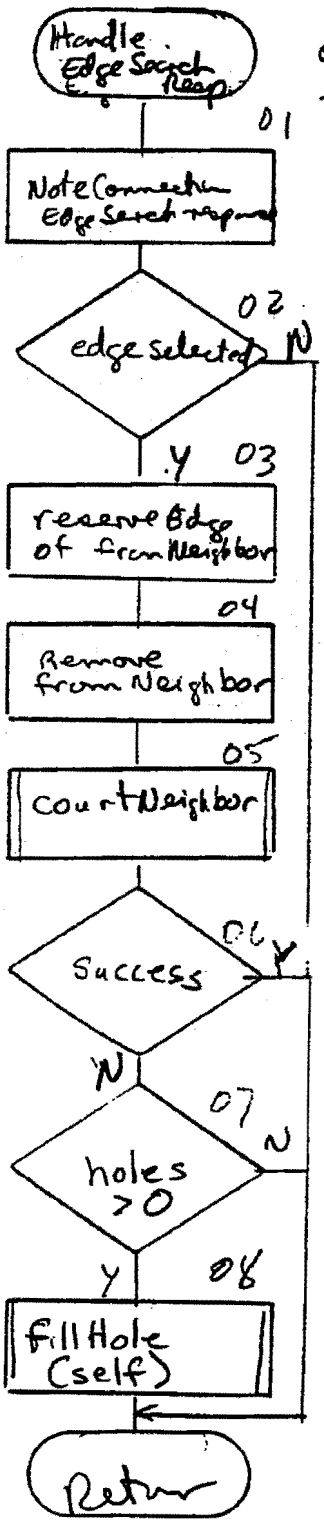

Handle Edge Search Reap
origin fromNeighbor message
01
Note Connection Edge Search response
02
edge selected?
N
Y
03
reserve Edge of fromNeighbor
04
Remove fromNeighbor
05
countNeighbor
06
Success
Y
N
07
holes >0
N
Y
08
fillHole (self)
Return

Fig 29

24

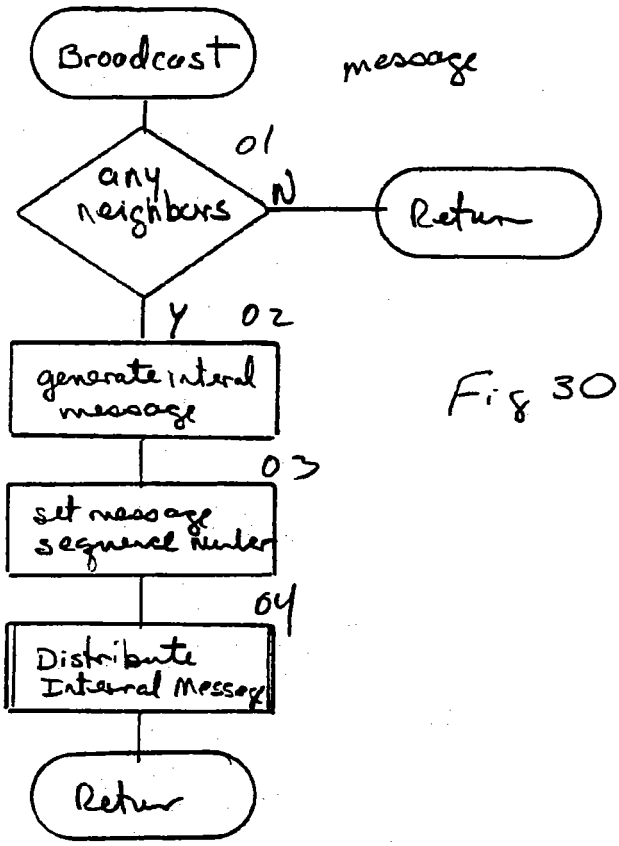

Broadcast message
any neighbors
01
N
Return
Y
02
generate Interal message
03
set message sequence number
04
Distribute Interal Message
Return

Fig 30

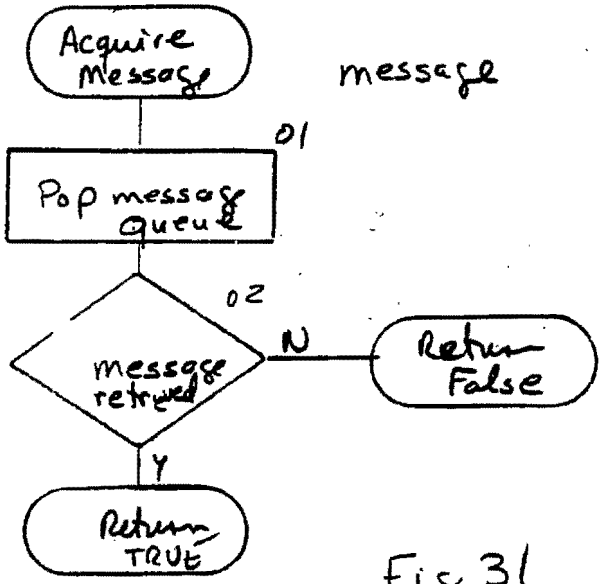
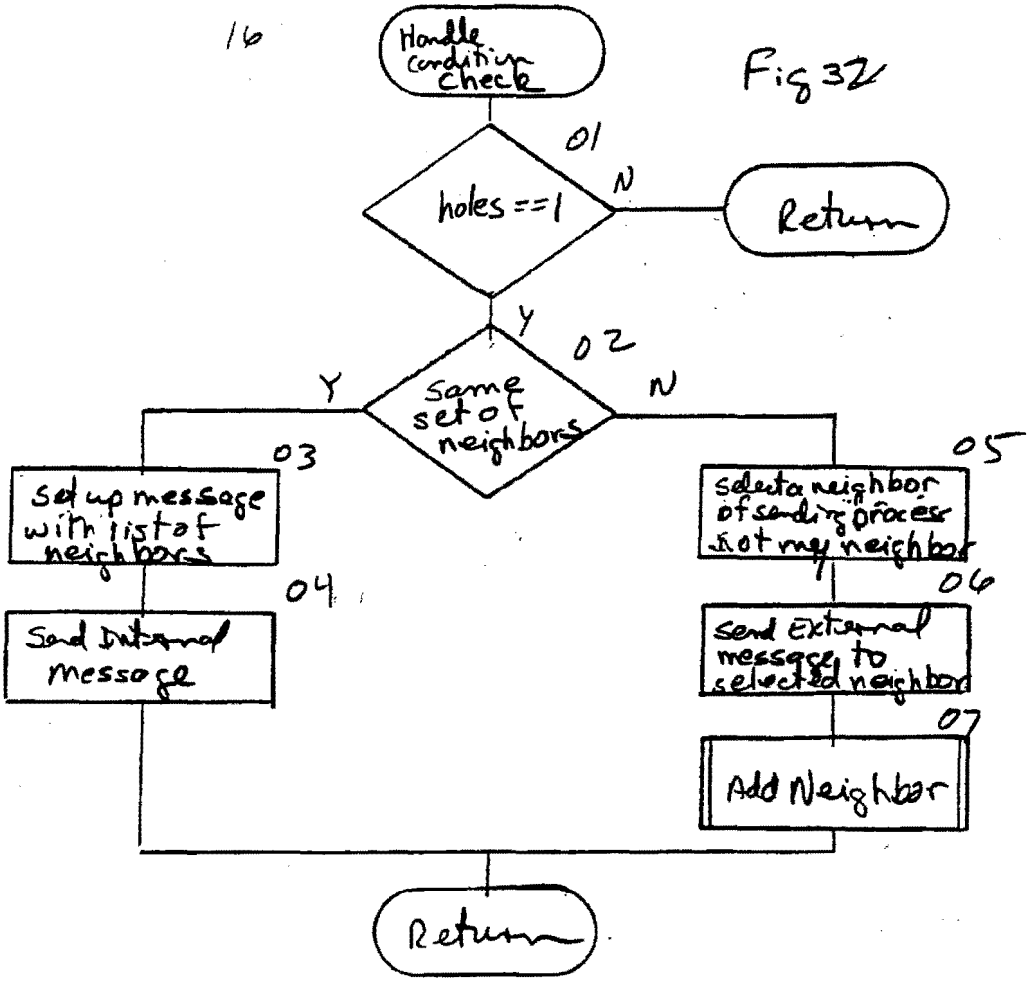

3
Acquire Message
message
01
Pop message Queue
02
message retrieved
N
Return False
Y
Return TRUE

Fig 31

Fig 3Z


Handle condition check
01
holes == 1
N
Return
Y
02
Same set of neighbors
Y
N
03
Set up message with list of neighbors
04
Send Internal message
05
select a neighbor of sending process not my neighbor
06
send External message to selected neighbor
07
Add Neighbor
Return

23

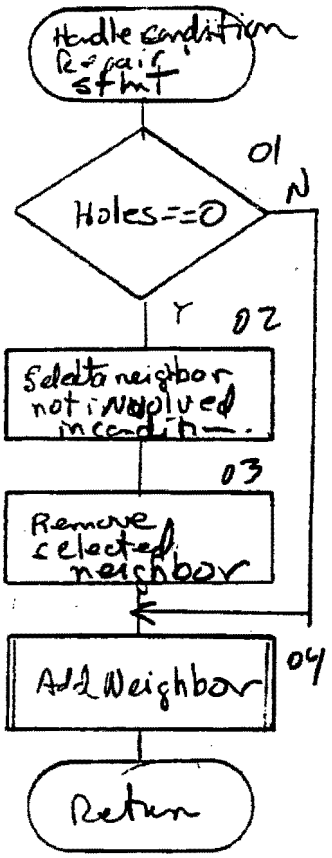
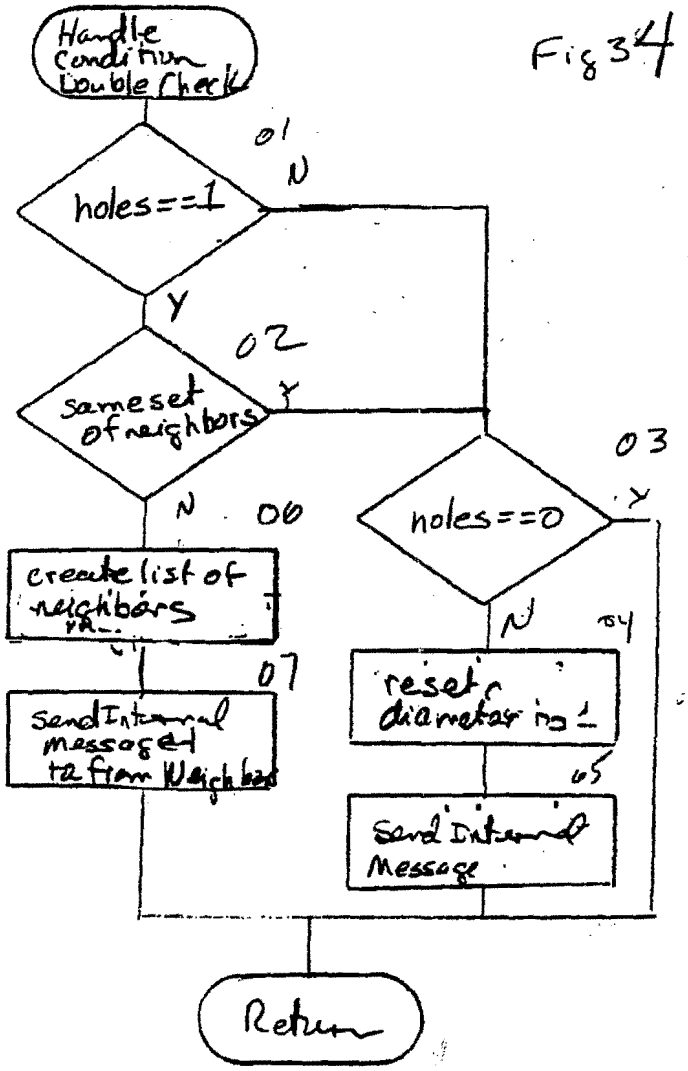

Handle condition
Repair
stmt
01
Holes==0
N
Y
02
Select a neighbor
not involved
in condition.
03
Remove
selected
neighbor
Add Neighbor
04
Return

Fig 33

17

Fig 34


Handle condition Double check
01
holes==1
N
Y
02
same set of neighbors
Y
N
03
holes==0
Y
N
06
create list of neighbors
07
Send Internal message to/from Neighbors
04
reset diameter to 1
05
Send Internal Message
Return

DISTRIBUTED CONFERENCING SYSTEM

CROSS-REFERENCE TO RELATED APPLICATIONS

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

TECHNICAL FIELD

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

BACKGROUND

There are a wide variety of computer network communications techniques such as point-to-point network protocols, client/server middleware, multicasting network

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

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

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

The multicasting network protocols allow the sending of broadcast messages to multiple recipients of a network. The current implementations of such multicasting network

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1023 of 1657

protocols tend to place an unacceptable overhead on the underlying network. For example, UDP multicasting would swamp the Internet when trying to locate all possible participants. IP multicasting has other problems that include needing special-purpose infrastructure (*e.g.*, routers) to support the sharing of information efficiently.

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

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

BRIEF DESCRIPTION OF THE DRAWINGS

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

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

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

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

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1024 of 1657

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

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

Figure 5C illustrates the neighbors with empty ports condition.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1025 of 1657

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1026 of 1657

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

DETAILED DESCRIPTION

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1027 of 1657

divide the graph into disjoint sub-graphs, that is two separate broadcast channels. This property is referred to as being 4-connected. Thus, the graph is both 4-regular and 4-connected.

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

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

Composing the Graph

To connect to the broadcast channel, the computer seeking the connection first locates a computer that is currently fully connected to the broadcast channel and then

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1028 of 1657

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

Since the broadcast channel is a 4-regular graph, each of the identified computers is already connected to four computers. Thus, some connections between computers need to be broken so that the seeking computer can connect to four computers. In one embodiment, the broadcast technique identifies two pairs of computers that are currently connected to each other. Each of these pairs of computers breaks the connection between them, and then each of the four computers (two from each pair) connects to the seeking computer. Figures 3A and 3B illustrate the process of a new computer Z connecting to the broadcast channel. Figure 3A illustrates the broadcast channel before computer Z is connected. The pairs of computers B and E and computers C and D are the two pairs that are identified as the neighbors for the new computer Z. The connections between each of these pairs is broken, and a connection between computer Z and each of computers B, C, D, and E

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1029 of 1657

is established as indicated by Figure 3B. The process of breaking the connection between two neighbors and reconnecting each of the former neighbors to another computer is referred to as "edge pinning" as the edge between two nodes may be considered to be stretched and pinned to a new node.

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1030 of 1657

seeking computer actually communicates through that transfer-to port, which is the external port. The call is transferred so that other computers can place calls to that computer via the call-in port. The seeking computer then communicates via that external port to request the portal computer to assist in connecting the seeking computer to the broadcast channel. The seeking computer could identify the call-in port number of a portal computer by successively dialing each port in port number order. As discussed below in detail, the broadcast technique uses a hashing algorithm to select the port number order, which may result in improved performance.

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1031 of 1657

Broadcasting Through the Graph

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

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

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

When the broadcast channel is in a steady state (*i.e.*, no computers connecting or disconnecting from the broadcast channel), out-of-order messages are not a problem because each computer will eventually receive both messages and can queue messages until all earlier ordered messages are received. If, however, the broadcast channel is not in a

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1032 of 1657

steady state, then problems can occur. In particular, a computer may connect to the broadcast channel after the second message has already been received and forwarded on by its new neighbors. When a new neighbor eventually receives the first message, it sends the message to the newly connected computer. Thus, the newly connected computer will receive the first message, but will not receive the second message. If the newly connected computer needs to process the messages in order, it would wait indefinitely for the second message.

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1033 of 1657

same originating computer through another neighbor. If the second set of messages contains a message that is ordered earlier than the messages of the first set received, then the newly connected computer may ignore that earlier ordered message if the computer already processed those later ordered messages.

Decomposing the Graph

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1034 of 1657

computer that is also short a connection receives the connection request, it communicates with the requesting computer through its external port to establish a connection between the two computers. Figure 5B illustrates the disconnecting of a computer from the broadcast channel in an unplanned manner. In this illustration, computer H has disconnected in an unplanned manner. When each of its neighbors, computers A, E, F, and I, recognizes the disconnection, each neighbor broadcasts a port connection request indicating that it needs to fill an empty port. As shown by the dashed lines, computers F and I and computers A and E respond to each other's requests and establish a connection.

It is possible that a planned or unplanned disconnection may result in two neighbors each having an empty internal port. In such a case, since they are neighbors, they are already connected and cannot fill their empty ports by connecting to each other. Such a condition is referred to as the "neighbors with empty ports" condition. Each neighbor broadcasts a port connection request when it detects that it has an empty port as described above. When a neighbor receives the port connection request from the other neighbor, it will recognize the condition that its neighbor also has an empty port. Such a condition may also occur when the broadcast channel is in the small regime. The condition can only be corrected when in the large regime. When in the small regime, each computer will have less than four neighbors. To detect this condition in the large regime, which would be a problem if not repaired, the first neighbor to receive the port connection request recognizes the condition and sends a condition check message to the other neighbor. The condition check message includes a list of the neighbors of the sending computer. When the receiving computer receives the list, it compares the list to its own list of neighbors. If the lists are different, then this condition has occurred in the large regime and repair is needed. To repair this condition, the receiving computer will send a condition repair request to one of the neighbors of the sending computer which is not already a neighbor of the receiving computer. When the computer receives the condition repair request, it disconnects from one of its neighbors (other than the neighbor that is involved with the condition) and connects to the computer that sent the condition repair request. Thus, one of the original neighbors involved in the condition will have had a port filled. However, two computers are still in need of a connection, the other original neighbor and the computer that is now disconnected from the computer that received the condition repair request. Those two computers send out port connection requests. If those two computers are not neighbors, then they will connect to

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1035 of 1657

each other when they receive the requests. If, however, the two computers are neighbors, then they repeat the condition repair process until two non-neighbors are in need of connections.

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

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

Figures 5E and 5F further illustrate the neighbors with empty ports condition. Figure 5E illustrates the neighbors with empty ports condition in the small regime. In this

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1036 of 1657

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

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

Port Selection

As described above, the TCP/IP protocol designates ports above number 2056 as user ports. The broadcast technique uses five user port numbers on each computer: one external port and four internal ports. Generally, user ports cannot be statically allocated to an application program because other applications programs executing on the same computer may use conflicting port numbers. As a result, in one embodiment, the computers connected to the broadcast channel dynamically allocate their port numbers. Each computer could simply try to locate the lowest number unused port on that computer and use that port as the call-in port. A seeking computer, however, does not know in advance the call-in port number of the portal computers when the port numbers are dynamically allocated. Thus, a seeking computer needs to dial ports of a portal computer starting with the lowest port number when locating the call-in port of a portal computer. If the portal computer is

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1037 of 1657

connected to (or attempting to connect to) the broadcast channel, then the seeking computer would eventually find the call-in port. If the portal computer is not connected, then the seeking computer would eventually dial every user port. In addition, if each application program on a computer tried to allocate low-ordered port numbers, then a portal computer may end up with a high-numbered port for its call-in port because many of the low-ordered port numbers would be used by other application programs. Since the dialing of a port is a relatively slow process, it would take the seeking computer a long time to locate the call-in port of a portal computer. To minimize this time, the broadcast technique uses a port ordering algorithm to identify the port number order that a portal computer should use when finding an available port for its call-in port. In one embodiment, the broadcast technique uses a hashing algorithm to identify the port order. The algorithm preferably distributes the ordering of the port numbers randomly through out the user port number space and only selects each port number once. In addition, every time the algorithm is executed on any computer for a given channel type and channel instance, it generates the same port ordering. As described below, it is possible for a computer to be connected to multiple broadcast channels that are uniquely identified by channel type and channel instance. The algorithm may be "seeded" with channel type and channel instance in order to generate a unique ordering of port numbers for each broadcast channel. Thus, a seeking computer will dial the ports of a portal computer in the same order as the portal computer used when allocating its call-in port.

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1038 of 1657

possible that eight seeking computers could be simultaneously dialing ports in different sequences which would reduce the chances of dialing a busy port.

Locating a Portal Computer

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1039 of 1657

then the one seeking computer can share that it has searched to a depth of eight with another seeking computer. If that other seeking computer has searched to a depth of, for example, only four, it can skip searching through depths five through eight and that other seeking computer can advance its searching to a depth of nine.

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

Identifying Neighbors for a Seeking Computer

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1040 of 1657

traveled far enough will detect this condition of already being a neighbor and send the message to a randomly selected neighbor.

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

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

External Data Representation

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1041 of 1657

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

M-Regular

In the embodiment described above, each fully connected computer has four internal connections. The broadcast technique can be used with other numbers of internal connections. For example, each computer could have 6, 8, or any even number of internal connections. As the number of internal connections increase, the diameter of the broadcast channel tends to decrease, and thus propagation time for a message tends to decrease. The time that it takes to connect a seeking computer to the broadcast channel may, however, increase as the number of internal connections increases. When the number of internal connectors is even, then the broadcast channel can be maintained as m-regular and m-connected (in the steady state). If the number of internal connections is odd, then when the broadcast channel has an odd number of computers connected, one of the computers will have less than that odd number of internal connections. In such a situation, the broadcast network is neither m-regular nor m-connected. When the next computer connects to the broadcast channel, it can again become m-regular and m-connected. Thus, with an odd number of internal connections, the broadcast channel toggles between being and not being m-regular and m-connected.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1042 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1043 of 1657

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 Conferencing System

In one embodiment, a conferencing system is implemented using the broadcast channel. Each participant in a conference connects to the conference's broadcast channel,

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1044 of 1657

and a participant is designated as the speaker. The conferencing application program may include a speaker component and an attendee component. The speaker component broadcasts the conference events on the broadcast channel. Each attendee component receives the conference events and displays the results of the conference events. For example, the speaker may present slides at the conference along with a description of each slide. Each attendee may receive an electronic copy of the slides in advance of the conference. At the scheduled time for the conference, the speaker and each attendee joins the conference by connecting to the broadcast channel of the conference. The speaker component allows the speaker to indicate when to display which slide. When a new slide is displayed, the speaker component broadcasts a new slide message. When the attendee component receives the new slide message, it displays the new slide to the participant. Also, the speaker component may allow the speaker to draw on a slide using a stylus or other pointing device. The speaker component then broadcasts draw messages on the broadcast channel so the attendee component can display the drawing to the attendees. The conferencing system may also use speech-to-text and text-to-speech to distribute the speaker's comments to all attendees.

The conferencing system may provide a directory web site where participants can locate and sign up for a conference of interest. The directory may provide a hierarchical categorization of scheduled conferences. When a user decides to sign up for a conference, the web server may download the broadcaster component and the conferencing application program to the attendee's computer, if not already stored on the attendee's computer. The web server will also download the channel type and channel instance associated with the broadcast channel for the conference along with the identification of the portal computers for the broadcast channel. The web server may also download the slides or other content to be displayed to the attendees during the conference.

The conferencing system may allow an entity to schedule conferences using the web site. For example, a software company may want to schedule a conference to announce a new product. The creation of the conference would entail the generation of a channel type and channel instance, the specification of a security level (e.g., encrypted messages), the specification of attendee qualifications, the providing of a description and scheduled time of the conference, the specification of the content to be distributed to the attendees, and so on. The speaker at a conference may not want to publicize the actual

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1045 of 1657

content (e.g., slides) in advance. In such a situation, the content can be encrypted when distributed to the attendees, and a key to decrypt the content can be distributed by the speaker during the conference. For example, each slide for the software company's announcement can be encrypted with a different key, and the appropriate key can be broadcast with each new slide message.

The conferencing system may allow attendees to broadcast comments on the broadcast channel. The times when an attendee can broadcast comments may be controlled by the speaker. For example, the speaker component may broadcast a comments allowed message and a comments not allowed message to delimit the times when comments will be allowed. Comments broadcast outside those times may be ignored. Alternatively, the attendees may be allowed to broadcast comments at any time, but the other attendees ignore those comments until the speaker broadcasts an approval message indicating that the attendee component can display a certain comment.

The conferencing system may allow each attendee to connect to and disconnect from the conference broadcast channel as this wish during the conference. In addition, the conferencing system may allow multiple speakers to share the "podium." The speakers can pass a speakers token between them to indicate who is currently speaking and thus in control of the conference. An attendee who joins the conference late may be able to synchronize with the conference by accessing a conference monitoring web server. The monitoring web server may be connected to the conference broadcast channel and monitor the current state of the conference. When an attendee joins late, the monitoring web server can provide the attendee with the current state of the conference. From then on, the attendee can listen on the broadcast channel to follow the progress of the conference. In addition, the attendee component may allow the attendee to view parts of the presentation other than that which is currently being presented. In this way, an attendee can refer back to or ahead to other portions of the presentation.

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1046 of 1657

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

INTERNAL MESSAGES

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1047 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1048 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1049 of 1657

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. 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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1050 of 1657

1102, the routine restarts the process of connecting to the broadcast channel and returns. In block 1103, the routine dials the process of one of the found portal computers through the call-in port. In decision block 1104, if the dialing is successful, then the routine continues at block 1105, else the routine continues at block 1113. The dialing may be unsuccessful if, for example, the dialed process recently disconnected from the broadcast channel. In block 1105, the routine sends an external message to the dialed process requesting a connection to the broadcast channel (*i.e.*, connection_request_call). In block 1106, the routine receives the response message (*i.e.*, connection_request_resp). In decision block 1107, if the response message is successfully received, then the routine continues at block 1108, else the routine continues at block 1113. In block 1108, the routine sets the expected number of holes (*i.e.*, empty internal connections) for this process based on the received response. When in the large regime, the expected number of holes is zero. When in the small regime, the expected number of holes varies from one to three. In block 1109, the routine sets the estimated diameter of the broadcast channel based on the received response. In decision block 1111, if the dialed process is ready to connect to this process as indicated by the response message, then the routine continues at block 1112, else the routine continues at block 1113. In block 1112, the routine invokes the add neighbor routine to add the answering process as a neighbor to this process. This adding of the answering process typically occurs when the broadcast channel is in the small regime. When in the large regime, the random walk search for a neighbor is performed. In block 1113, the routine hangs up the external connection with the answering process computer and then returns.

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 process is attempting to establish a connection to the broadcast channel through this process. In block 1201, the routine attempts to answer a call on the call-in port. In decision block 1202, if the answer is successful, then the routine continues at block 1203, else the routine returns. In block 1203, the routine receives the external message from the external port. In decision block 1204, if the type of the message indicates that a seeking process is calling (*i.e.*, seeking_connection_call), then the routine continues at block 1205, else the routine returns. In block 1205, the routine sends an external message (*i.e.*, seeking_connection_resp) to the other seeking process indicating that this process is also is seeking a connection. In decision block 1206, if the sending of the external message is successful, then the routine

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1051 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1052 of 1657

type is a connected statement (*i.e.*, connected_stmt), the routine invokes the handle connected statement in block 1112, else the routine continues at block 1212. In decision block 1412, if the message type is a condition repair statement (*i.e.*, condition_repair_stmt), then the routine invokes the handle condition repair routine in block 1413, else the routine loops to block 1414 to process the next message. After each handling routine is invoked, the routine loops to block 1414. In block 1414, the routine hangs up on the external port and continues at block 1401 to receive the next message.

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1053 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1054 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1055 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1056 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1057 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1058 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1059 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1060 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1061 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1062 of 1657

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.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1063 of 1657

CLAIMS

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

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

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

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

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

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

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1064 of 1657

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

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

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

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

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

14. A distributed conferencing system comprising:

a plurality of broadcast channels, each broadcast channel for conducting a conference;

means for identifying a broadcast channel for a conference of interest; and

means for connecting to the identified broadcast channel.

15. The distributed conferencing system of claim 14 wherein means for identifying a conference of interest includes accessing a web server that maps conferences to corresponding broadcast channel.

16. The distributed conferencing system of claim 14 wherein a broadcast channel is formed by attendee computers and a speaker computer that are each interconnected to at least three other computers.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1065 of 1657

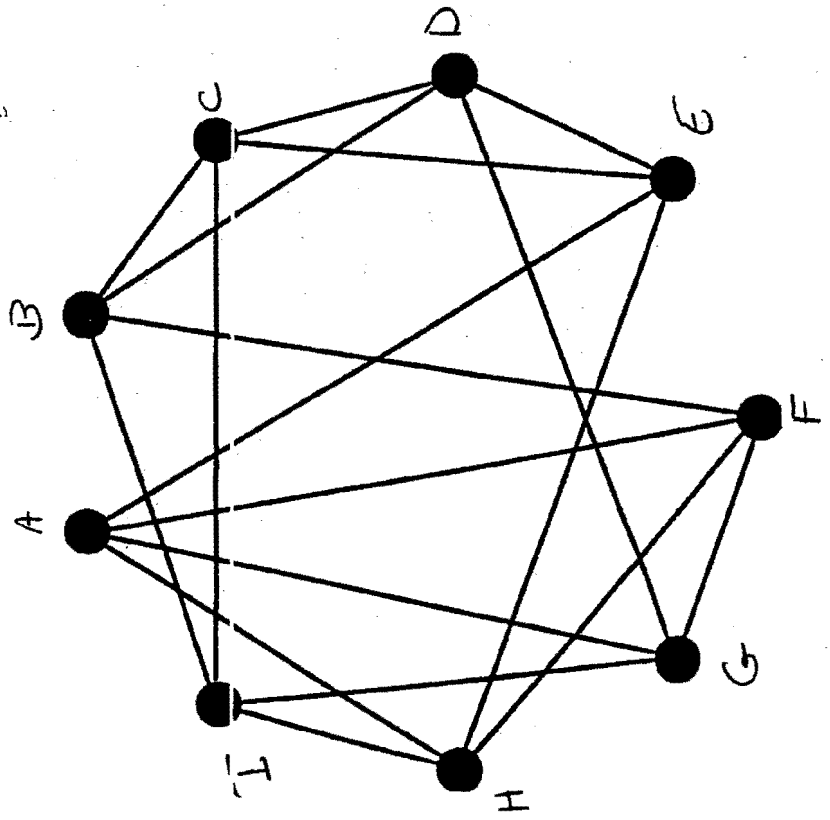


Fig 1

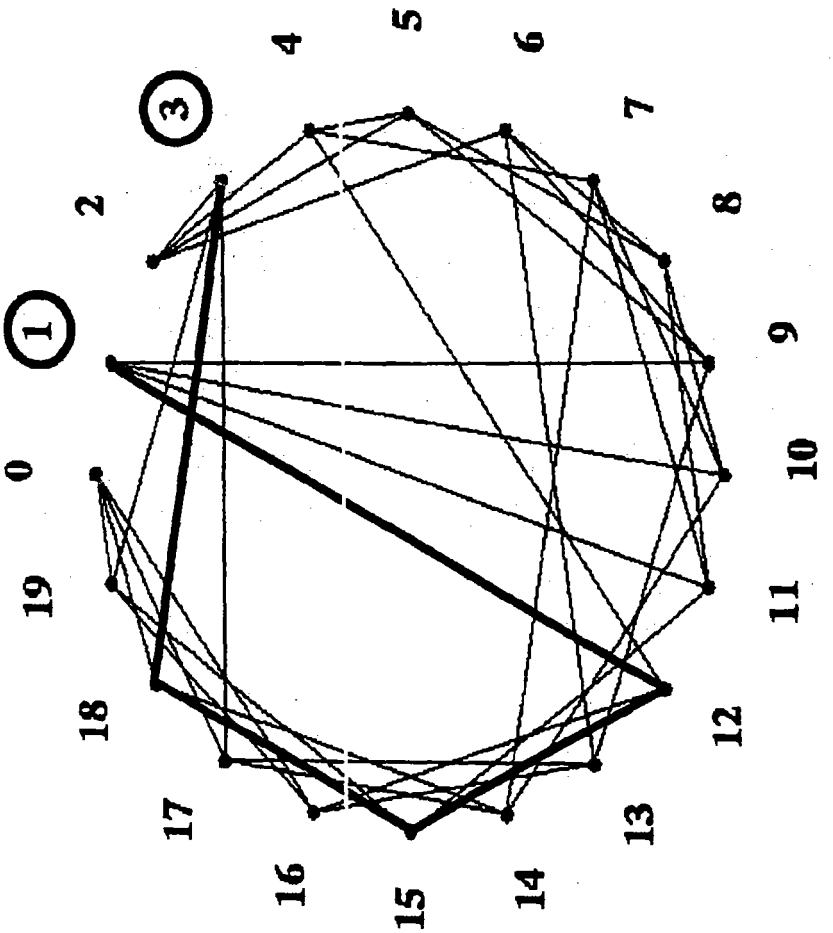


Fig 2

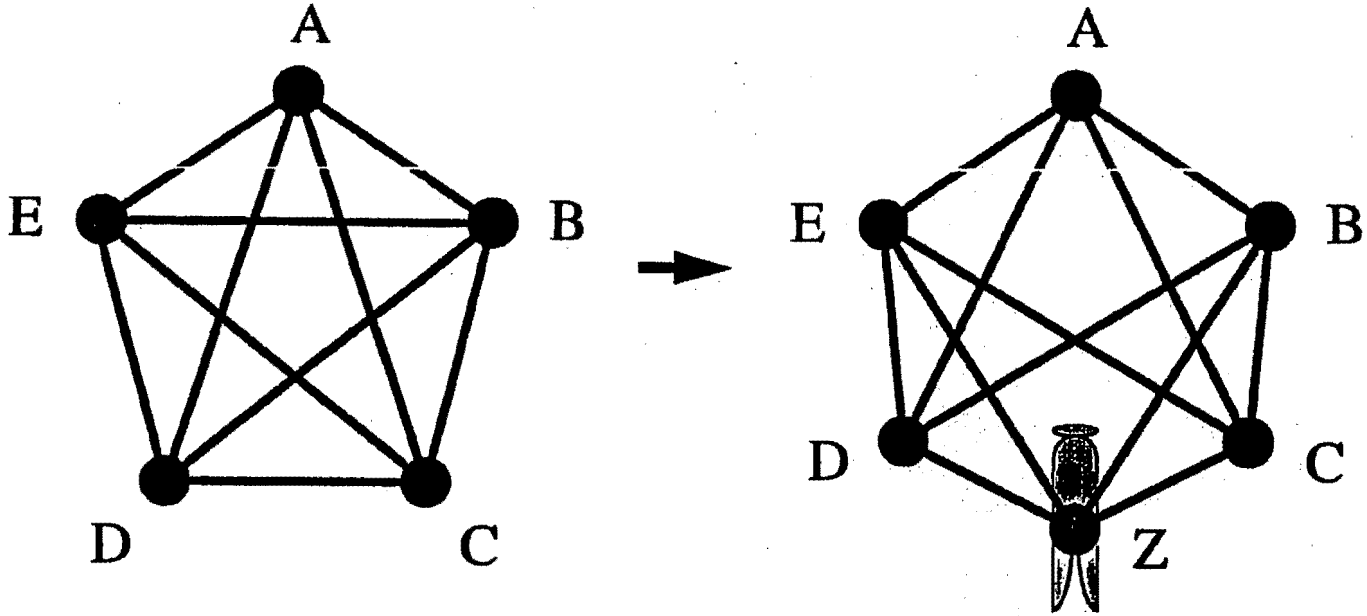


Fig 3A

Fig 3B

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1068 of 1657

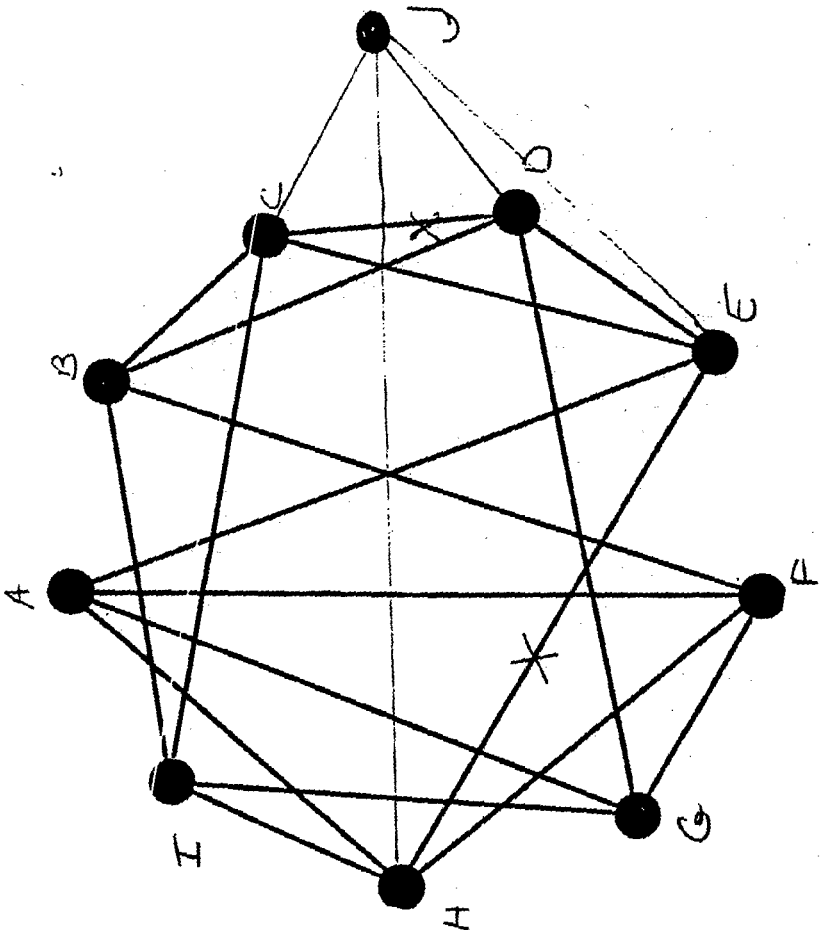


Fig 4A

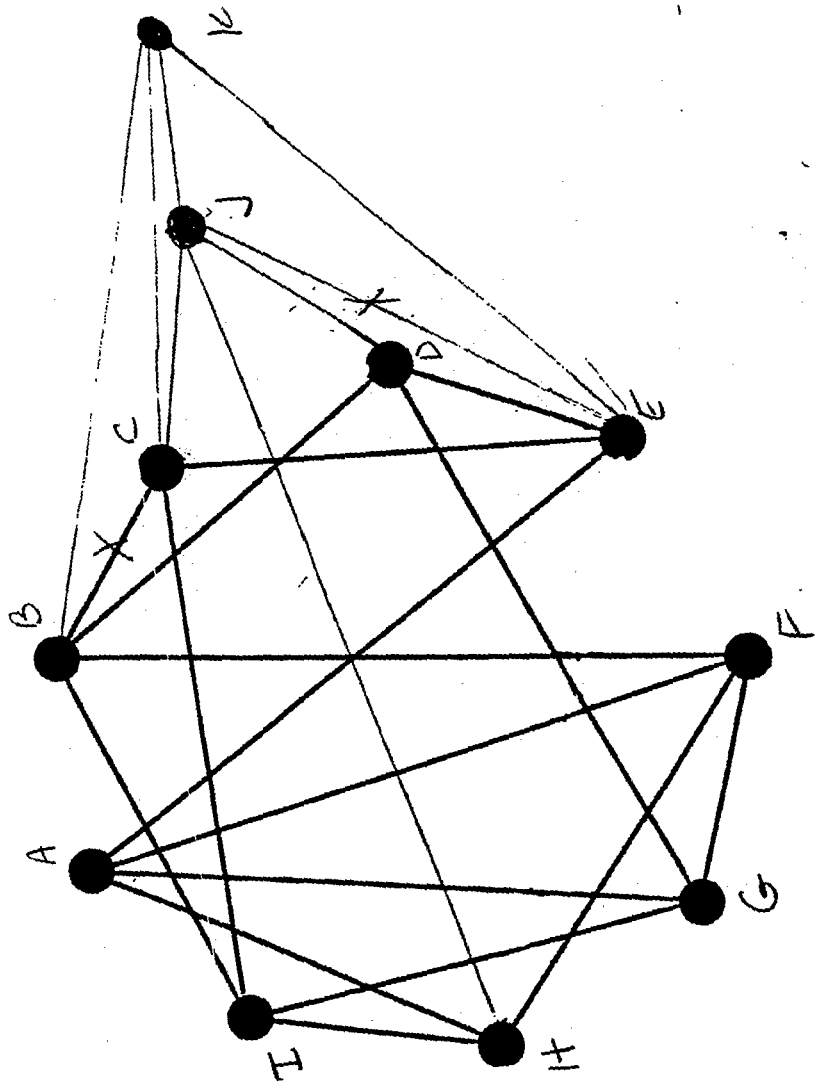


Fig 4B

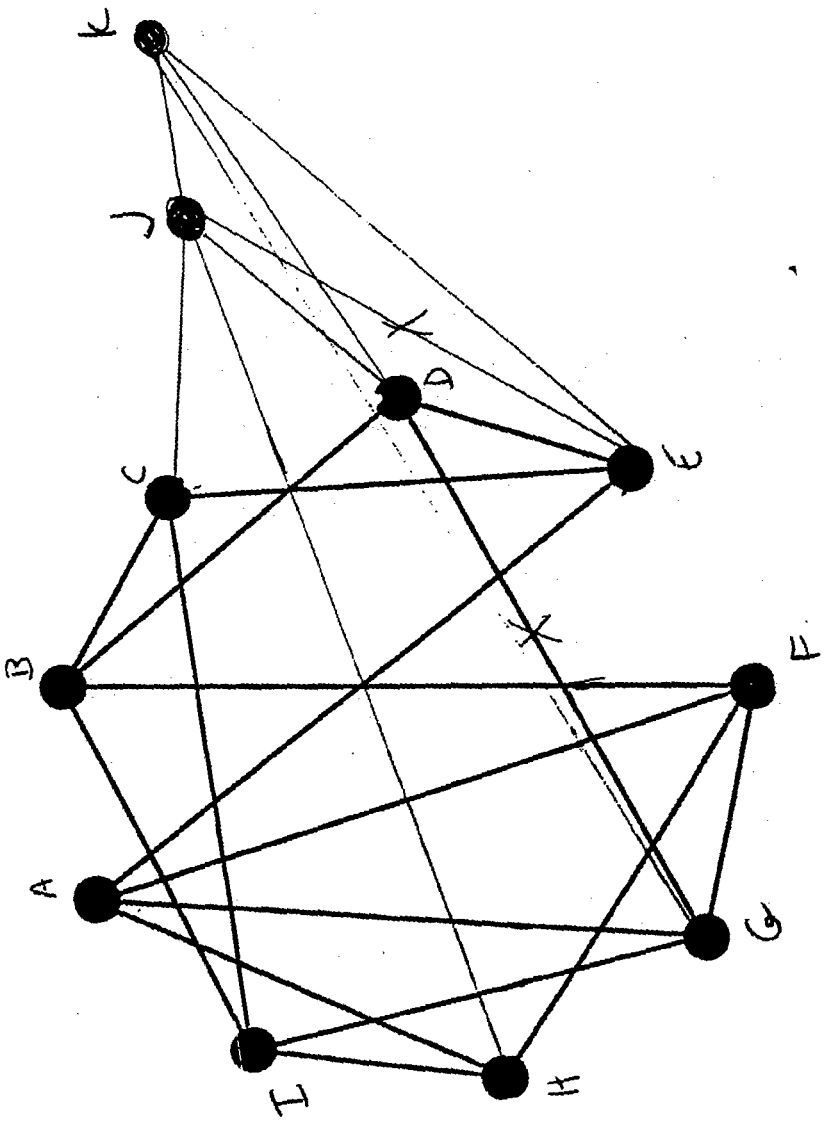


Fig 4C

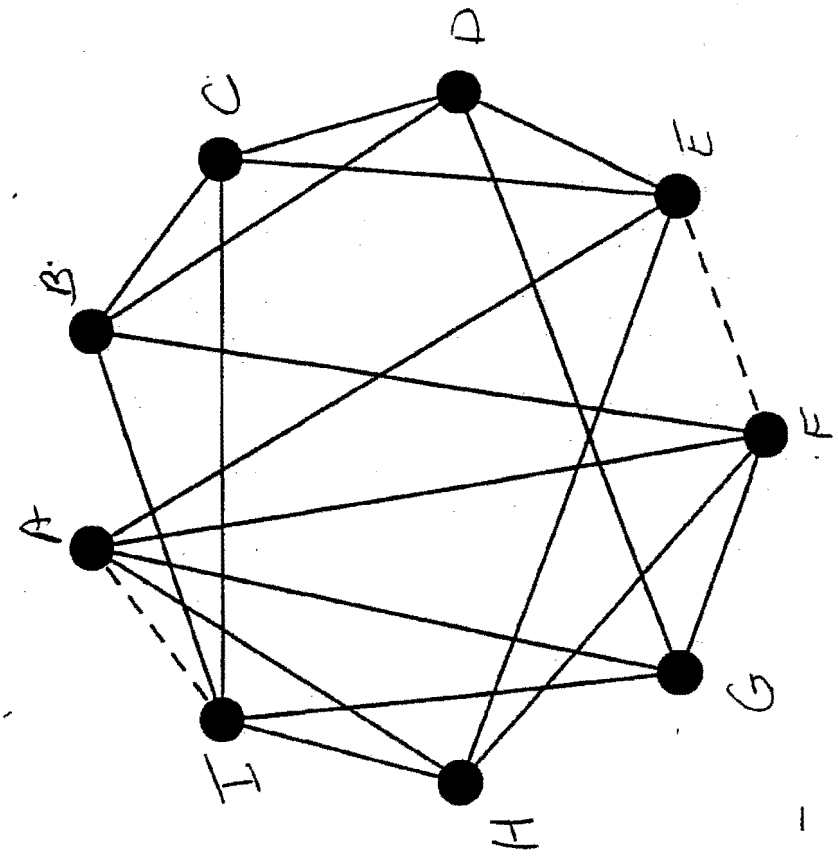


Fig 5A

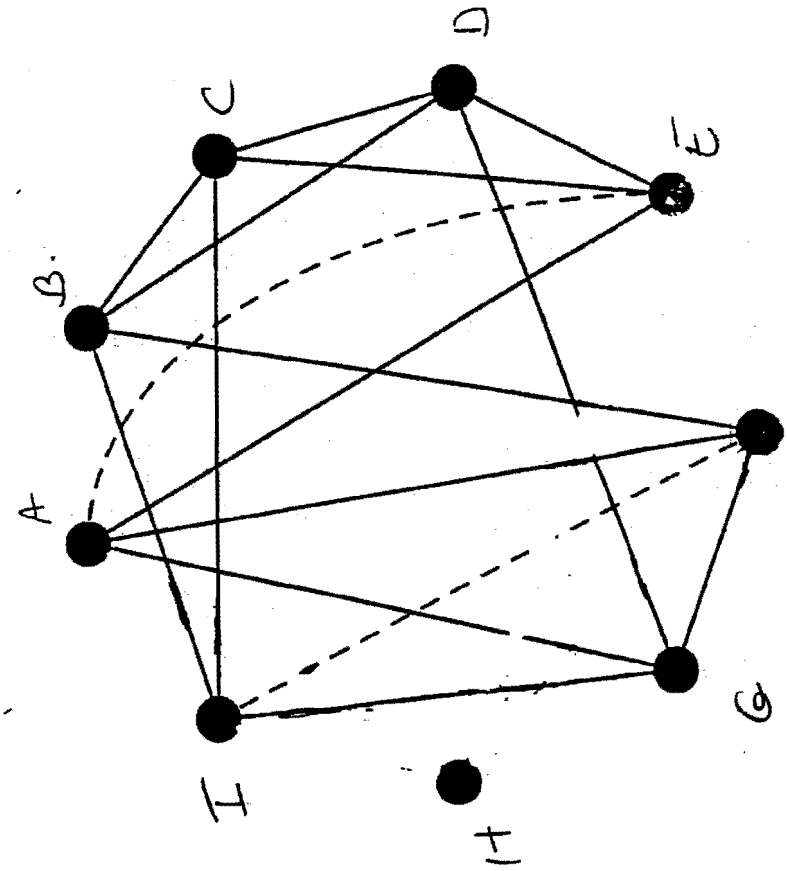


Fig 5B

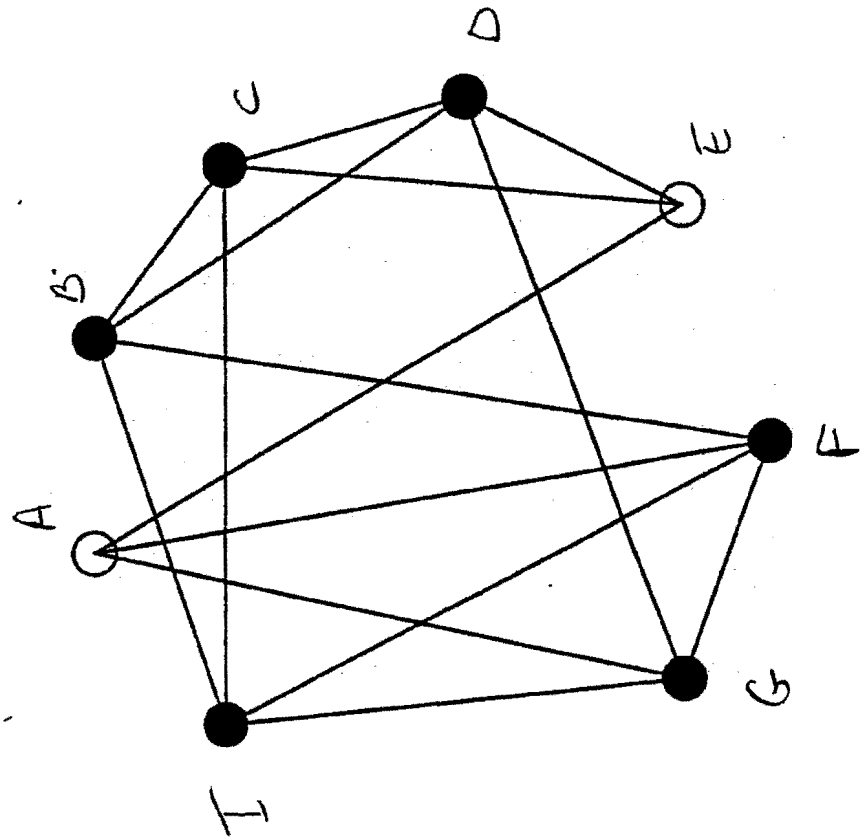


Fig 5C

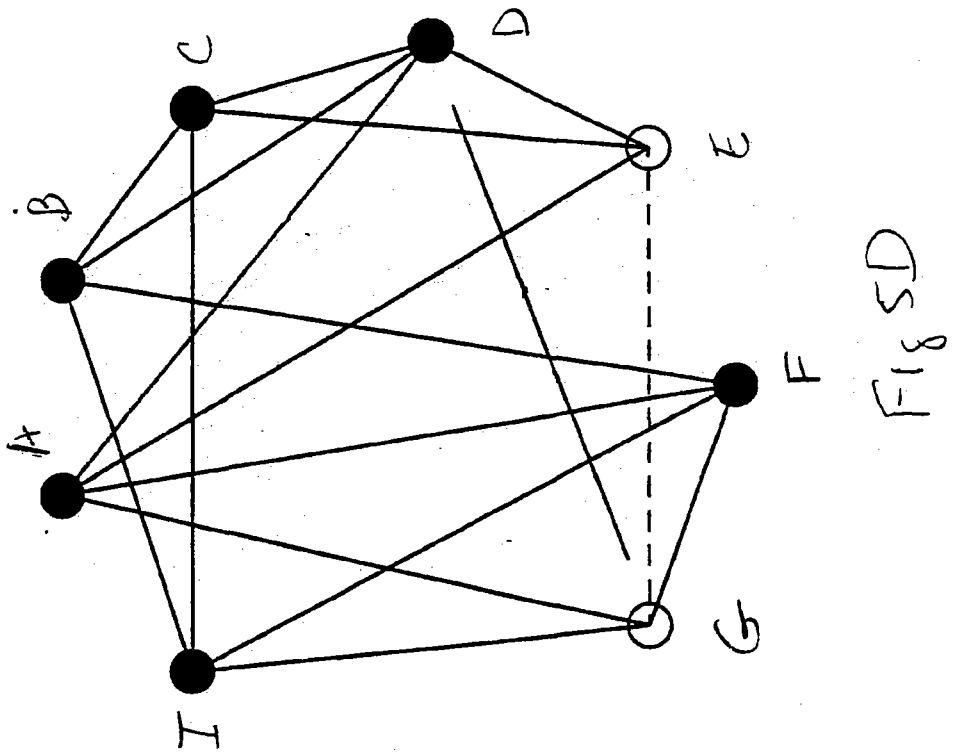


FIG 5D

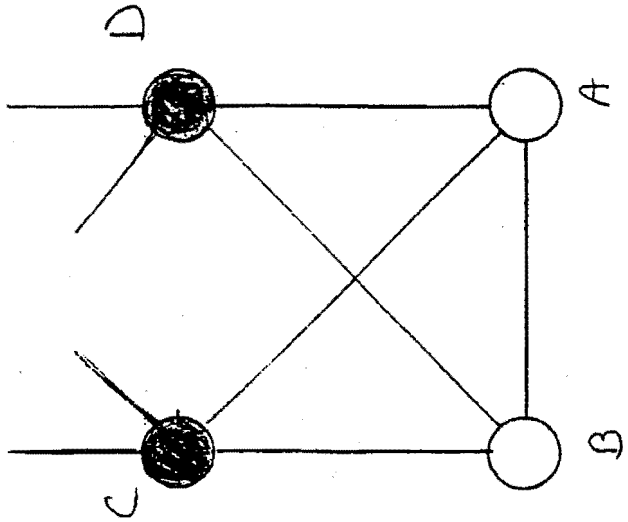


Fig 5F

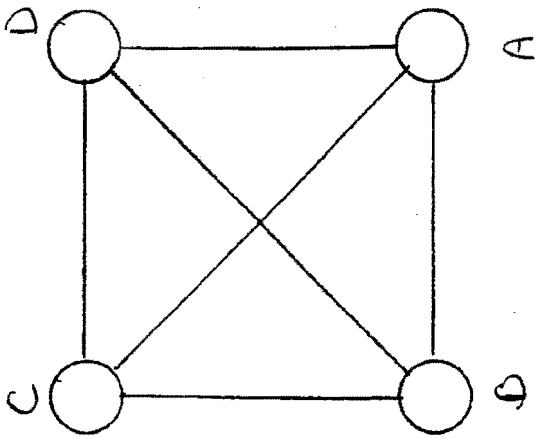


Fig 5E

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1077 of 1657

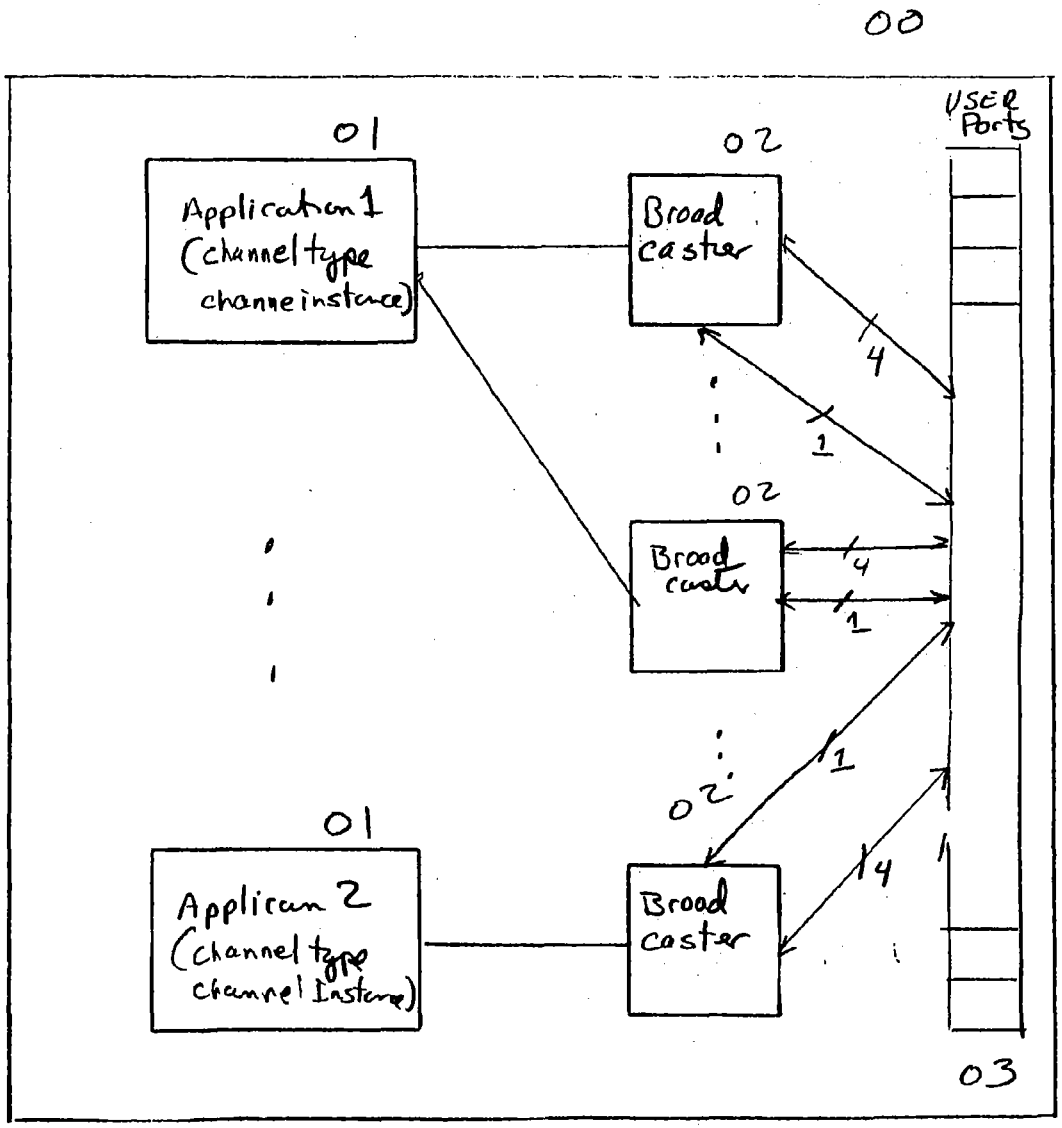


Fig 6

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1078 of 1657

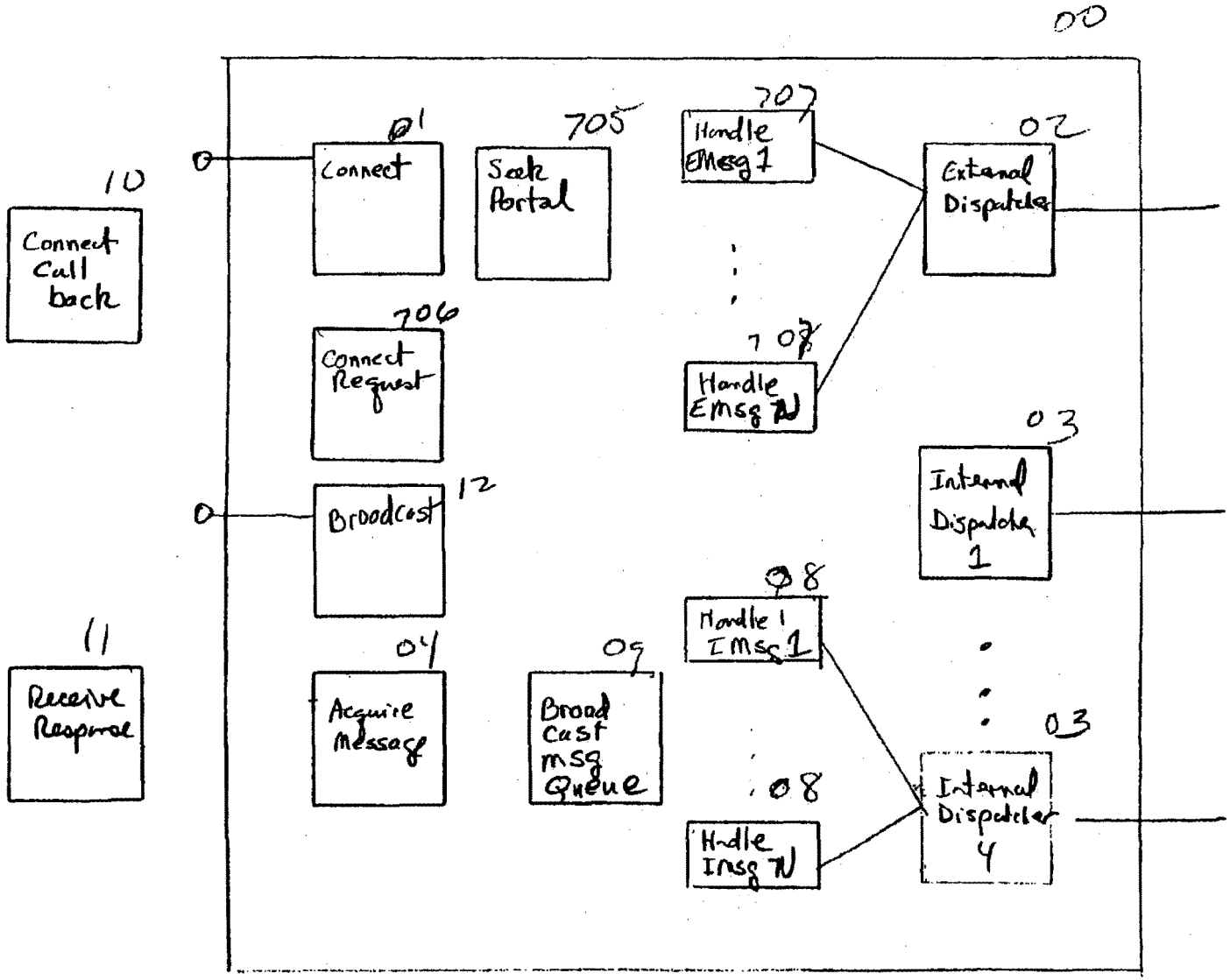


Figure 7

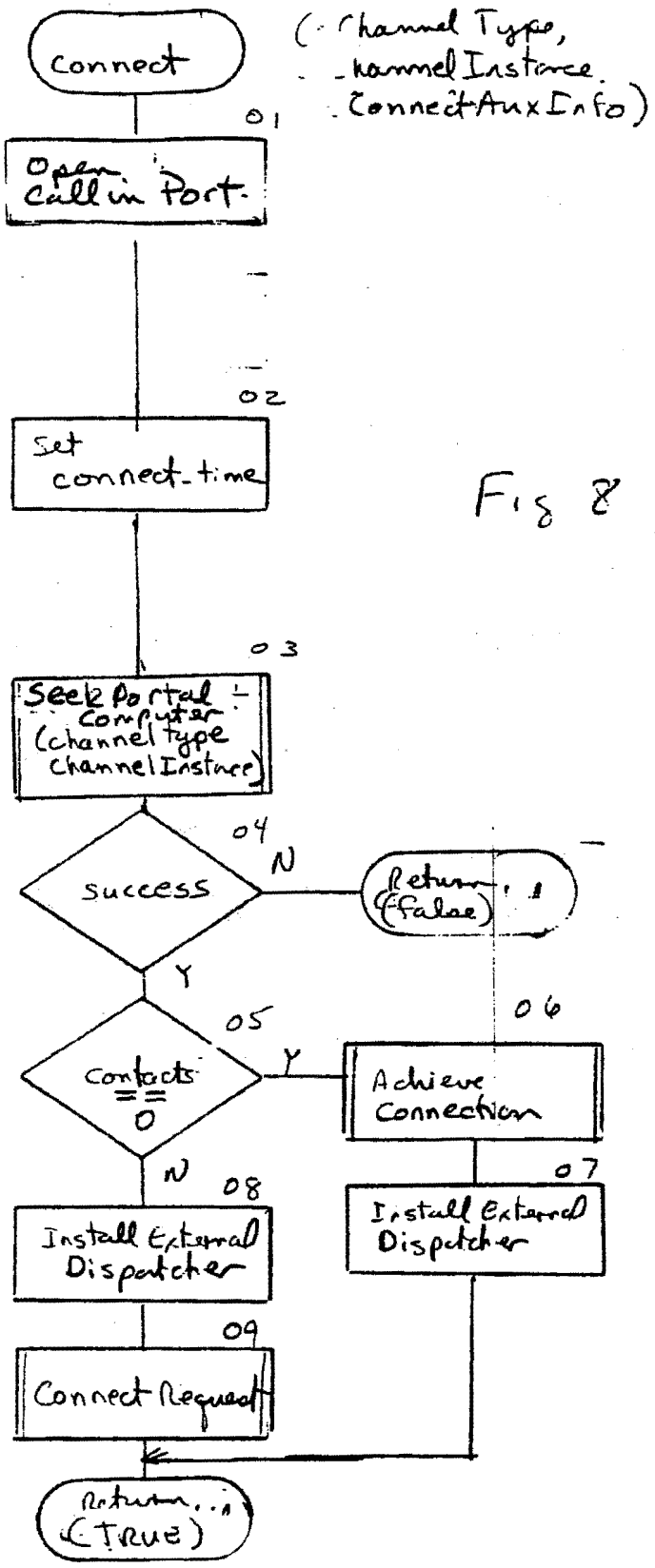

Connect
(channel Type,
channel Instance,
ConnectAuxInfo)
01
Open call in Port.
02
Set connect_time
03
Seek Portal Computer (channel type channel Instance)
04
success
N
Return (false)
Y
05
contacts == 0
Y
06
Achieve Connection
07
Install External Dispatcher
N
08
Install External Dispatcher
09
Connect Request
Return (TRUE)

Fig 8

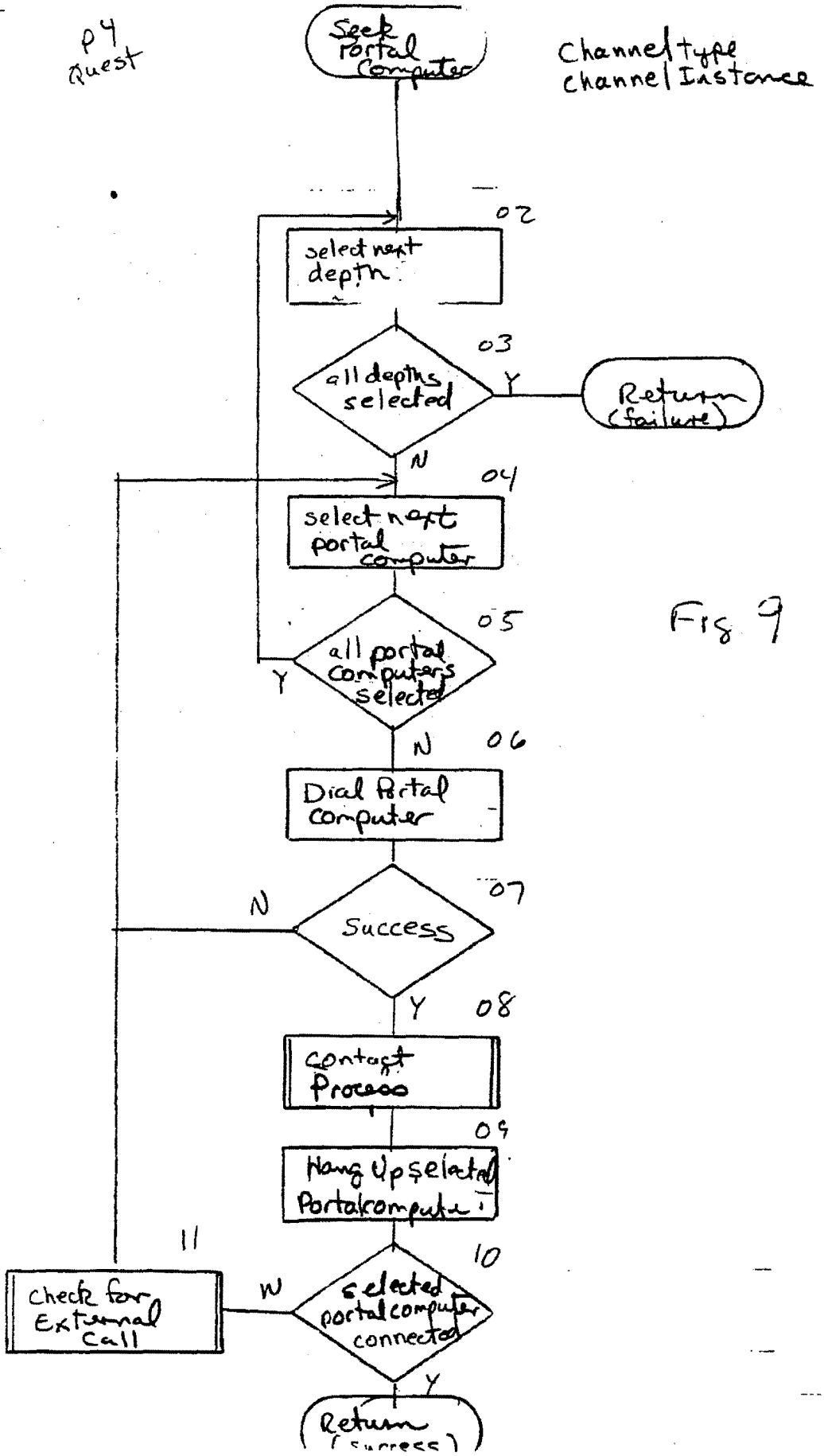

P4
Quest
Seek portal computer
Channel type
channel Instance
02
select next depth
03
all depths selected
Y
Return (failure)
N
04
select next portal computer
05
all portal computers selected
Y
N
06
Dial Portal computer
07
Success
N
Y
08
contact Process
09
Hang Up selected Portal computer
10
selected portal computer connected
N
11
check for External Call
Y
Return (success)

Fig 9

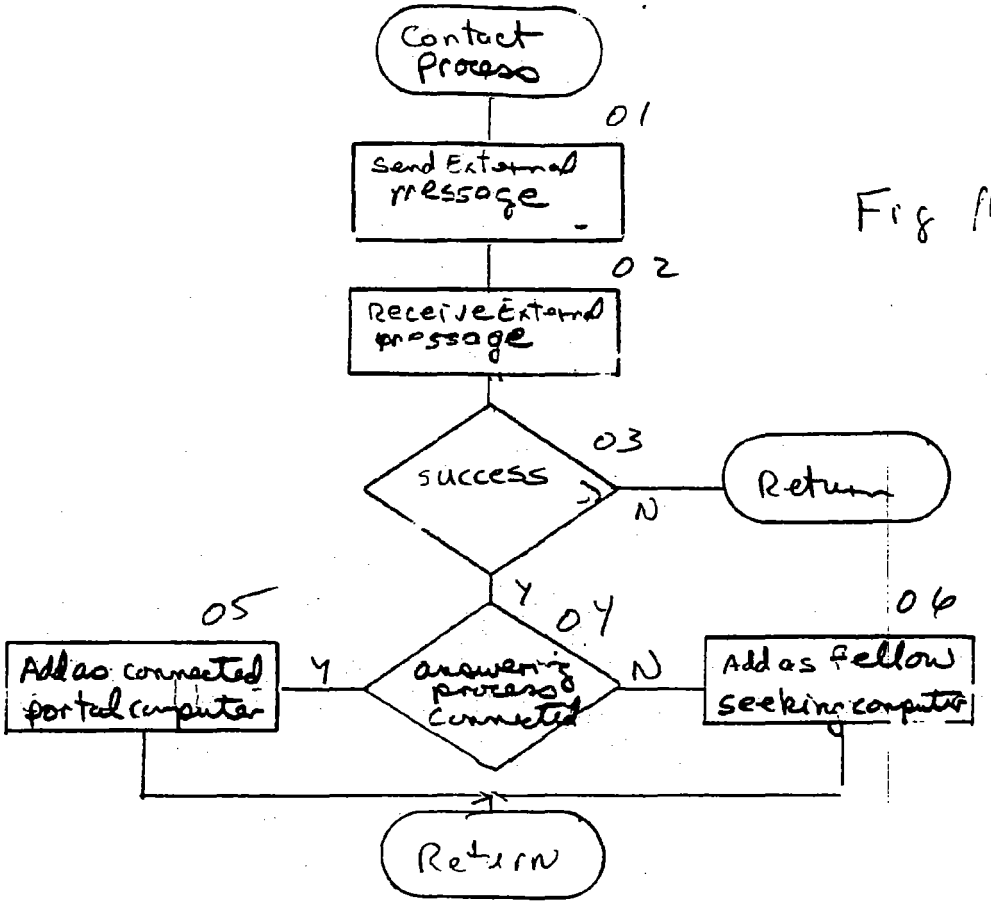
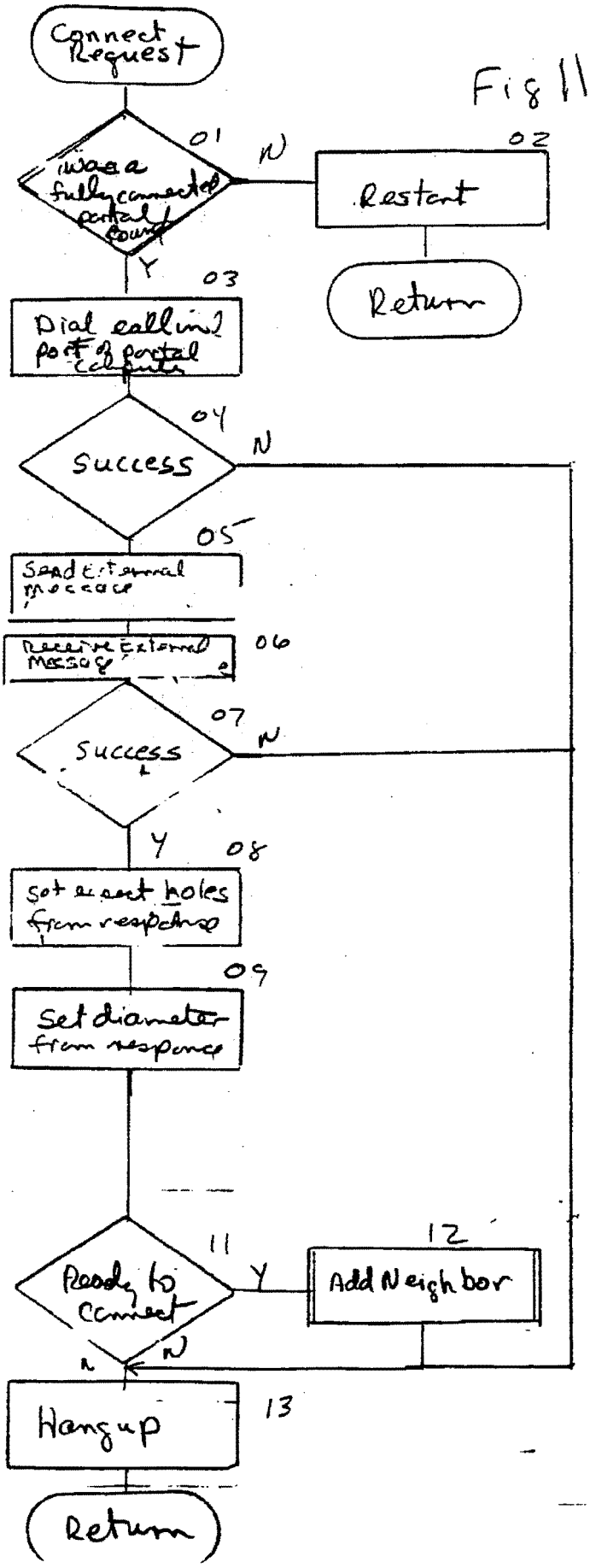

Contact Process
01
Send External message
02
Receive External message
success
03
N
Return
Y
04
05
Add as connected portal computer
Y
answering process connected
N
06
Add as fellow seeking computer
Return

Fig 10

p23

Fig 11


Connect Request
01
N
02
Restart
Return
Y
03
Dial calling port of portal
04
Success
N
05
Send External message
06
Receive External Message
07
Success
N
Y
08
Set exact holes from response
09
Set diameter from response
11
Ready to Connect
Y
12
Add Neighbor
N
N
Hangup
13
Return

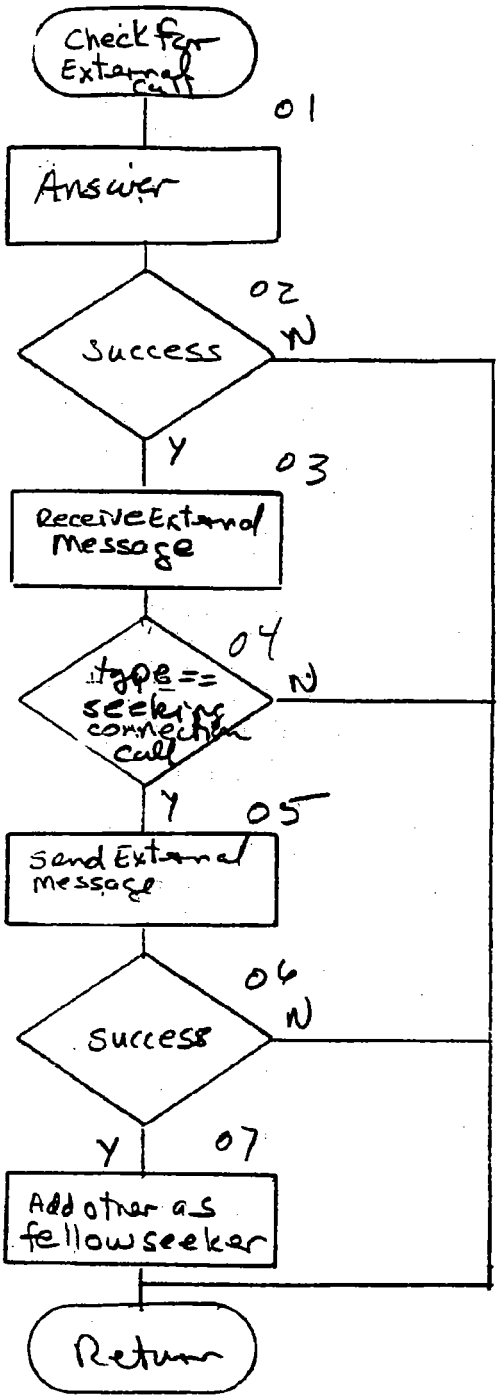

Check for External call
01
Answer
02
Success
N
Y
03
Receive External message
04
type == seeking connection call
N
Y
05
Send External message
06
Success
N
Y
07
Add other as fellow seeker
Return

Fig 12

Achieve Connection
22

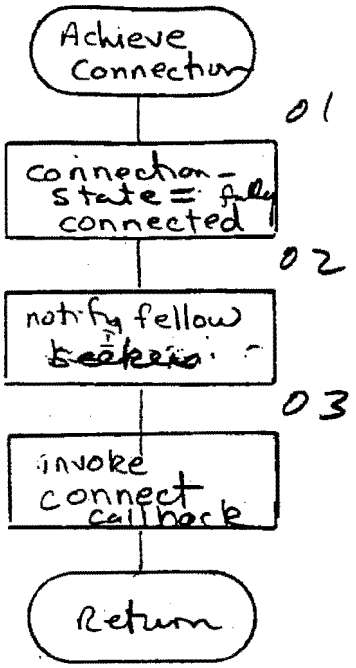
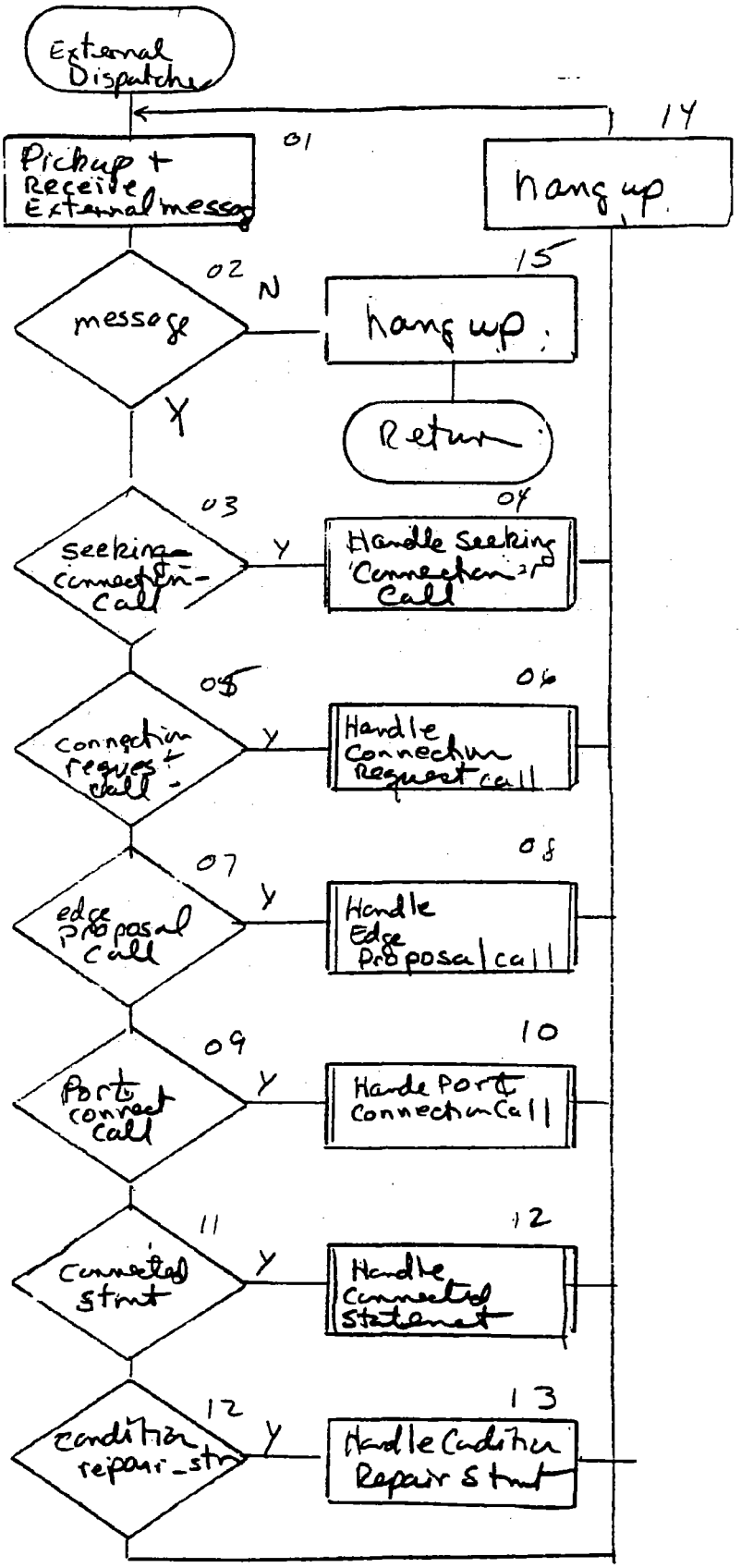


Fig 13

p7

Fig 14


External Dispatcher
01
Pickup + Receive External message
14
hang up
02
message
N
15
hang up
Y
Return
03
seeking connection call
Y
04
Handle Seeking Connection Call
05
connection request call
Y
06
Handle Connection Request call
07
edge proposal call
Y
08
Handle Edge Proposal call
09
Port connect call
Y
10
Handle Port Connection Call
11
connected stmt
Y
12
Handle Connected Statement
12
condition repair_stm
Y
13
Handle Condition Repair Stmt

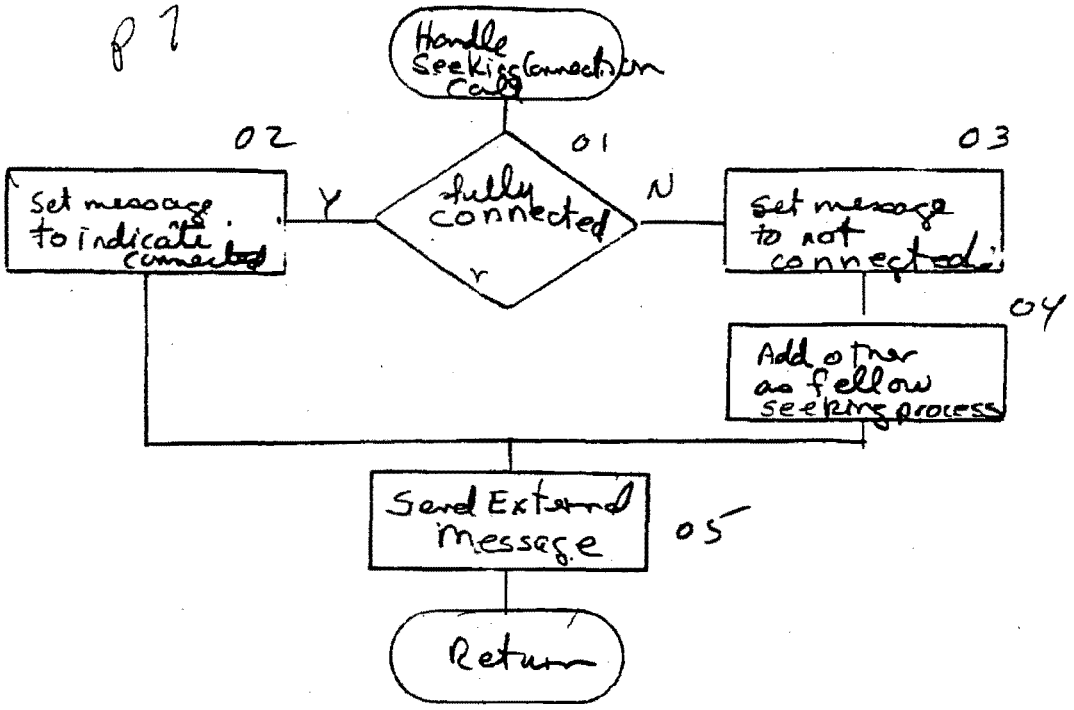
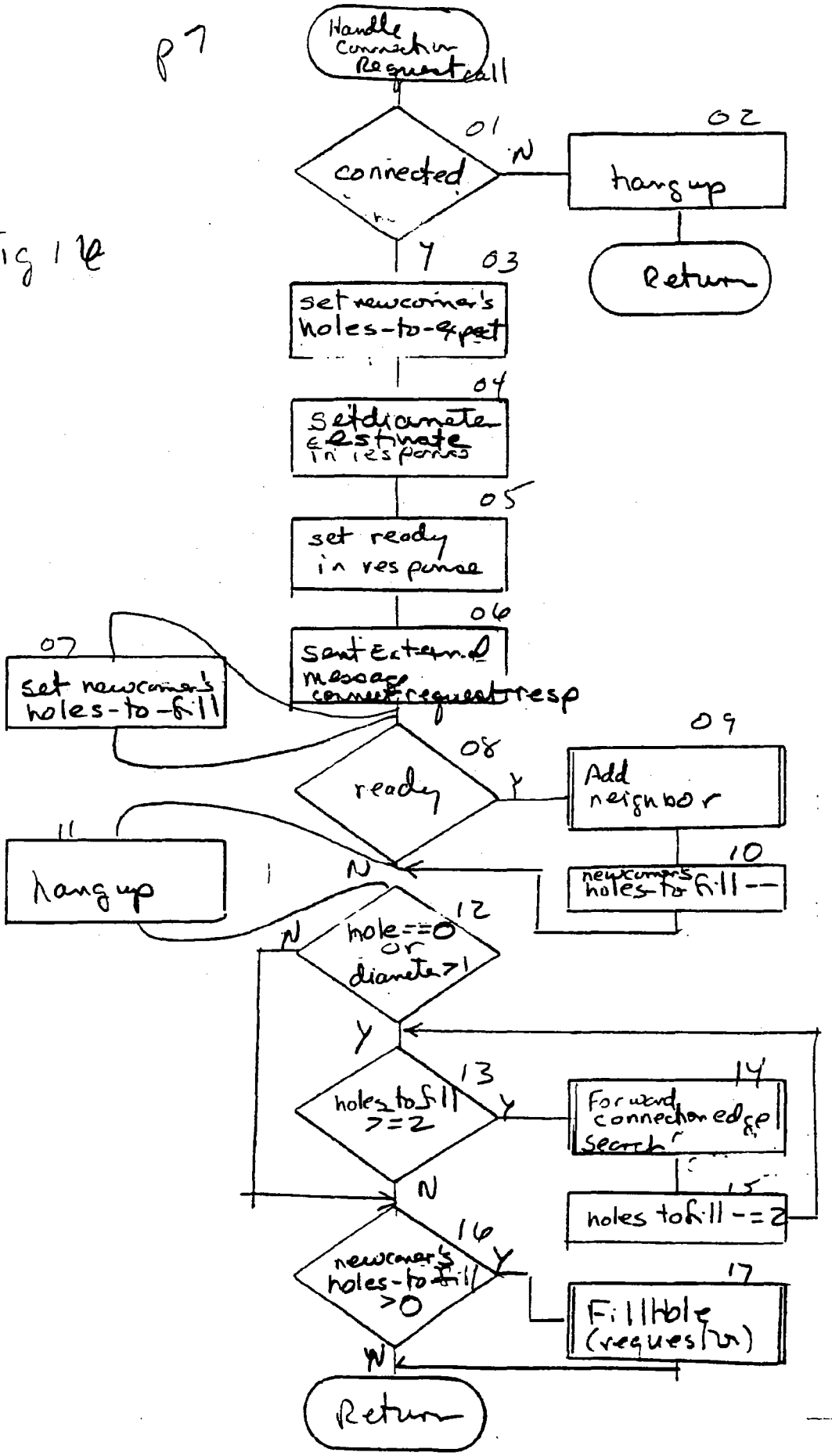

Handle Seeking Connection Call
01
02
03
fully connected
Y
N
Set message to indicate connected
Set message to not connected
04
Add other as fellow seeking process
Send Extend message
05
Return

Fig 15

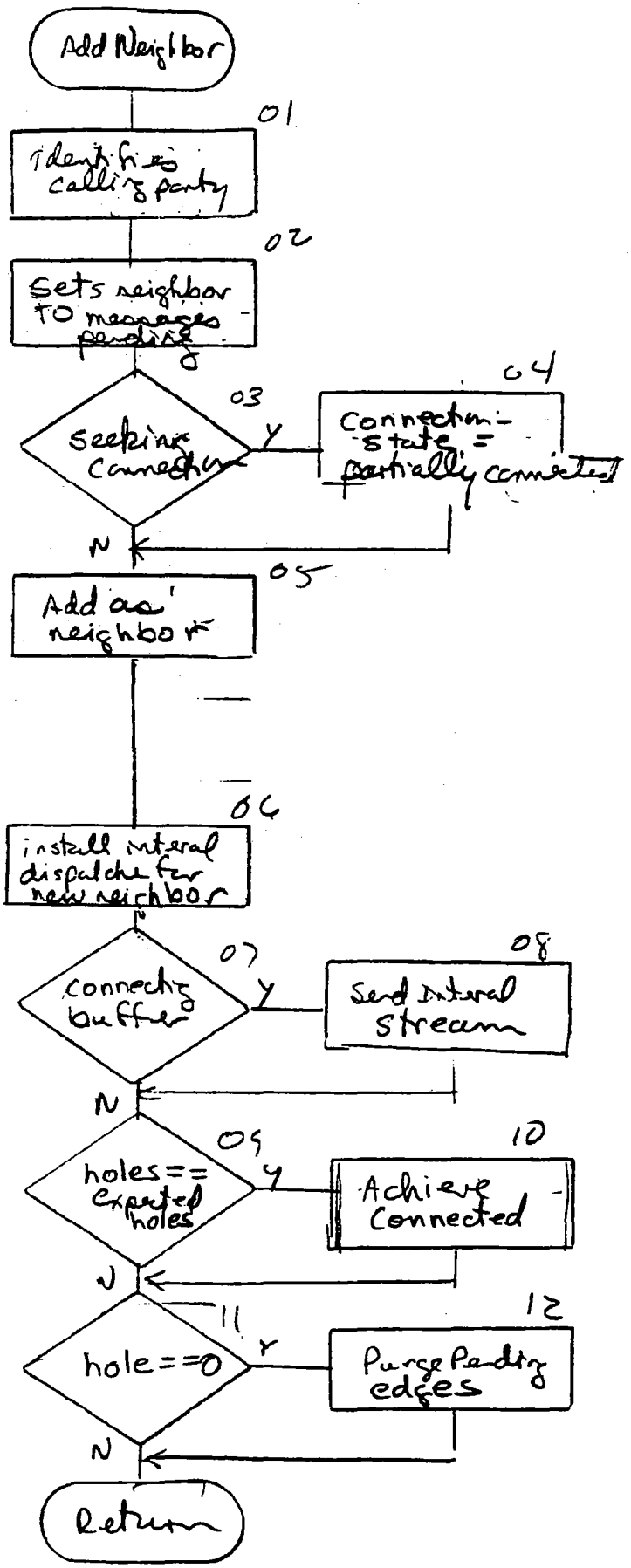
p7

Fig 1b


Handle Connection Request call
01
connected
N
02
hangup
Return
Y
03
set newcomer's holes-to-expect
04
set diameter estimate in response
05
set ready in response
06
sent External message connect request resp
07
set newcomer's holes-to-fill
08
ready
Y
09
Add neighbor
10
newcomer's holes-to fill --
11
hangup
N
12
hole==0 or diameter>1
N
Y
13
holes to fill >=2
Y
14
Forward connection edge search
15
holes to fill -=2
N
16
newcomer's holes-to-fill >0
Y
17
Fill Hole (requestor)
N
Return

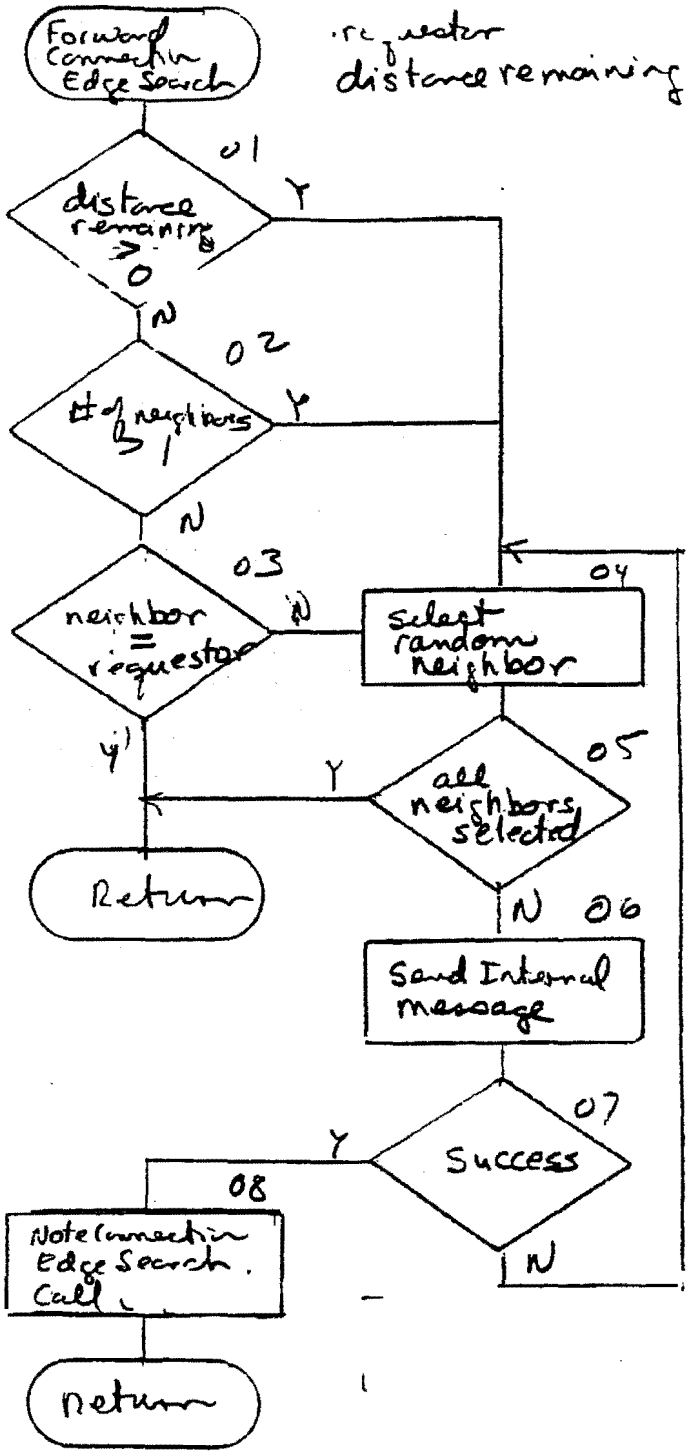
p9

Fig 17


Add Neighbor
01
Identifies calling party
02
Sets neighbor TO messages pending
03
Seeking Connection
Y
04
Connection-state = partially connected
N
05
Add as' neighbor
06
install interal dispatcher for new neighbor
07
connect buffer
Y
08
Send Interal Stream
N
09
holes == expected holes
Y
10
Achieve Connected
N
11
hole == 0
Y
12
Purge Pending edges
N
Return

15

Fig 18


Forward Connection Edge Search
requestor distance remaining
01
distance remaining > 0
Y
N
02
# of neighbors = 1
Y
N
03
neighbor = requestor
N
04
select random neighbor
Y
05
all neighbors selected
Y
N
Return
06
Send Internal message
07
Success
Y
N
08
Note Connection Edge Search Call
return

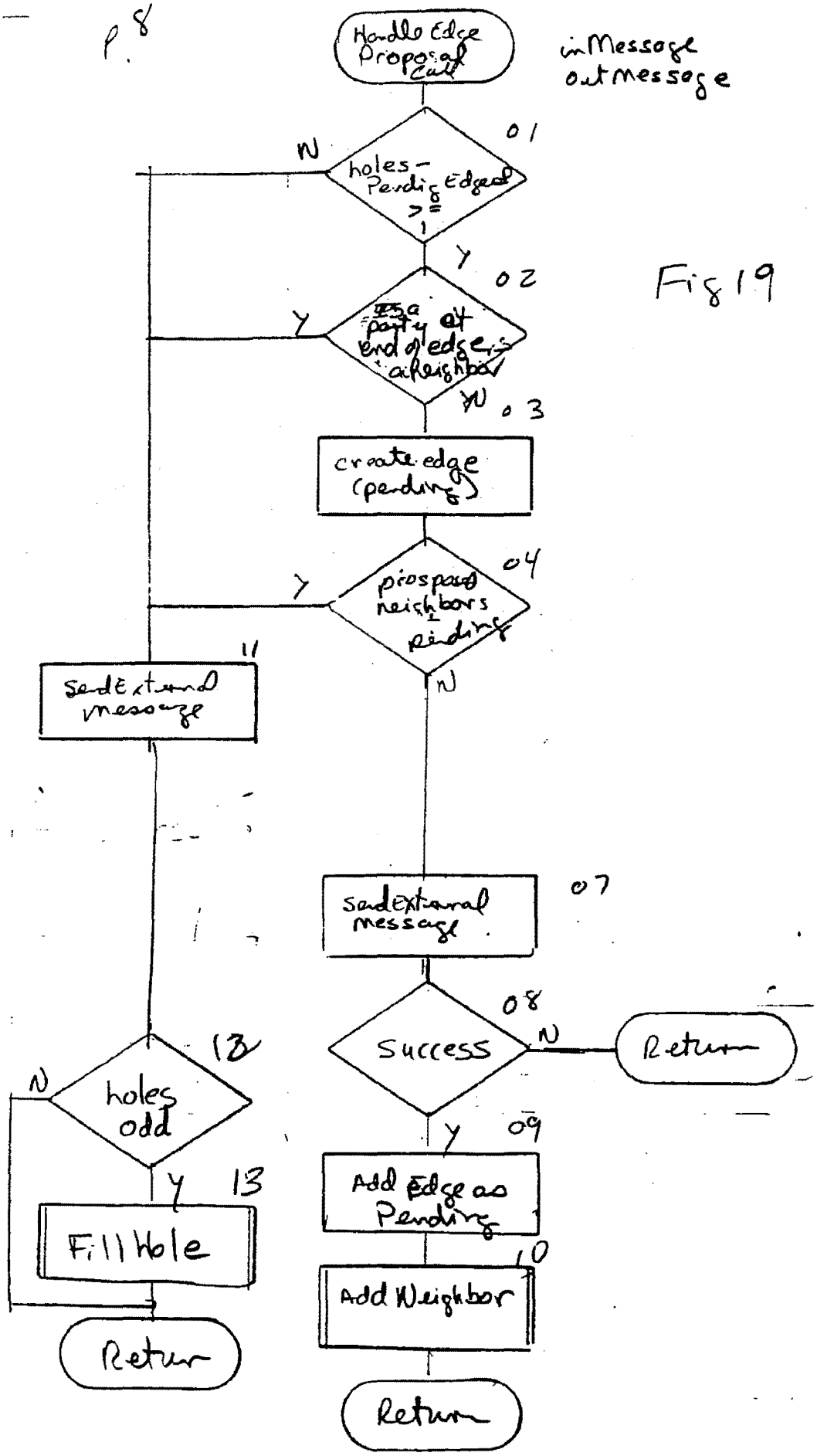

Handle Edge Proposal Exit
in Message
out message
01
holes - Pending Edges >= 1
N
Y
02
Is a party at end of edge a neighbor
Y
N
03
create edge (pending)
04
prospect neighbors pending
Y
N
11
Send Extend message
07
Send External message
08
Success
N
Return
Y
09
Add Edge as Pending
10
Add Neighbor
Return
12
holes odd
N
Y
13
Fill Hole
Return

Fig 19

Fig 20

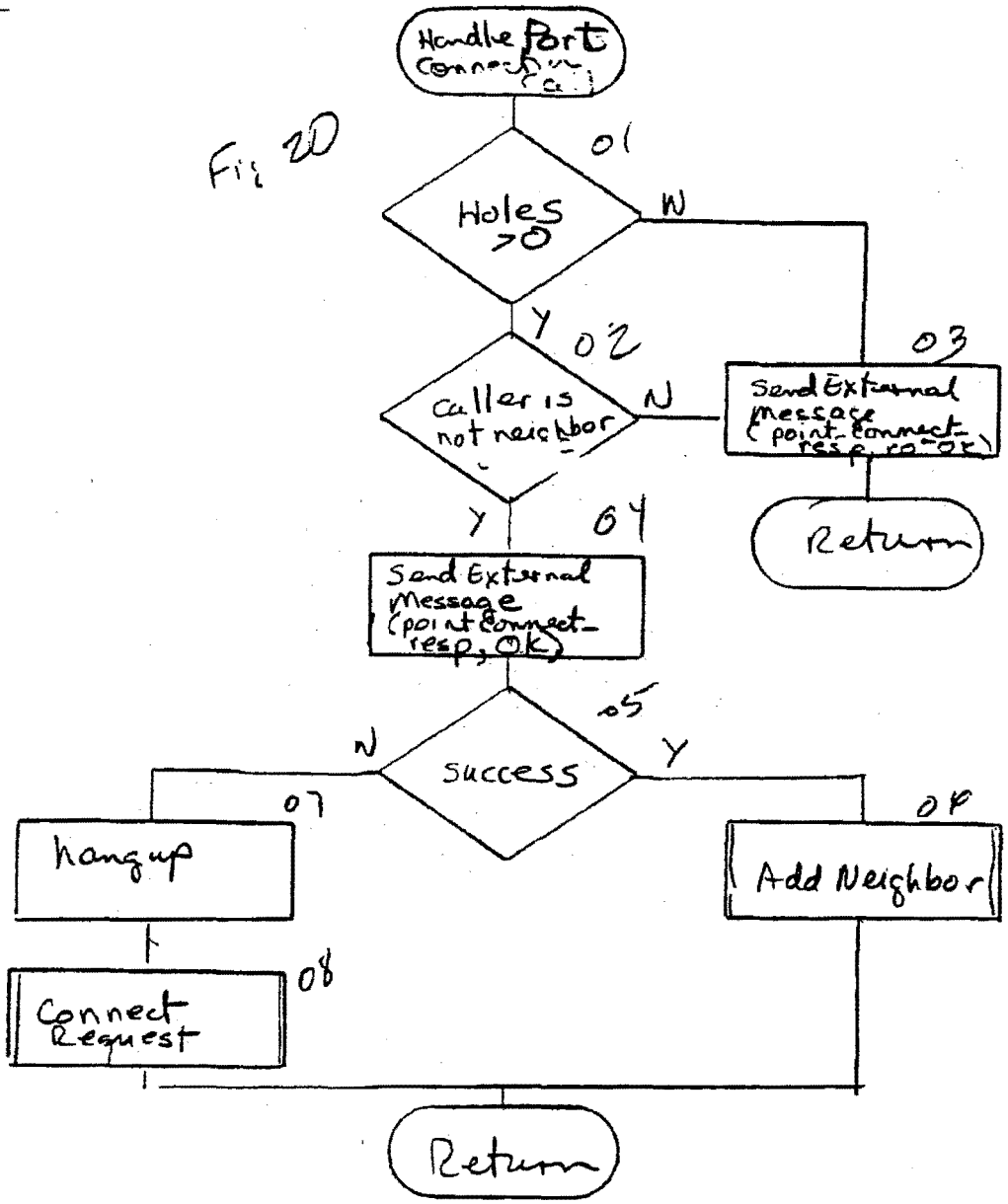
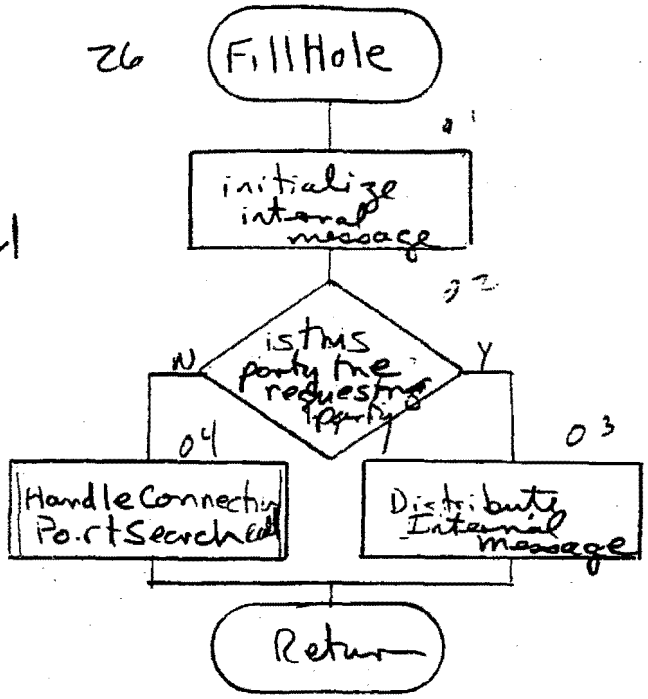

Handle Port Connect (...)
01
Holes >0
N
Y
02
Caller is not neighbor
N
Y
03
Send External Message (point_connect_resp, ...)
Return
04
Send External Message (point_connect_resp, OK)
05
Success
N
Y
07
hangup
08
Connect Request
0F
Add Neighbor
Return

Fig 21


26
FillHole
initialize internal message
is TMS party the requesting party
N
Y
04
03
Handle Connection PortSearched
Distribute Internal message
Return

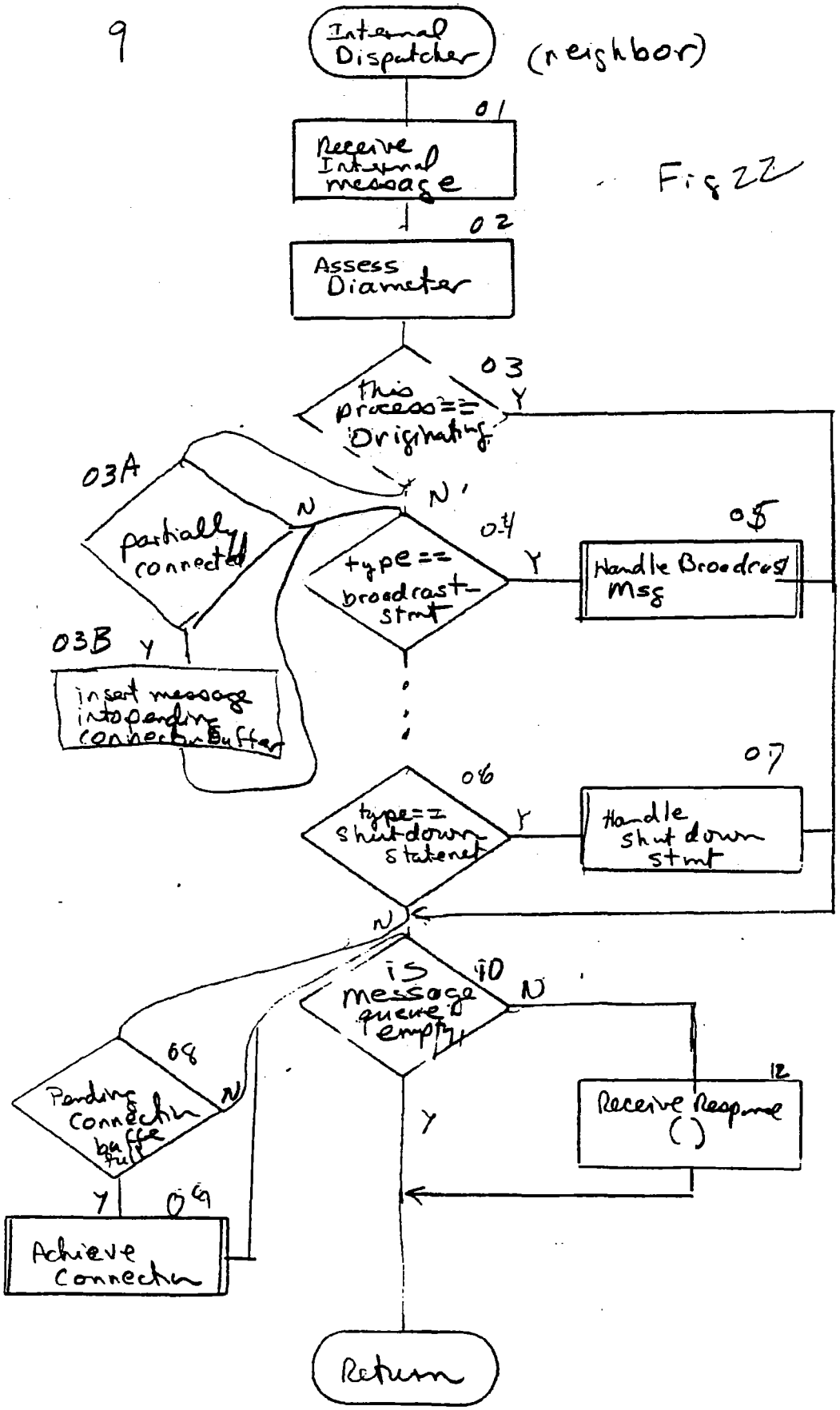

Internal Dispatcher
(neighbor)
01
Receive Internal message
02
Assess Diameter
03
this process == Originating
Y
N
03A
partially connected
N
03B
Y
insert message into pending connection buffer
04
type == broadcast-stmt
Y
05
Handle Broadcast Msg
06
type == shut-down statement
Y
07
Handle shut down stmt
N
10
is message queue empty
N
08
Pending Connection buffer
N
Y
09
Achieve Connection
Y
12
Receive Response ( )
Return

Fig 22

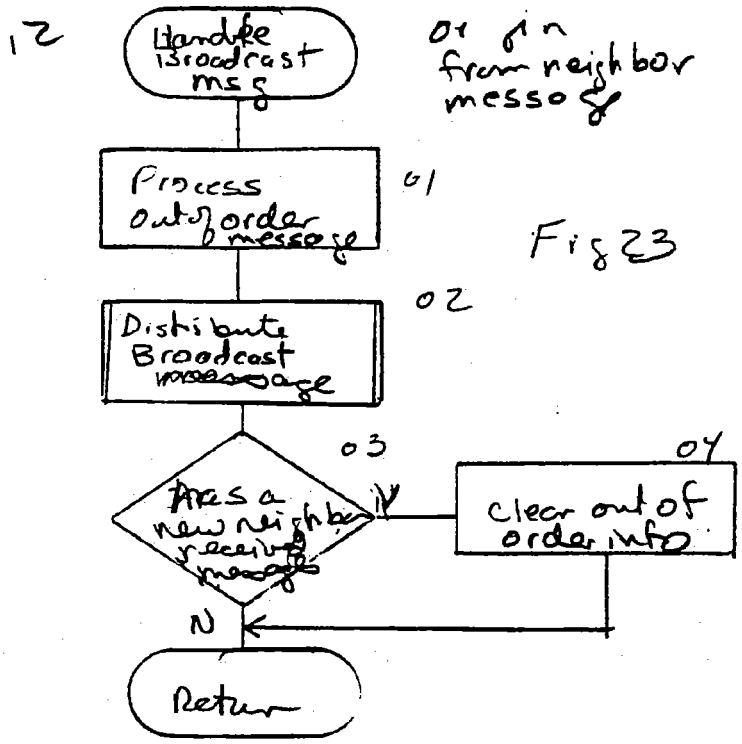

12
Handle Broadcast msg
or join from neighbor message
Process out of order message
01
Distribute Broadcast message
02
03
Has a new neighbor received message
Y
04
Clear out of order info
N
Return

Fig 23

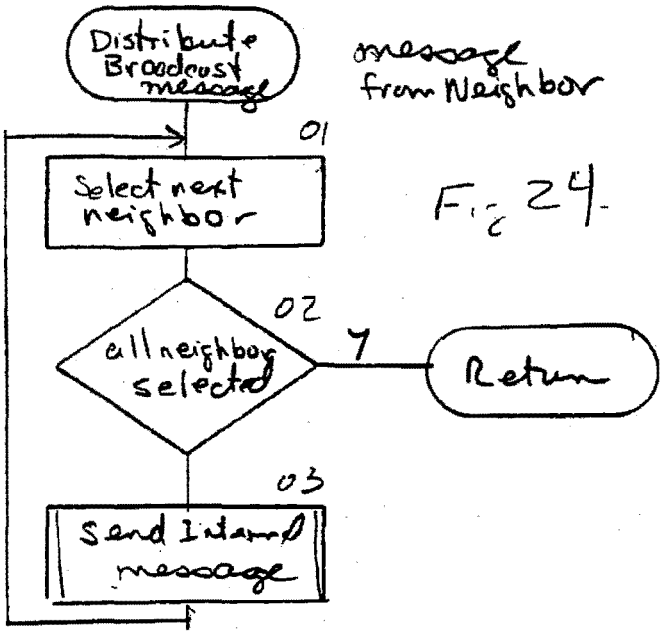

Distribute Broadcast message
message from Neighbor
01
Select next neighbor
02
all neighbors selected
Y
Return
03
Send Intermd message

Fig 24.

13

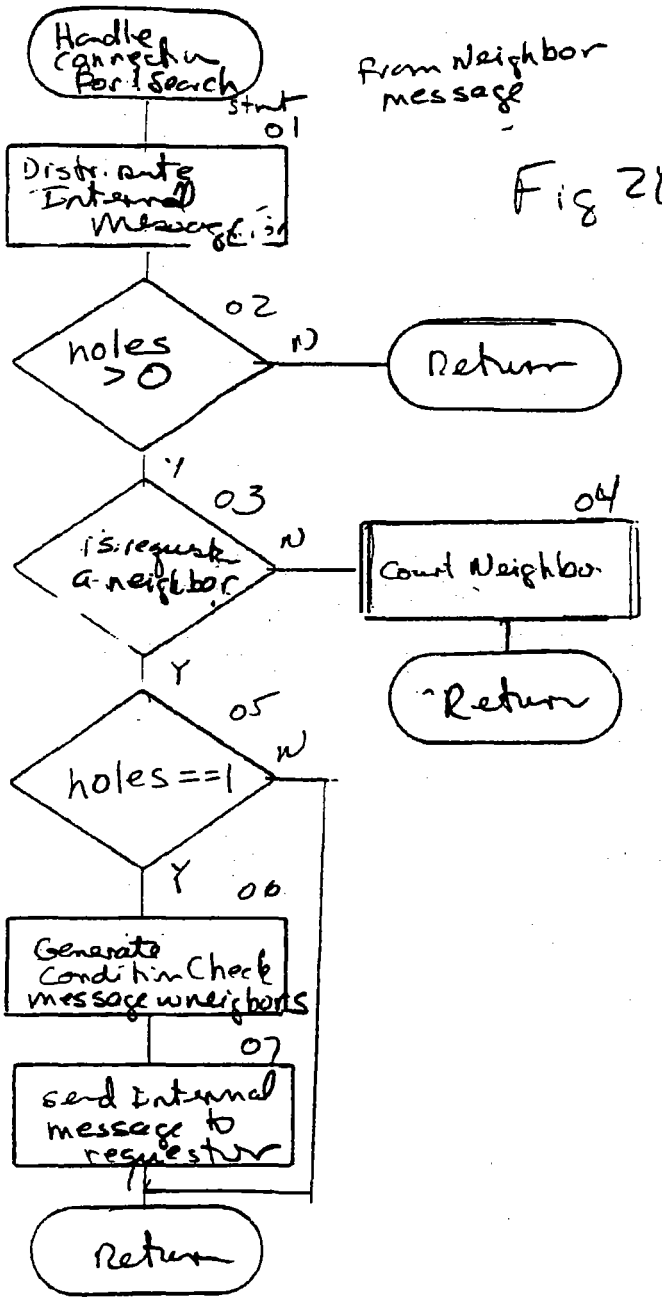

Handle Connection For Search
Start
01
From Neighbor message
Distribute Internal Message
02
holes >0
N
Return
Y
03
is request a neighbor
N
04
Count Neighbor
Return
Y
05
holes == 1
N
Y
06
Generate Condition Check message w neighbors
07
send Internal message to requestor
Return

Fig 26

21

21

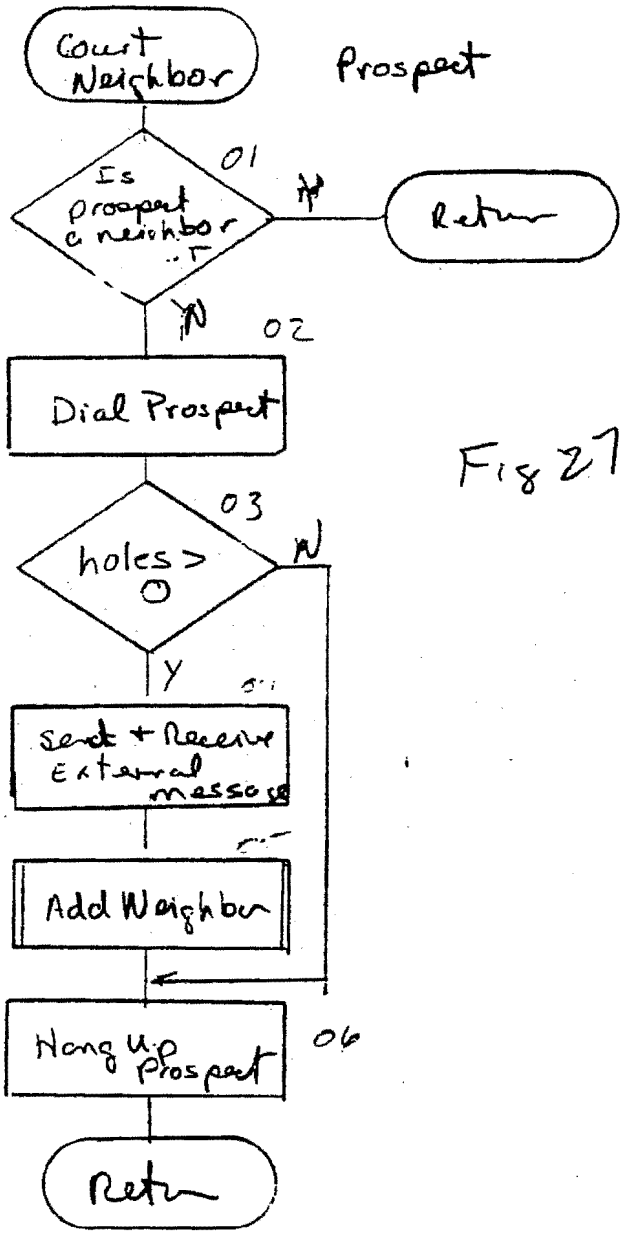

Court Neighbor
Prospect
Is prospect a neighbor
01
Y
Return
N
02
Dial Prospect
03
holes > 0
N
Y
Send + Receive External messages
Add Neighbor
Hang Up Prospect
06
Return

Fig 27

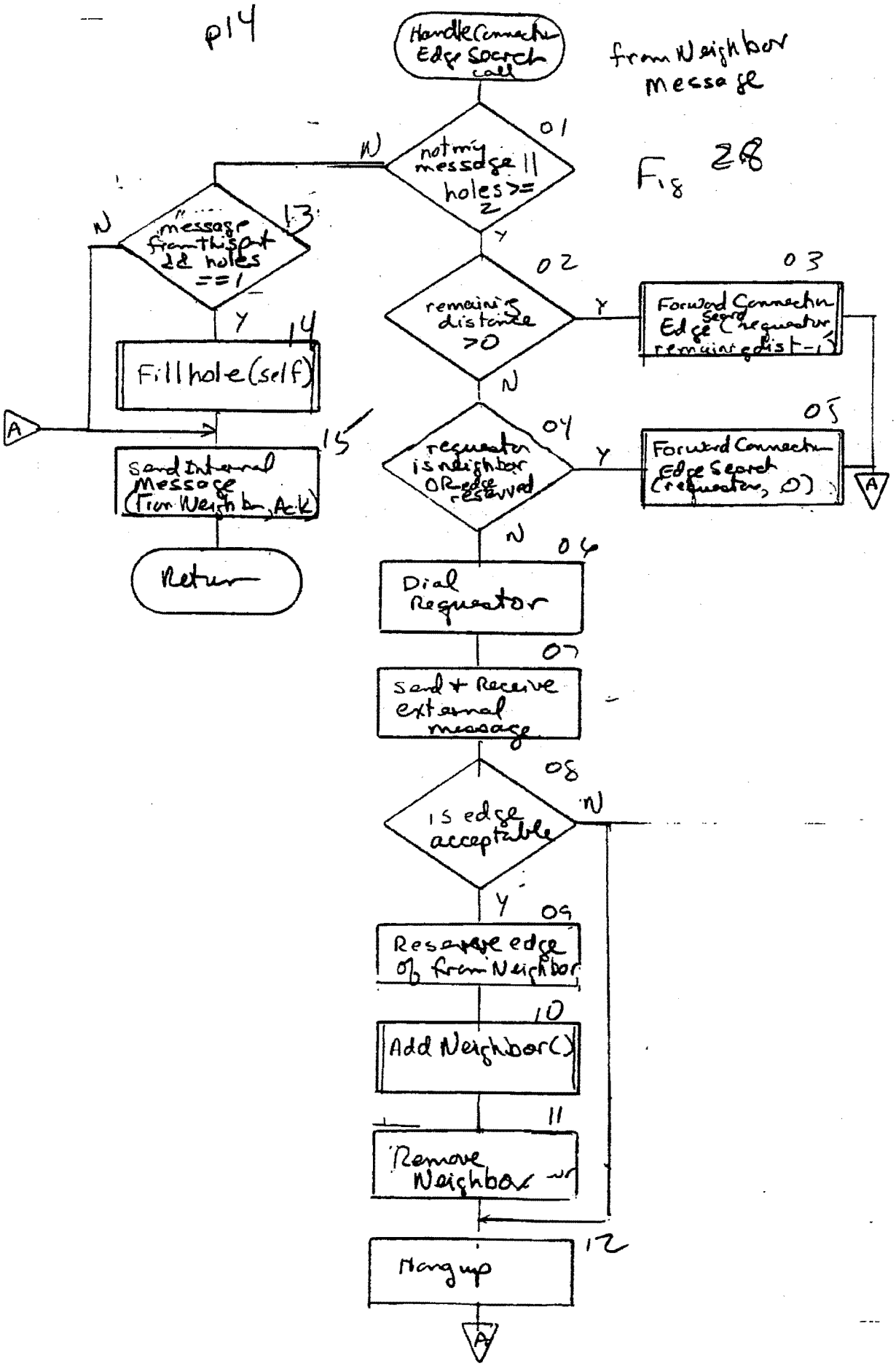

Handle Connection Edge Search call
from Neighbor message
01
not my message || holes >= 2
N
Y
13
message from this spot && holes == 1
N
Y
14
Fill hole (self)
A
15
Send Internal Message (From Neighbor, Ack)
Return
02
remaining distance > 0
Y
03
Forward Connection Edge Search (requestor, remaining dist - 1)
N
04
requestor is neighbor OR edge reserved
Y
05
Forward Connection Edge Search (requestor, 0)
A
N
06
Dial Requestor
07
Send + Receive external message
08
is edge acceptable
N
Y
09
Reserve edge of from Neighbor
10
Add Neighbor()
11
Remove Neighbor
12
Hang up
A

Fig 28

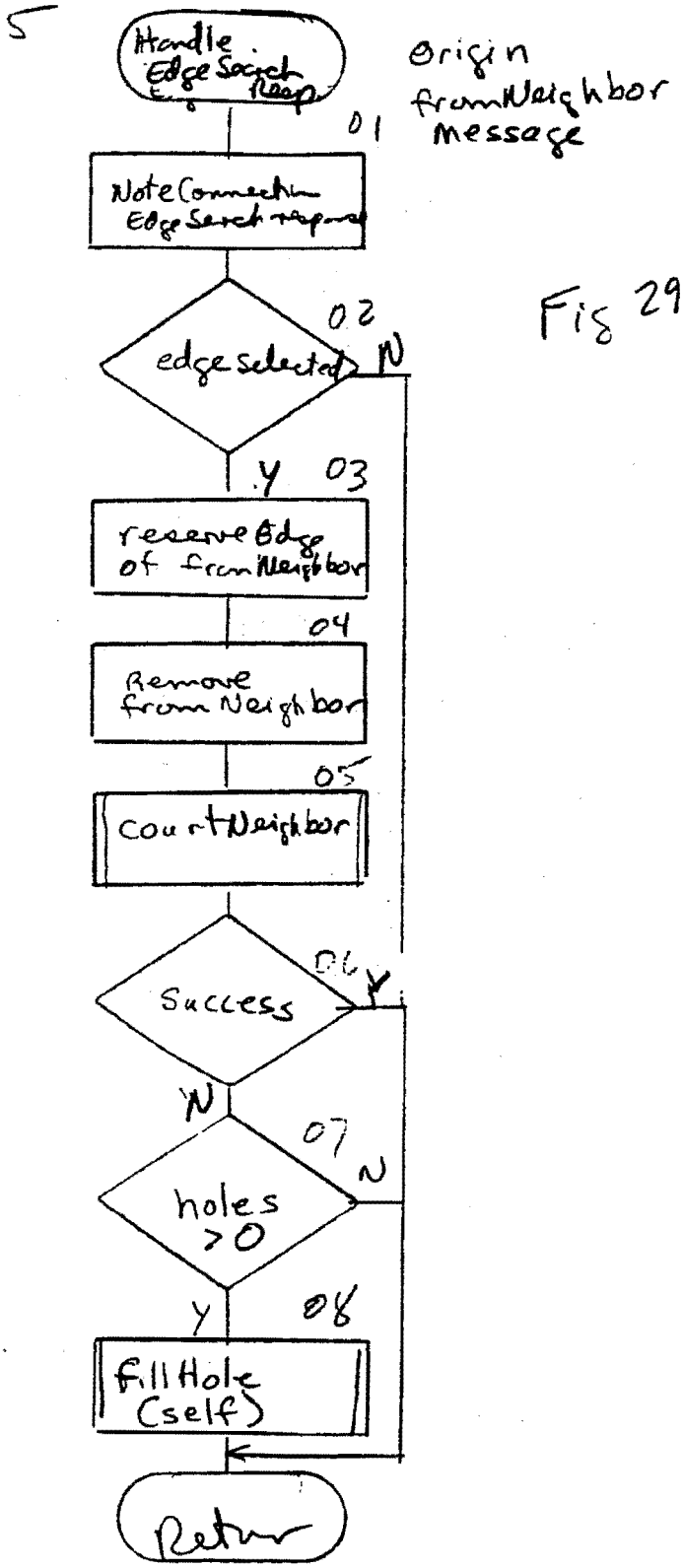

5
Handle Edge Search Resp.
Origin from Neighbor message
01
Note Connection Edge Search response
02
edge selected
N
Y
03
reserve Edge of from Neighbor
04
Remove from Neighbor
05
Court Neighbor
06
Success
Y
N
07
holes >0
N
Y
08
Fill Hole (self)
Return

Fig 29

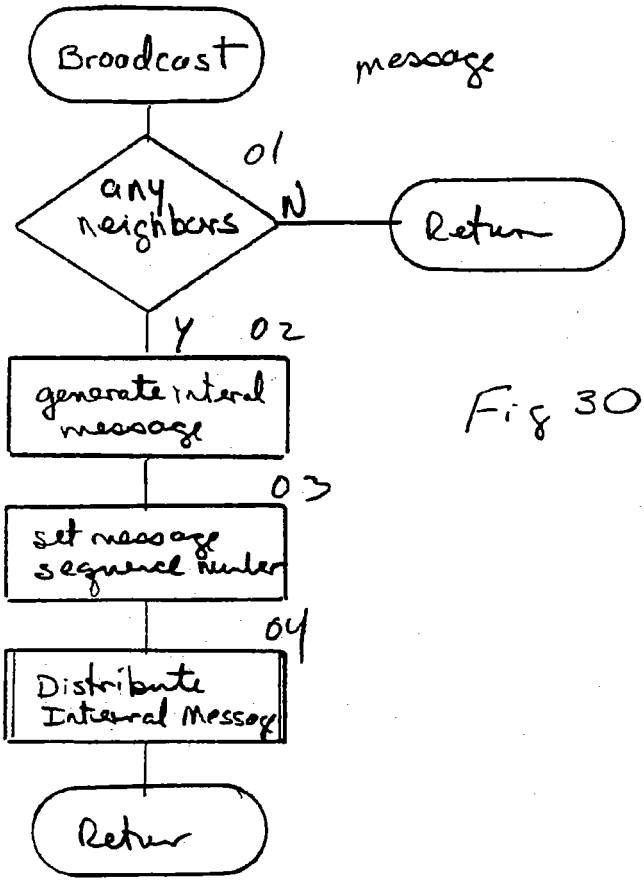

Broadcast message
01
any neighbors
N
Return
Y
02
generate internal message
03
set message sequence number
04
Distribute Internal Message
Return

Fig 30

3

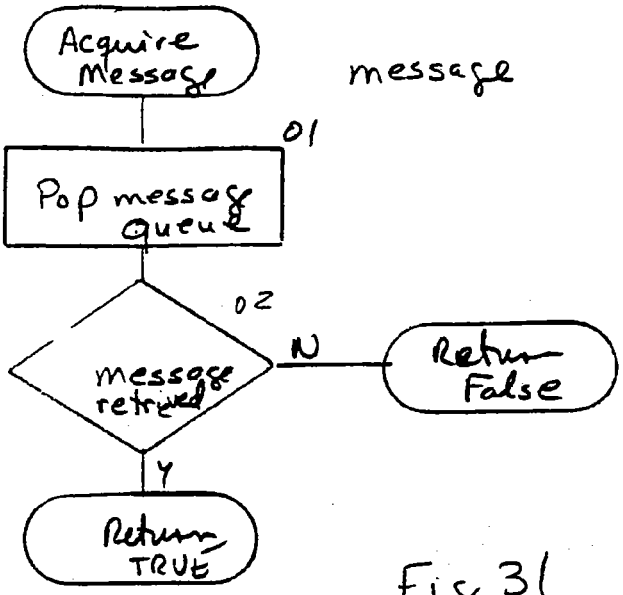

Acquire Message
message
01
Pop message queue
02
message retrieved
N
Return False
Y
Return TRUE

Fig 31

10

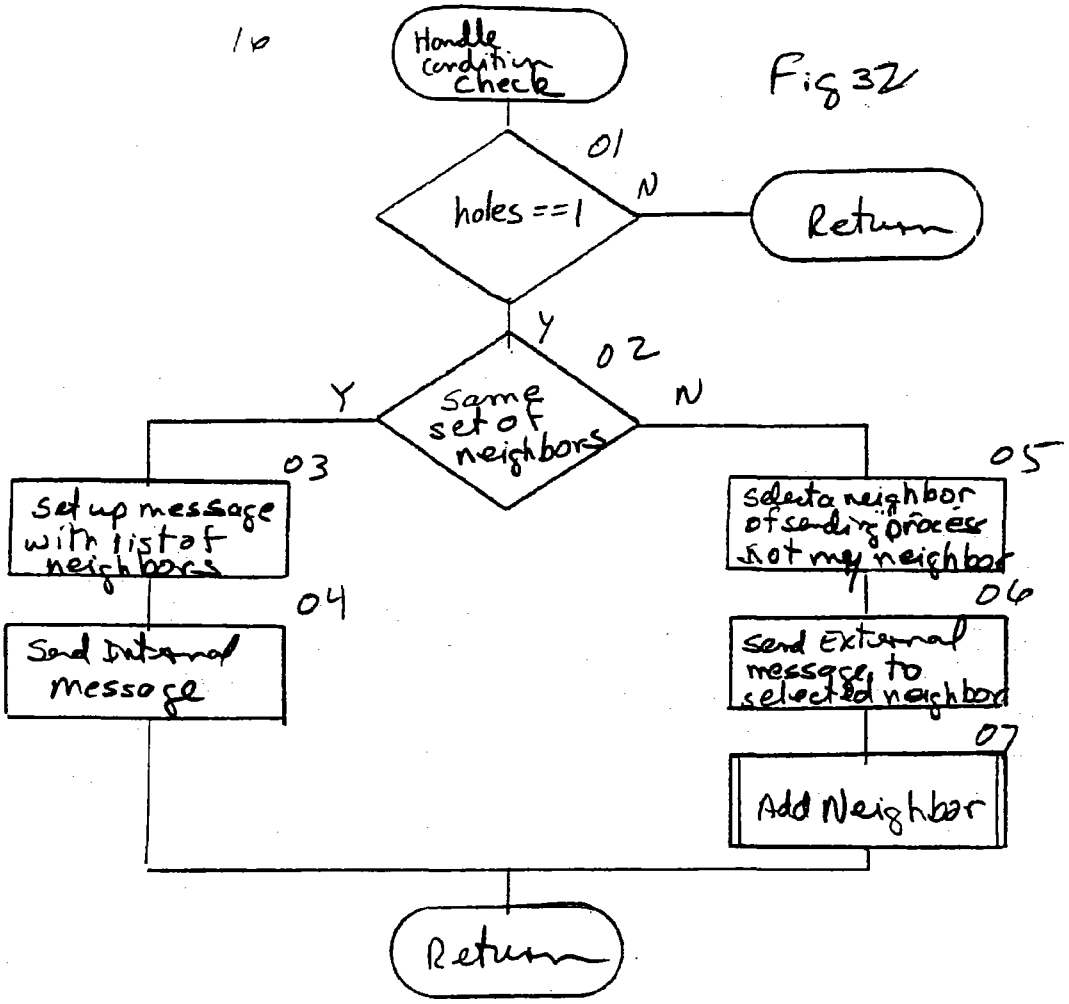

Handle condition check
01
holes == 1
N
Return
Y
02
Same set of neighbors
Y
N
03
Set up message with list of neighbors
04
Send Internal message
05
Select a neighbor of sending process not my neighbor
06
Send External message to selected neighbor
07
Add Neighbor
Return

Fig 32

23

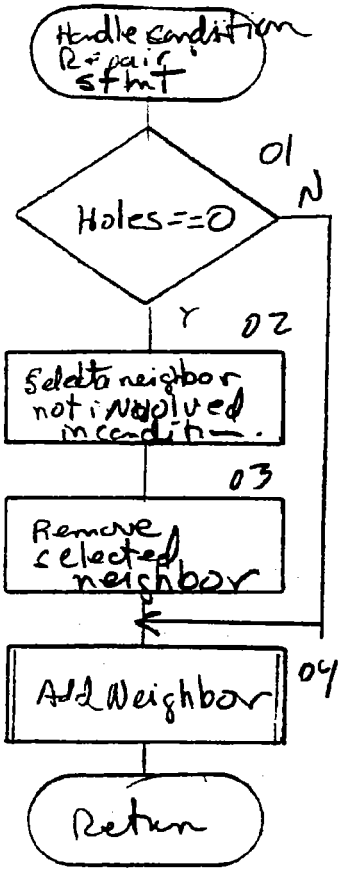
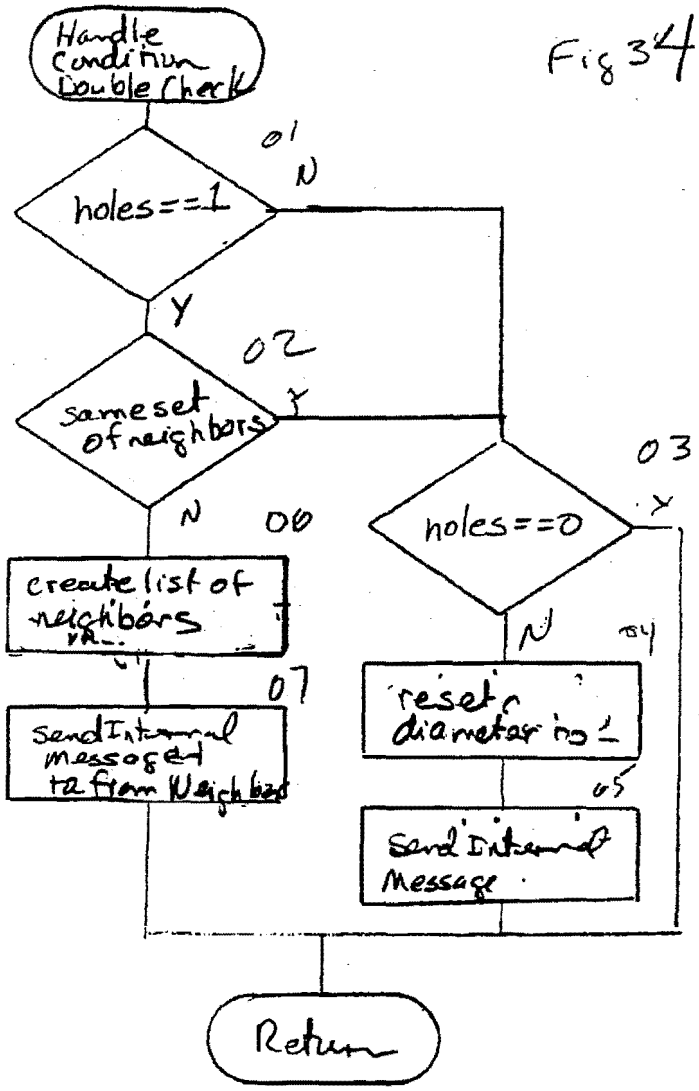

Handle condition
Repair
stmt
01
Holes==0
N
Y
02
Select a neighbor not involved in condition.
03
Remove selected neighbor
Add Neighbor
04
Return

Fig 33

17

Fig 34


Handle Condition Double Check
01
holes==1
N
Y
02
same set of neighbors
Y
N
06
03
holes==0
Y
N
04
create list of neighbors
07
reset diameter to 1
05
Send Internal Message
Send Internal messages to/from Neighbors
Return

AN INFORMATION DELIVERY SERVICE

CROSS-REFERENCE TO RELATED APPLICATIONS

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

TECHNICAL FIELD

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

BACKGROUND

There are a wide variety of computer network communications techniques such as point-to-point network protocols, client/server middleware, multicasting network

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1105 of 1657

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

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

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

The multicasting network protocols allow the sending of broadcast messages to multiple recipients of a network. The current implementations of such multicasting network

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1106 of 1657

protocols tend to place an unacceptable overhead on the underlying network. For example, UDP multicasting would swamp the Internet when trying to locate all possible participants. IP multicasting has other problems that include needing special-purpose infrastructure (*e.g.*, routers) to support the sharing of information efficiently.

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

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

BRIEF DESCRIPTION OF THE DRAWINGS

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

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

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

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

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1107 of 1657

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

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

Figure 5C illustrates the neighbors with empty ports condition.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1108 of 1657

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1109 of 1657

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

DETAILED DESCRIPTION

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1110 of 1657

divide the graph into disjoint sub-graphs, that is two separate broadcast channels. This property is referred to as being 4-connected. Thus, the graph is both 4-regular and 4-connected.

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

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

Composing the Graph

To connect to the broadcast channel, the computer seeking the connection first locates a computer that is currently fully connected to the broadcast channel and then

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1111 of 1657

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

Since the broadcast channel is a 4-regular graph, each of the identified computers is already connected to four computers. Thus, some connections between computers need to be broken so that the seeking computer can connect to four computers. In one embodiment, the broadcast technique identifies two pairs of computers that are currently connected to each other. Each of these pairs of computers breaks the connection between them, and then each of the four computers (two from each pair) connects to the seeking computer. Figures 3A and 3B illustrate the process of a new computer Z connecting to the broadcast channel. Figure 3A illustrates the broadcast channel before computer Z is connected. The pairs of computers B and E and computers C and D are the two pairs that are identified as the neighbors for the new computer Z. The connections between each of these pairs is broken, and a connection between computer Z and each of computers B, C, D, and E

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1112 of 1657

is established as indicated by Figure 3B. The process of breaking the connection between two neighbors and reconnecting each of the former neighbors to another computer is referred to as "edge pinning" as the edge between two nodes may be considered to be stretched and pinned to a new node.

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1113 of 1657

seeking computer actually communicates through that transfer-to port, which is the external port. The call is transferred so that other computers can place calls to that computer via the call-in port. The seeking computer then communicates via that external port to request the portal computer to assist in connecting the seeking computer to the broadcast channel. The seeking computer could identify the call-in port number of a portal computer by successively dialing each port in port number order. As discussed below in detail, the broadcast technique uses a hashing algorithm to select the port number order, which may result in improved performance.

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1114 of 1657

Broadcasting Through the Graph

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

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

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

When the broadcast channel is in a steady state (*i.e.*, no computers connecting or disconnecting from the broadcast channel), out-of-order messages are not a problem because each computer will eventually receive both messages and can queue messages until all earlier ordered messages are received. If, however, the broadcast channel is not in a

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1115 of 1657

steady state, then problems can occur. In particular, a computer may connect to the broadcast channel after the second message has already been received and forwarded on by its new neighbors. When a new neighbor eventually receives the first message, it sends the message to the newly connected computer. Thus, the newly connected computer will receive the first message, but will not receive the second message. If the newly connected computer needs to process the messages in order, it would wait indefinitely for the second message.

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1116 of 1657

same originating computer through another neighbor. If the second set of messages contains a message that is ordered earlier than the messages of the first set received, then the newly connected computer may ignore that earlier ordered message if the computer already processed those later ordered messages.

Decomposing the Graph

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1117 of 1657

computer that is also short a connection receives the connection request, it communicates with the requesting computer through its external port to establish a connection between the two computers. Figure 5B illustrates the disconnecting of a computer from the broadcast channel in an unplanned manner. In this illustration, computer H has disconnected in an unplanned manner. When each of its neighbors, computers A, E, F, and I, recognizes the disconnection, each neighbor broadcasts a port connection request indicating that it needs to fill an empty port. As shown by the dashed lines, computers F and I and computers A and E respond to each other's requests and establish a connection.

It is possible that a planned or unplanned disconnection may result in two neighbors each having an empty internal port. In such a case, since they are neighbors, they are already connected and cannot fill their empty ports by connecting to each other. Such a condition is referred to as the "neighbors with empty ports" condition. Each neighbor broadcasts a port connection request when it detects that it has an empty port as described above. When a neighbor receives the port connection request from the other neighbor, it will recognize the condition that its neighbor also has an empty port. Such a condition may also occur when the broadcast channel is in the small regime. The condition can only be corrected when in the large regime. When in the small regime, each computer will have less than four neighbors. To detect this condition in the large regime, which would be a problem if not repaired, the first neighbor to receive the port connection request recognizes the condition and sends a condition check message to the other neighbor. The condition check message includes a list of the neighbors of the sending computer. When the receiving computer receives the list, it compares the list to its own list of neighbors. If the lists are different, then this condition has occurred in the large regime and repair is needed. To repair this condition, the receiving computer will send a condition repair request to one of the neighbors of the sending computer which is not already a neighbor of the receiving computer. When the computer receives the condition repair request, it disconnects from one of its neighbors (other than the neighbor that is involved with the condition) and connects to the computer that sent the condition repair request. Thus, one of the original neighbors involved in the condition will have had a port filled. However, two computers are still in need of a connection, the other original neighbor and the computer that is now disconnected from the computer that received the condition repair request. Those two computers send out port connection requests. If those two computers are not neighbors, then they will connect to

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1118 of 1657

each other when they receive the requests. If, however, the two computers are neighbors, then they repeat the condition repair process until two non-neighbors are in need of connections.

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

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

Figures 5E and 5F further illustrate the neighbors with empty ports condition. Figure 5E illustrates the neighbors with empty ports condition in the small regime. In this

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1119 of 1657

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

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

Port Selection

As described above, the TCP/IP protocol designates ports above number 2056 as user ports. The broadcast technique uses five user port numbers on each computer: one external port and four internal ports. Generally, user ports cannot be statically allocated to an application program because other applications programs executing on the same computer may use conflicting port numbers. As a result, in one embodiment, the computers connected to the broadcast channel dynamically allocate their port numbers. Each computer could simply try to locate the lowest number unused port on that computer and use that port as the call-in port. A seeking computer, however, does not know in advance the call-in port number of the portal computers when the port numbers are dynamically allocated. Thus, a seeking computer needs to dial ports of a portal computer starting with the lowest port number when locating the call-in port of a portal computer. If the portal computer is

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1120 of 1657

connected to (or attempting to connect to) the broadcast channel, then the seeking computer would eventually find the call-in port. If the portal computer is not connected, then the seeking computer would eventually dial every user port. In addition, if each application program on a computer tried to allocate low-ordered port numbers, then a portal computer may end up with a high-numbered port for its call-in port because many of the low-ordered port numbers would be used by other application programs. Since the dialing of a port is a relatively slow process, it would take the seeking computer a long time to locate the call-in port of a portal computer. To minimize this time, the broadcast technique uses a port ordering algorithm to identify the port number order that a portal computer should use when finding an available port for its call-in port. In one embodiment, the broadcast technique uses a hashing algorithm to identify the port order. The algorithm preferably distributes the ordering of the port numbers randomly through out the user port number space and only selects each port number once. In addition, every time the algorithm is executed on any computer for a given channel type and channel instance, it generates the same port ordering. As described below, it is possible for a computer to be connected to multiple broadcast channels that are uniquely identified by channel type and channel instance. The algorithm may be "seeded" with channel type and channel instance in order to generate a unique ordering of port numbers for each broadcast channel. Thus, a seeking computer will dial the ports of a portal computer in the same order as the portal computer used when allocating its call-in port.

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1121 of 1657

possible that eight seeking computers could be simultaneously dialing ports in different sequences which would reduce the chances of dialing a busy port.

Locating a Portal Computer

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1122 of 1657

then the one seeking computer can share that it has searched to a depth of eight with another seeking computer. If that other seeking computer has searched to a depth of, for example, only four, it can skip searching through depths five through eight and that other seeking computer can advance its searching to a depth of nine.

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

Identifying Neighbors for a Seeking Computer

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1123 of 1657

traveled far enough will detect this condition of already being a neighbor and send the message to a randomly selected neighbor.

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

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

External Data Representation

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1124 of 1657

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

M-Regular

In the embodiment described above, each fully connected computer has four internal connections. The broadcast technique can be used with other numbers of internal connections. For example, each computer could have 6, 8, or any even number of internal connections. As the number of internal connections increase, the diameter of the broadcast channel tends to decrease, and thus propagation time for a message tends to decrease. The time that it takes to connect a seeking computer to the broadcast channel may, however, increase as the number of internal connections increases. When the number of internal connectors is even, then the broadcast channel can be maintained as m-regular and m-connected (in the steady state). If the number of internal connections is odd, then when the broadcast channel has an odd number of computers connected, one of the computers will have less than that odd number of internal connections. In such a situation, the broadcast network is neither m-regular nor m-connected. When the next computer connects to the broadcast channel, it can again become m-regular and m-connected. Thus, with an odd number of internal connections, the broadcast channel toggles between being and not being m-regular and m-connected.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1125 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1126 of 1657

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.

An Information Delivery Service

In one embodiment, an information delivery service application is implemented using the broadcast channel. The information delivery service allows

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1127 of 1657

participants to monitor messages as they are broadcast on the broadcast channel. Each participant may function as a producer of information, as a consumer of information, or both. The producers broadcast messages on the broadcast channel, and consumers receive the broadcast messages. For example, a sports broadcast channel may be used to disseminate the results of sporting events. Certain organizations, such as the National Football League, may be authorized to broadcast results of sporting events on the broadcast channel. The operators of the broadcast channel may sell subscriptions to the broadcast channel to sports enthusiasts. The information delivery service may be used to distribute a broad range of content including news articles, stock prices, weather alerts, medical alerts, traffic reports, and so on.

The information delivery service may provide a directory web site where consumers can locate and subscribe to broadcast channels of interest. The directory may provide a hierarchical organization of topics of the various broadcast channels. When a user decides to subscribe to a broadcast channel, the broadcaster component and information delivery service application program may be downloaded to the user's computer if not already available on the user's computer. Also, the channel type and channel instance associated with that broadcast channel and the identification of the portal computers for that broadcast channel may be downloaded to the subscriber's computer. The information delivery service may also provide a subscriber identifier that may be used by a portal computer to authorize access to or track who has connected to the broadcast channel.

The information delivery service web site may also allow an entity to create new broadcast channels. For example, the NFL may want a broadcast channel dedicated to the dissemination of information under its control. In which case, the entity would interact with the web site to create the broadcast channel. The creation of the broadcast channel would entail the generation of a channel type and channel instance, the specification of security level (e.g., encrypted messages), the specification of subscriber qualifications, and so on.

A user may subscribe to a broadcast channel for an individual topic, which corresponds to a leaf node in the hierarchy, or a user may subscribe to a category of topics, which corresponds to a non-leaf node in the hierarchy. For example, a user may subscribe to a category of sports scores or subscribe to the topic of NFL scores. In one embodiment, each topic would have its own broadcast channel. As a result, the subscribing to a category of topics would mean subscribing to multiple broadcast channels.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1128 of 1657

Alternatively, a category of topics may have a single broadcast channel. If a user subscribes to just one topic in the category, the information delivery service application program executing at the subscriber's computer would simply disregard messages not related to the topic.

Many different fee structures can be used by the information delivery service. A subscriber may be charged a fixed fee per month for subscribing to a topic. Alternatively, a subscriber may be charged based on time actually connected. For example, when a subscriber's computer is connected, it might broadcast an identification message every hour or so. A billing computer could monitor the broadcast and record the connect time based on the identification messages. If the billing computer does not receive an identification message for a certain time period, it assumes that the subscriber's computer has disconnected. Also, the operator of the broadcast channel may derive revenue from advertisements broadcast over the broadcast channel. The fee for advertising on a broadcast channel may vary based on the number of subscribers connected to the broadcast channel at the time the advertisement is broadcast.

The following tables list messages sent by the broadcaster components.

EXTERNAL MESSAGES

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1129 of 1657

	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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1130 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1131 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1132 of 1657

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. In one embodiment, a seeking computer may always search its entire search depth and find multiple portal computers through which it can connect to the broadcast channel. In block 1102, the routine restarts the process of connecting to the broadcast channel and returns. In block 1103, the routine dials the process of one of the found portal computers through the call-in port. In decision block 1104, if the dialing is successful, then the routine continues at block 1105, else the routine continues at block 1113. The dialing may be unsuccessful if, for example, the dialed process recently disconnected from the broadcast channel. In block 1105, the routine sends an external message to the dialed process requesting a connection to the broadcast channel (*i.e.*, connection_request_call). In block 1106, the routine receives the response message (*i.e.*, connection_request_resp). In decision block 1107, if the response message is successfully received, then the routine continues at block 1108, else the routine continues at block 1113. In block 1108, the routine sets the expected number of holes (*i.e.*,

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1133 of 1657

empty internal connections) for this process based on the received response. When in the large regime, the expected number of holes is zero. When in the small regime, the expected number of holes varies from one to three. In block 1109, the routine sets the estimated diameter of the broadcast channel based on the received response. In decision block 1111, if the dialed process is ready to connect to this process as indicated by the response message, then the routine continues at block 1112, else the routine continues at block 1113. In block 1112, the routine invokes the add neighbor routine to add the answering process as a neighbor to this process. This adding of the answering process typically occurs when the broadcast channel is in the small regime. When in the large regime, the random walk search for a neighbor is performed. In block 1113, the routine hangs up the external connection with the answering process computer and then returns.

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1134 of 1657

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 1112, else the routine continues at block 1212. In decision block 1412, if the message type is a condition repair statement (*i.e.*, condition_repair_stmt), then the routine invokes the handle condition repair routine in block 1413, else the routine loops to block 1414 to process the next message. After each handling routine is invoked, the routine loops to block 1414. In block 1414, the routine hangs up on the external port and continues at block 1401 to receive the next message.

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.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1135 of 1657

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.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1136 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1137 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1138 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1139 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1140 of 1657

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.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1141 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1142 of 1657

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.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1143 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1144 of 1657

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,

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1145 of 1657

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.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1146 of 1657

CLAIMS

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

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

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

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

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

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

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1147 of 1657

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

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

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

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

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

14. A information delivery service comprising:

a plurality of broadcast channels, each broadcast channel for distributing information relating to a topic;

means for identifying a broadcast channel for a topic of interest; and

means for connecting to the identified broadcast channel.

15. The information delivery service of claim 14 wherein means for identifying a topic of interest includes accessing a web server that maps topics to corresponding broadcast channel.

16. The information deliver service of claim 14 wherein a broadcast channel is formed by subscriber computers that are each interconnected to at least three other subscriber computers.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1148 of 1657

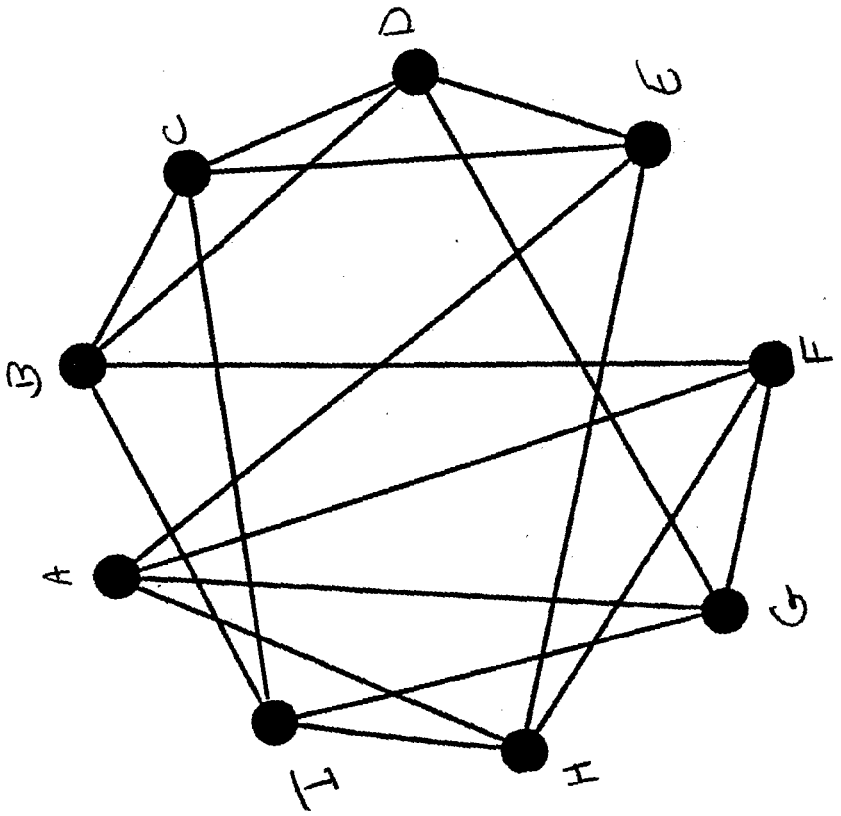


Fig 1

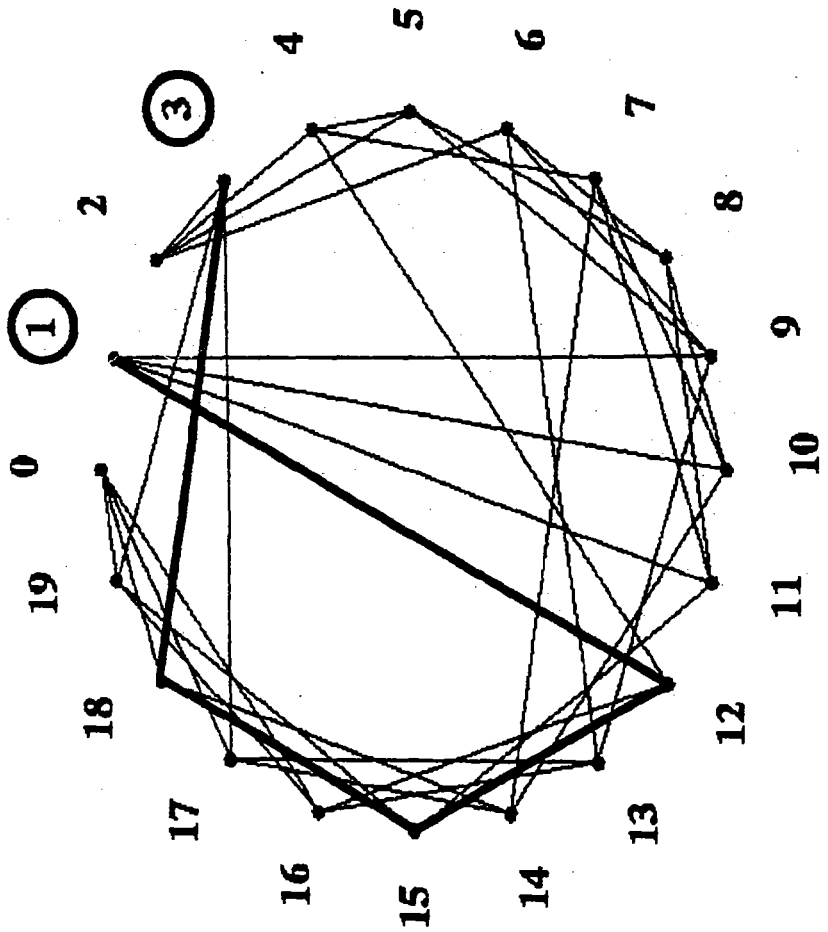


Fig 2

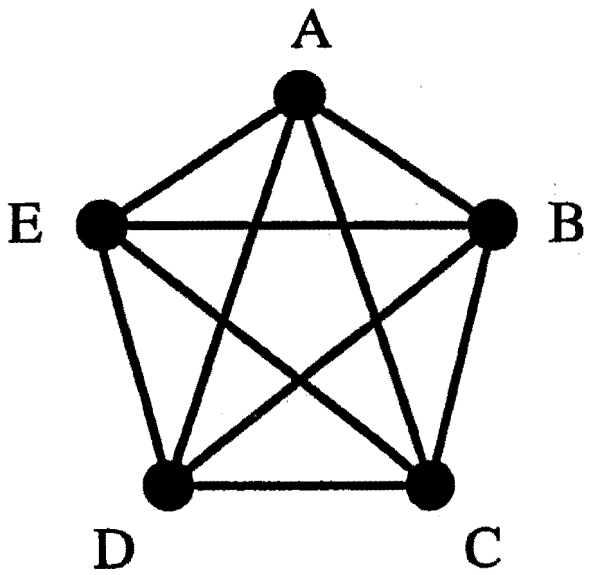


Fig 3A

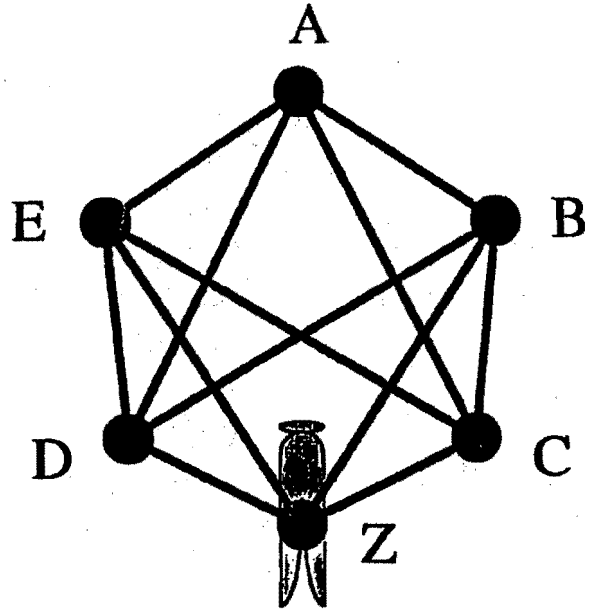


Fig 3B

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1151 of 1657

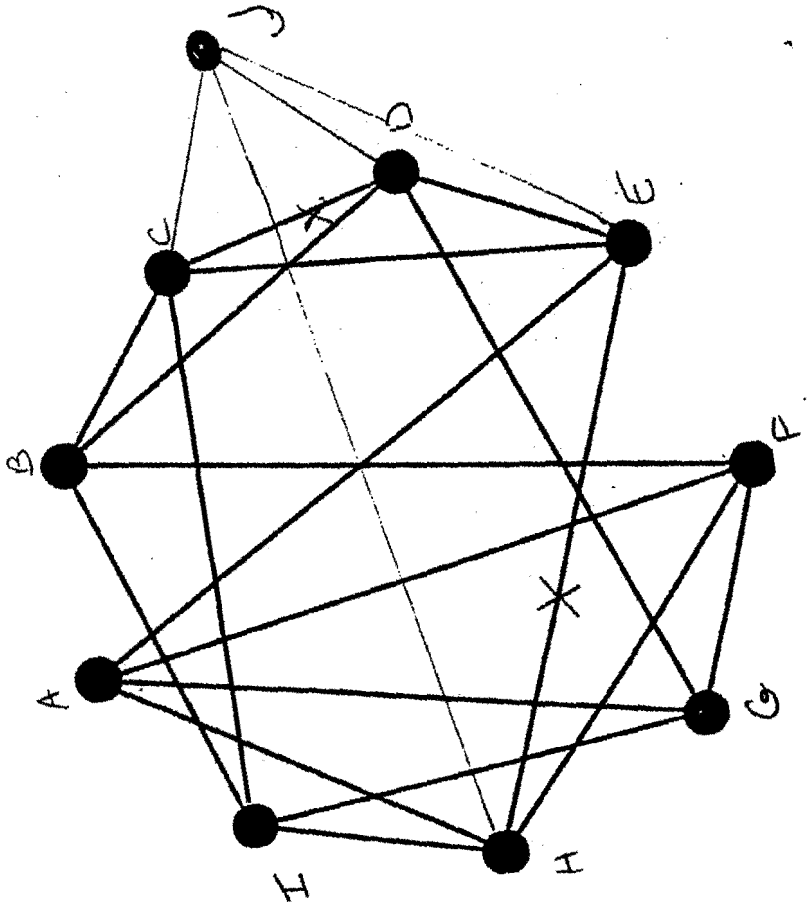


Fig 4A

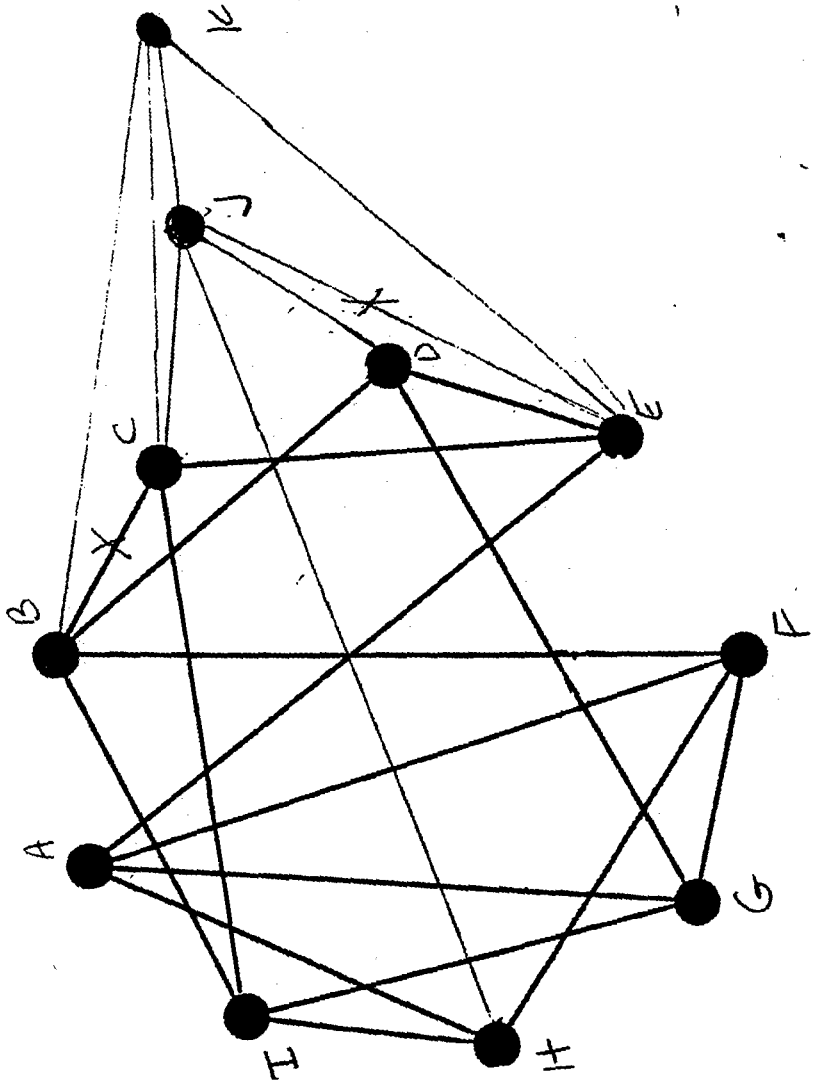


Fig 4B

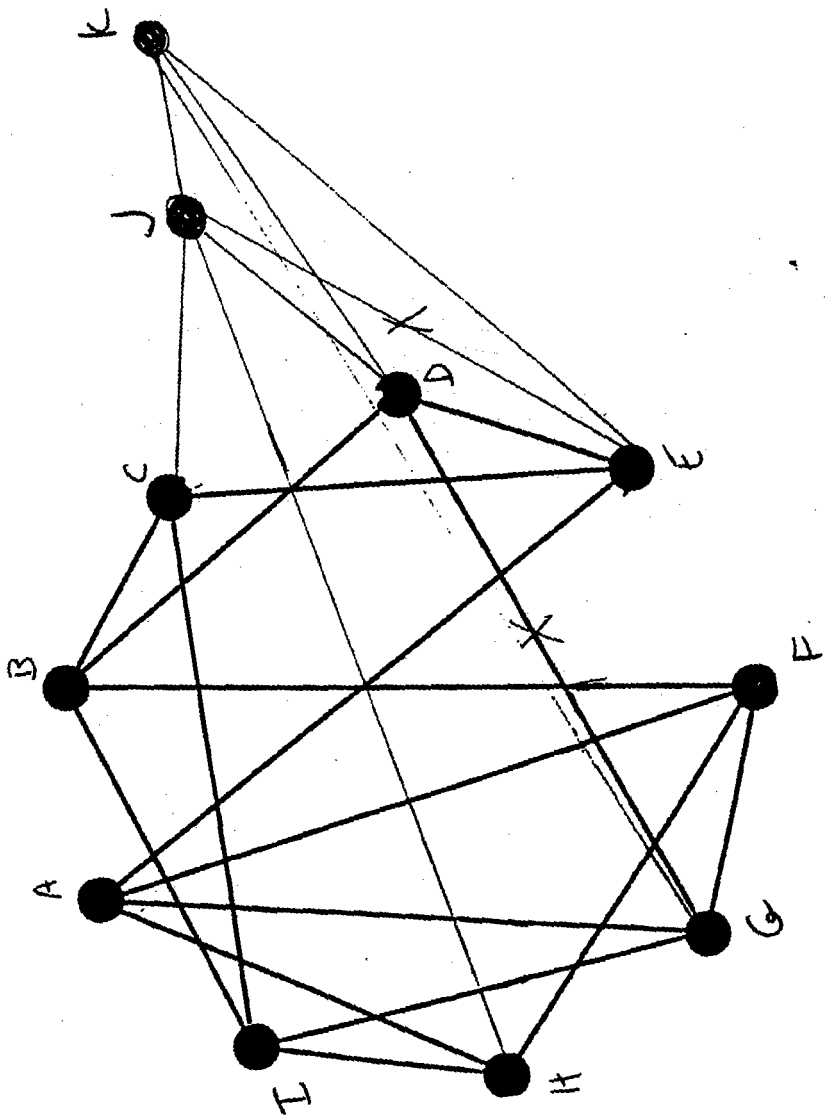


Fig 4C

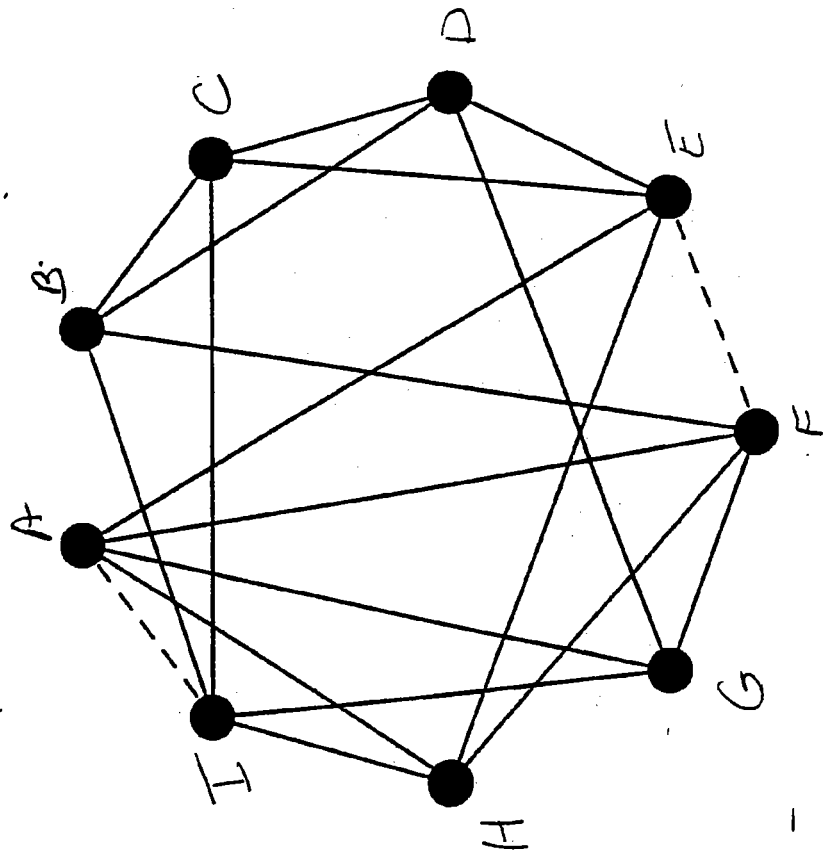


FIG 5A

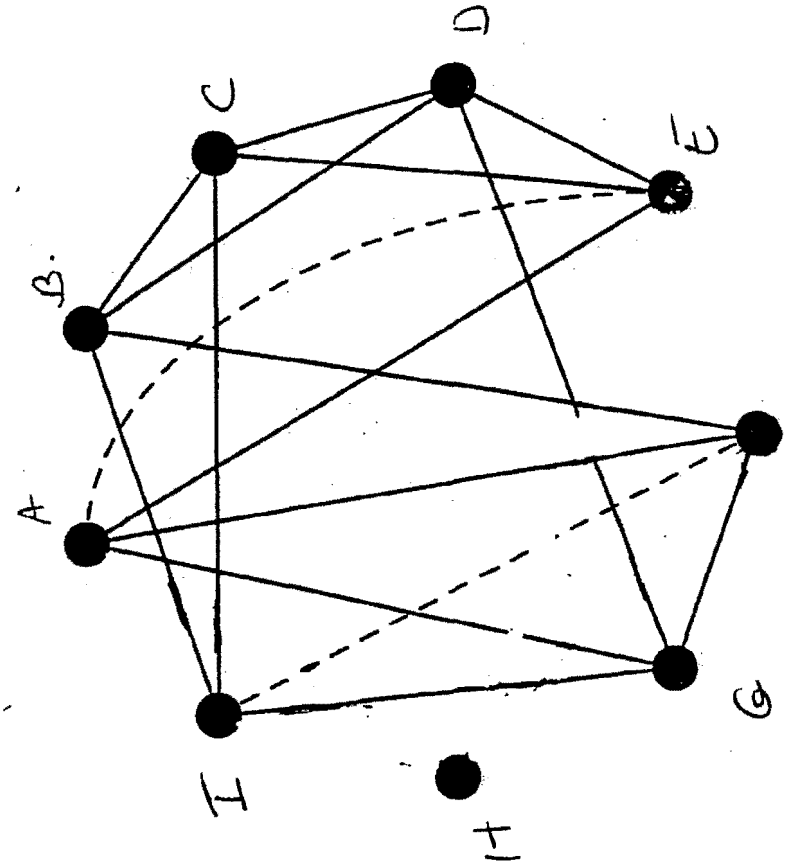


Fig 5B

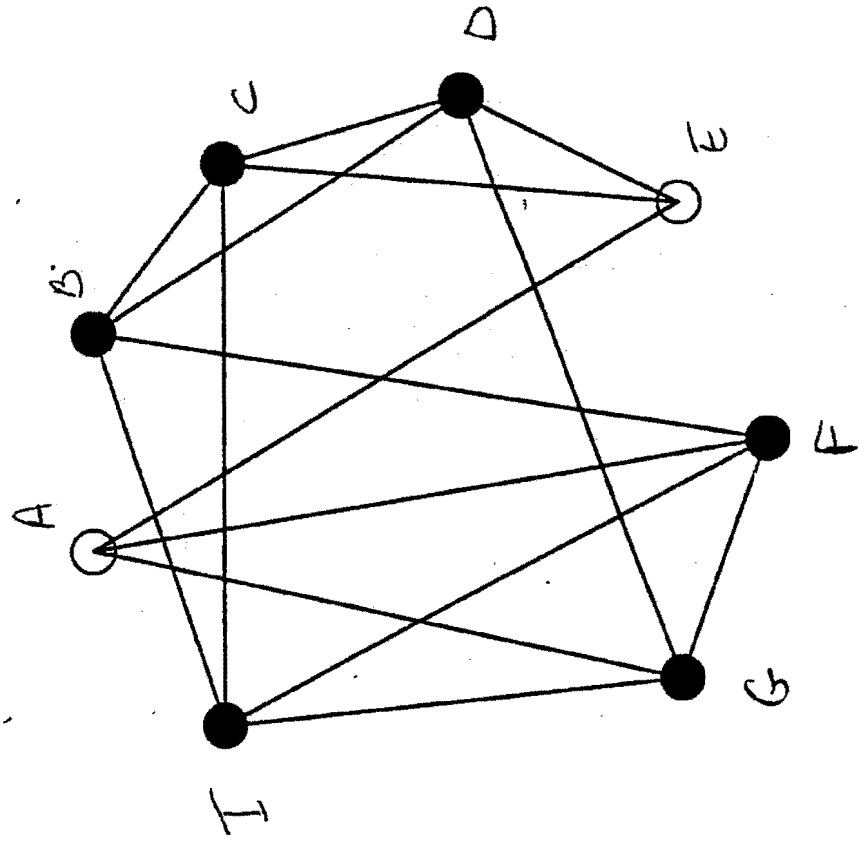


Fig 5C

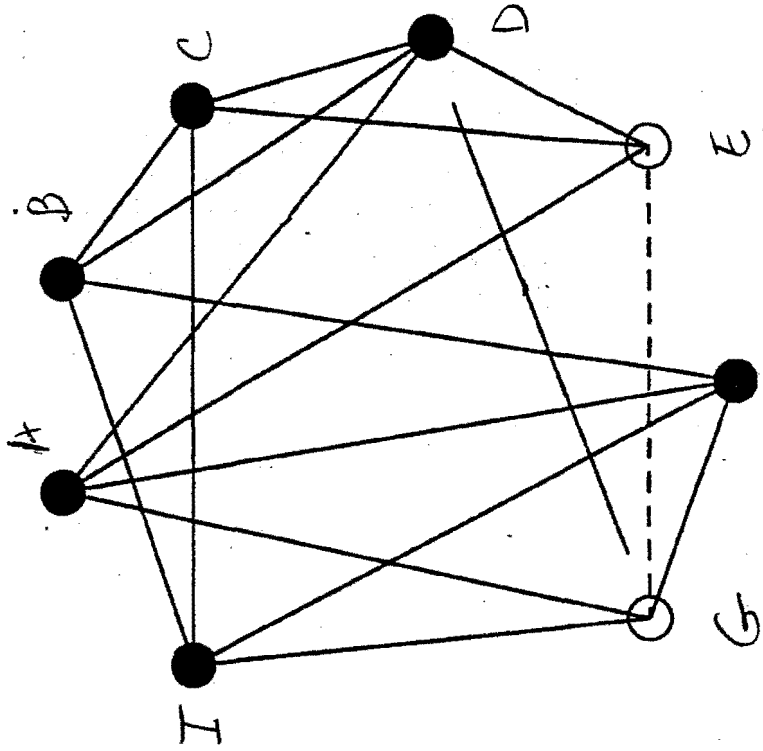


Fig 5D

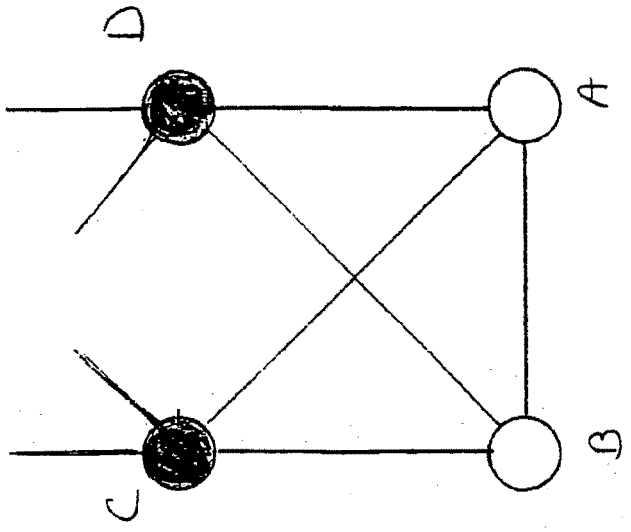


Fig 5F

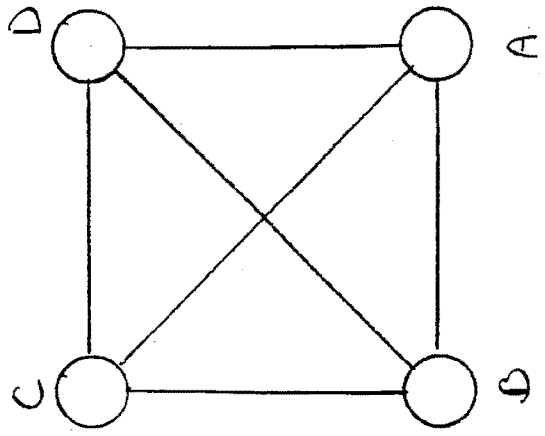


Fig 5E

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1160 of 1657

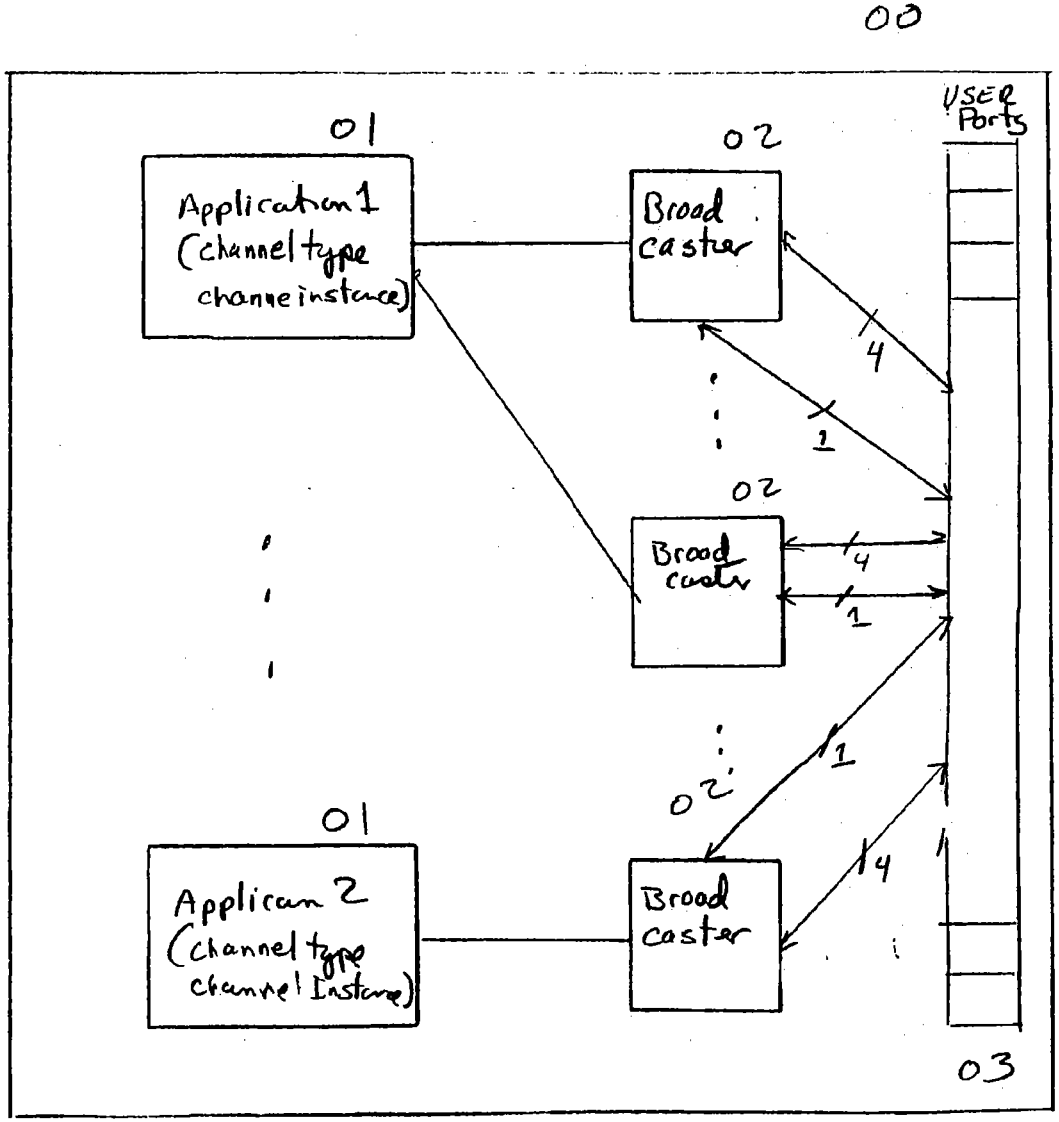


Fig 6

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1161 of 1657

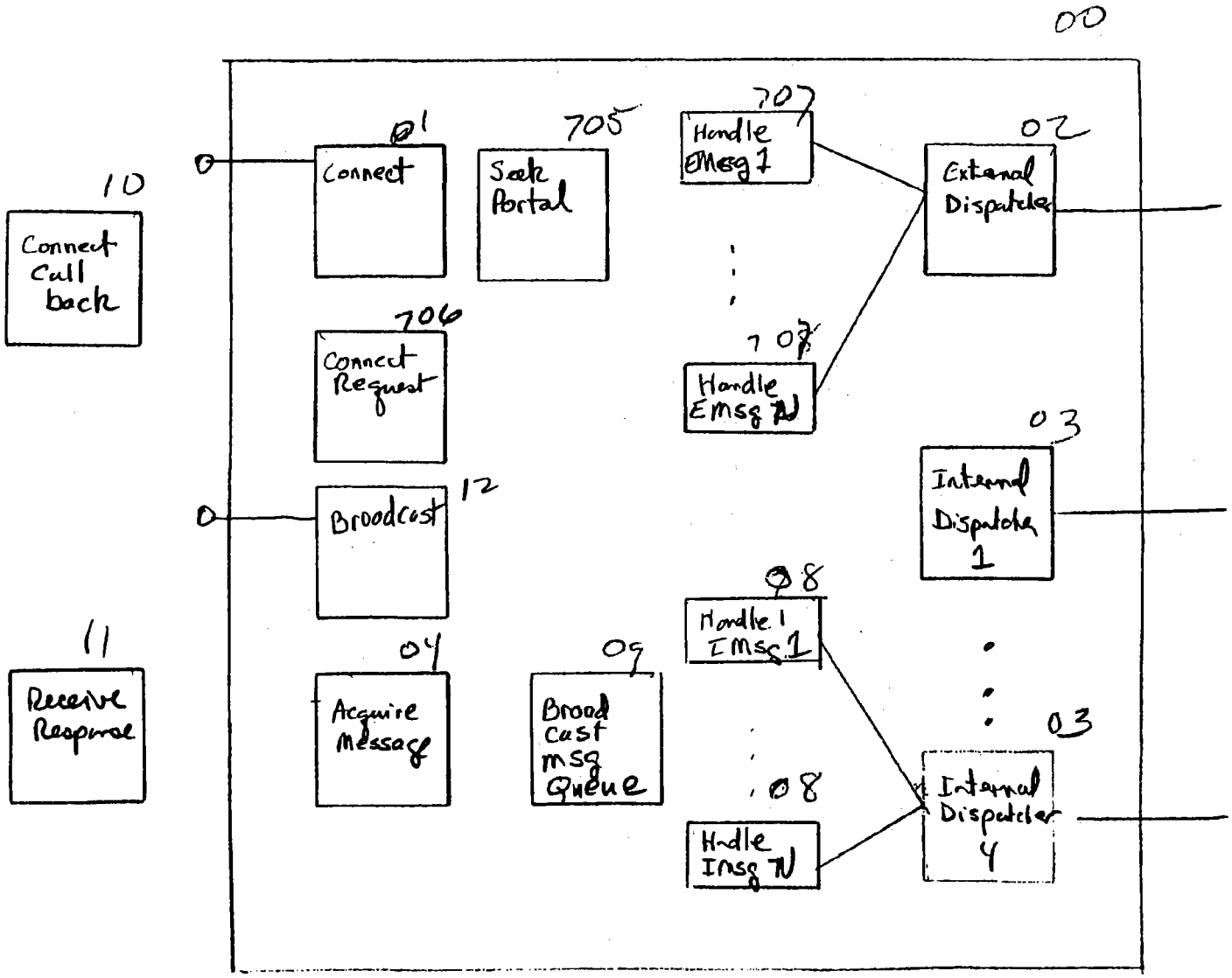


Figure 7

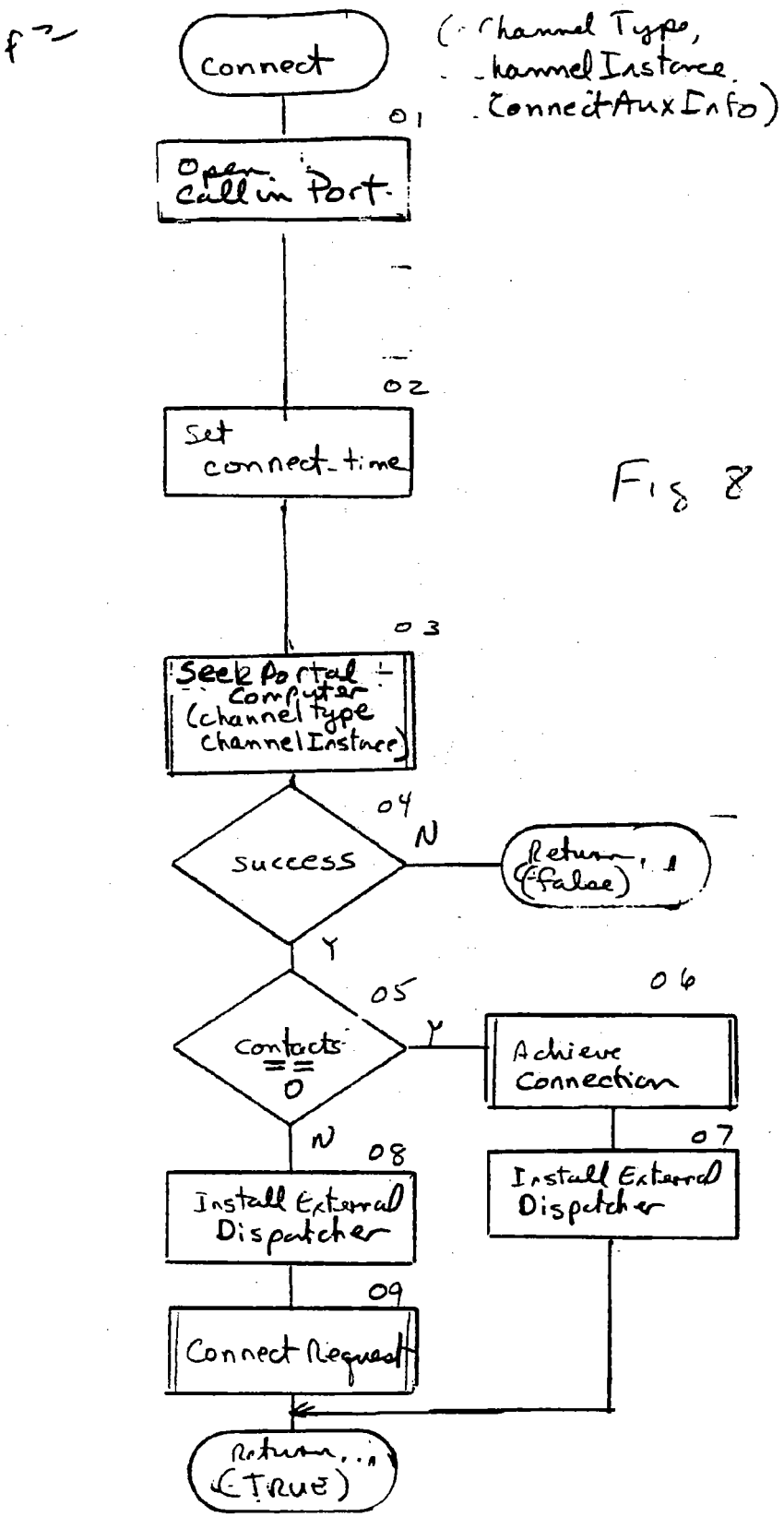

Connect
(channel Type,
channel Instance,
ConnectAuxInfo)
01
Open call in Port
02
Set connect_time
03
Seek Portal in Computer (channel type channel Instance)
04
success
N
Return (false)
Y
05
Contacts == 0
Y
06
Achieve Connection
07
Install External Dispatcher
N
08
Install External Dispatcher
09
Connect Request
Return (TRUE)

Fig 8

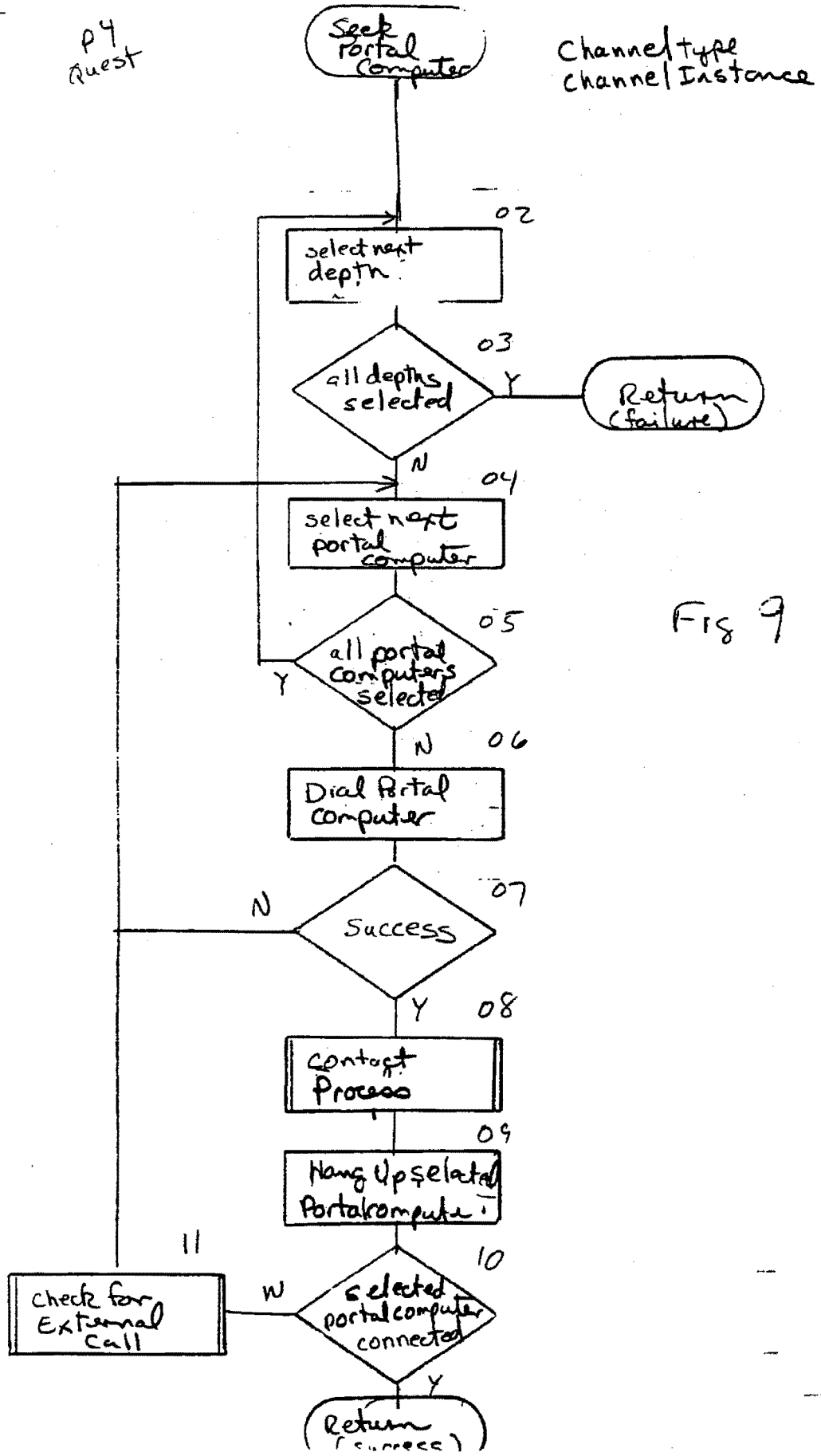

P4
Quest
Seek portal computer
Channel type
Channel Instance
02
select next depth
03
all depths selected
Y
Return (failure)
N
04
select next portal computer
05
all portal computers selected
Y
N
06
Dial Portal computer
07
Success
N
Y
08
Contact Process
09
Hang Up selected Portal computer
10
selected portal computer connected
W
11
Check for External Call
Y
Return

Fig 9

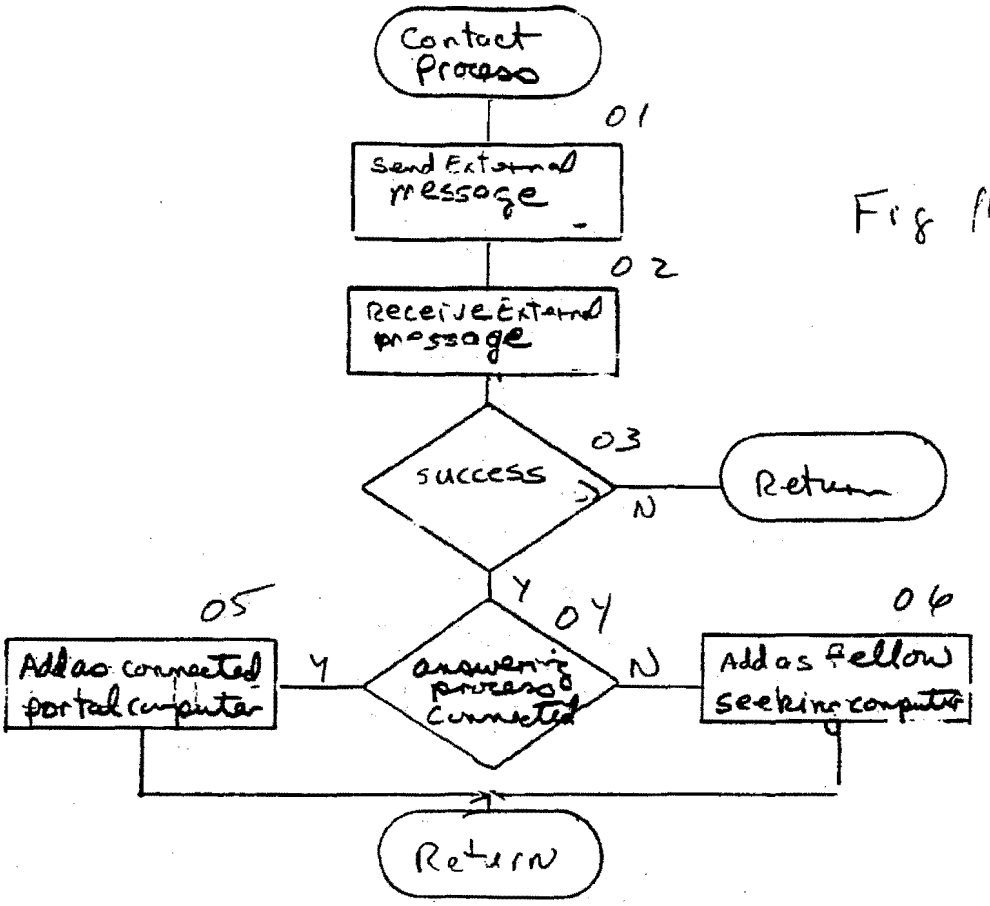
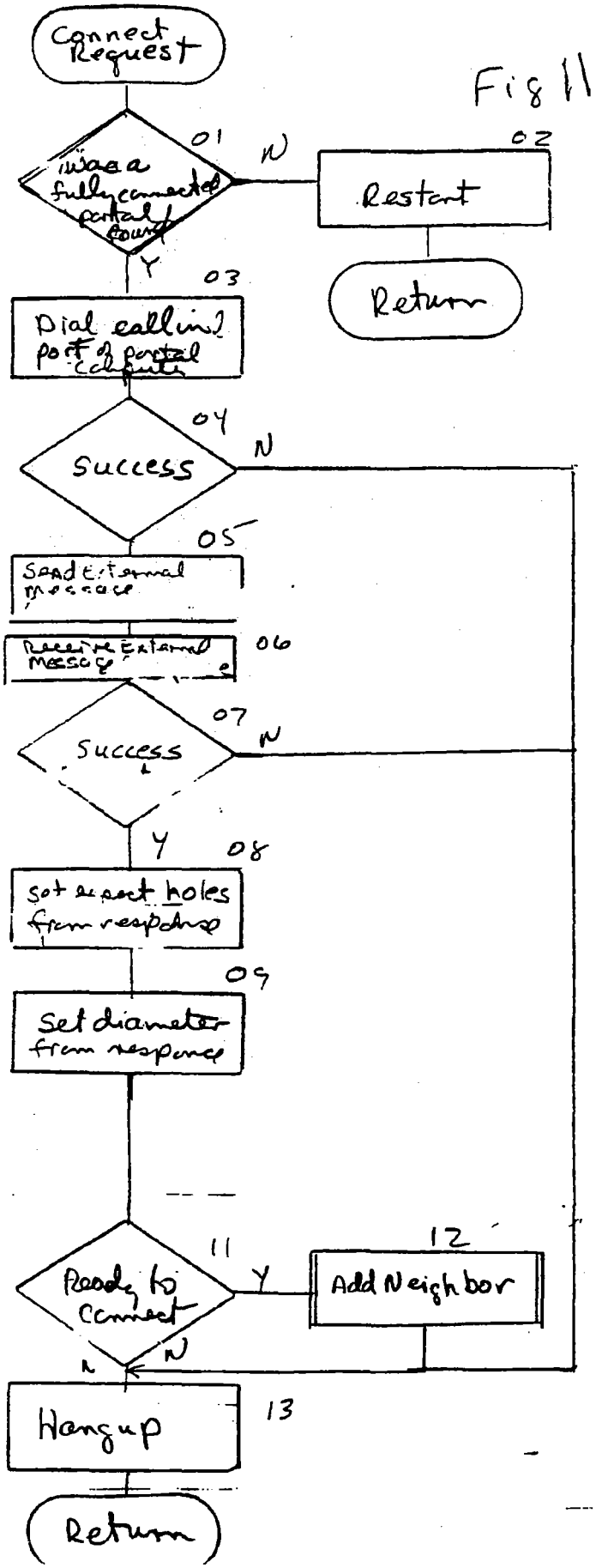

Contact Process
01
Send External message
02
Receive External message
03
success
N
Return
Y
04
05
Add as connected portal computer
Y
answering process connected
N
06
Add as fellow seeking computer
Return

Fig 10

p23

Fig 11


Connect Request
01
N
02
Restart
Return
Y
03
Dial calling port of portal
04
Success
N
05
Send External message
Receive External message
06
07
Success
N
Y
08
Set aspect holes from response
09
Set diameter from response
11
Ready to Connect
Y
12
Add Neighbor
N
N
Hangup
13
Return

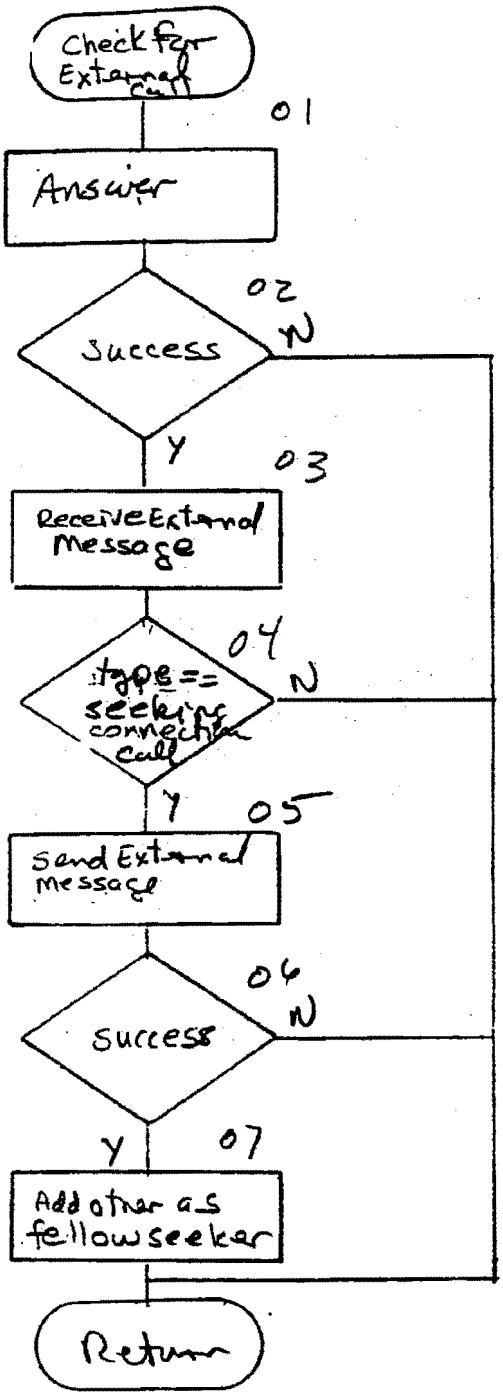

Check for External Call
01
Answer
02
Success
N
Y
03
Receive External Message
04
type == seeking connection call
N
Y
05
Send External Message
06
Success
N
Y
07
Add other as fellow seeker
Return

Fig 12

Achieve Connection
22

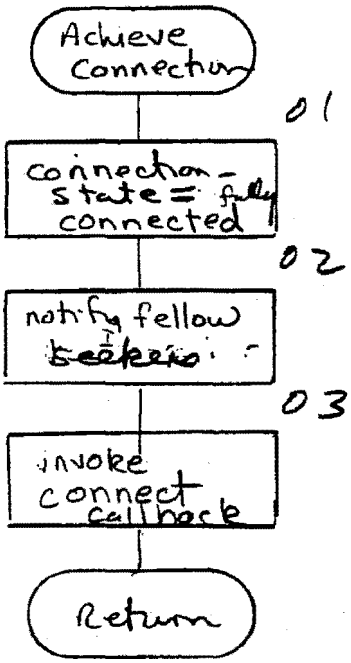
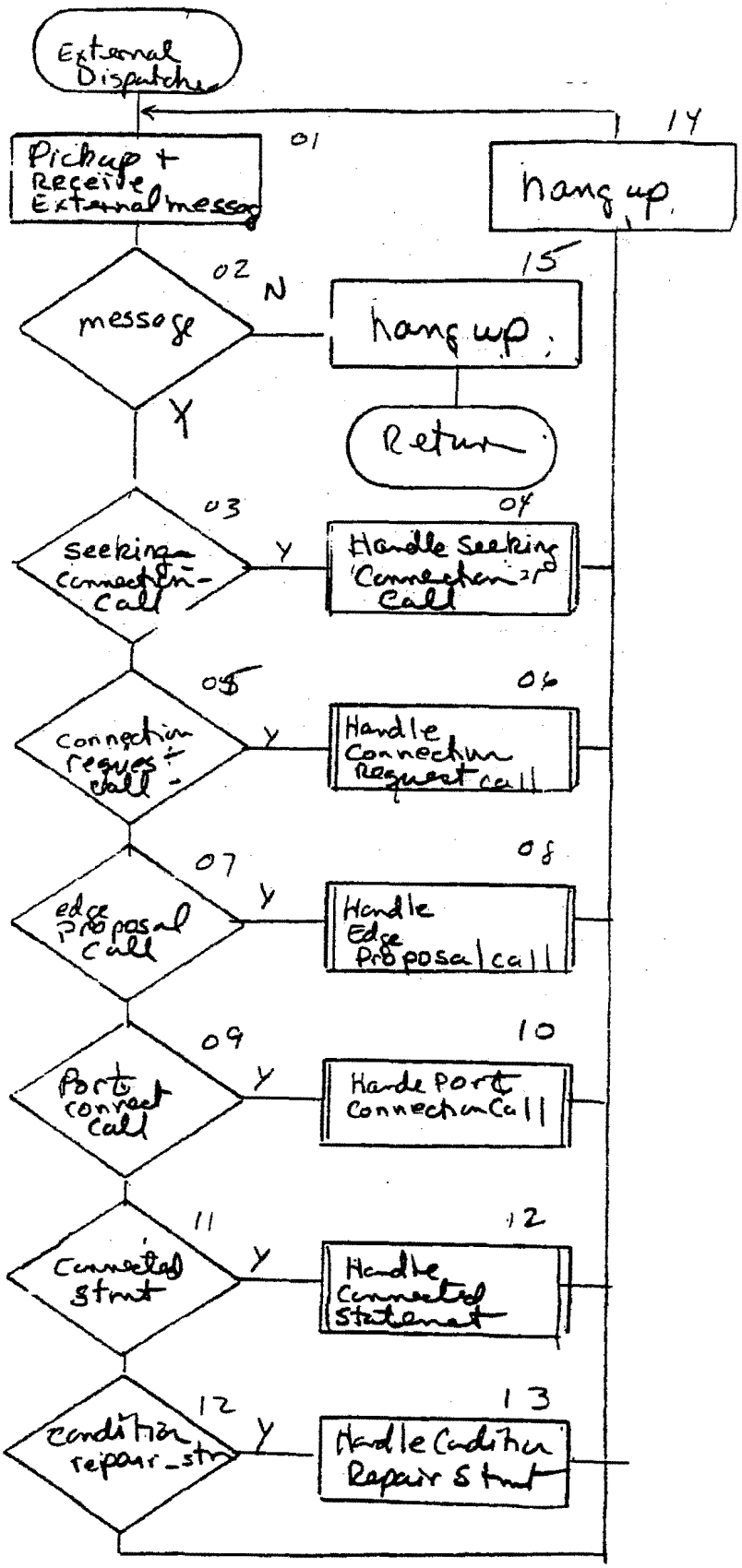


Fig 13

p7

Fig 14


External Dispatcher
01
Pickup + Receive External message
14
hang up
02
message
N
15
hang up;
Return
Y
03
seeking connection-call
Y
04
Handle seeking connection call
05
connection request call
Y
06
Handle Connection Request call
07
edge Proposal Call
Y
08
Handle Edge Proposal call
09
Port connect Call
Y
10
Handle Port Connection Call
11
Connected stmt
Y
12
Handle Connected Statement
12
condition repair_stmt
Y
13
Handle Condition Repair Stmt

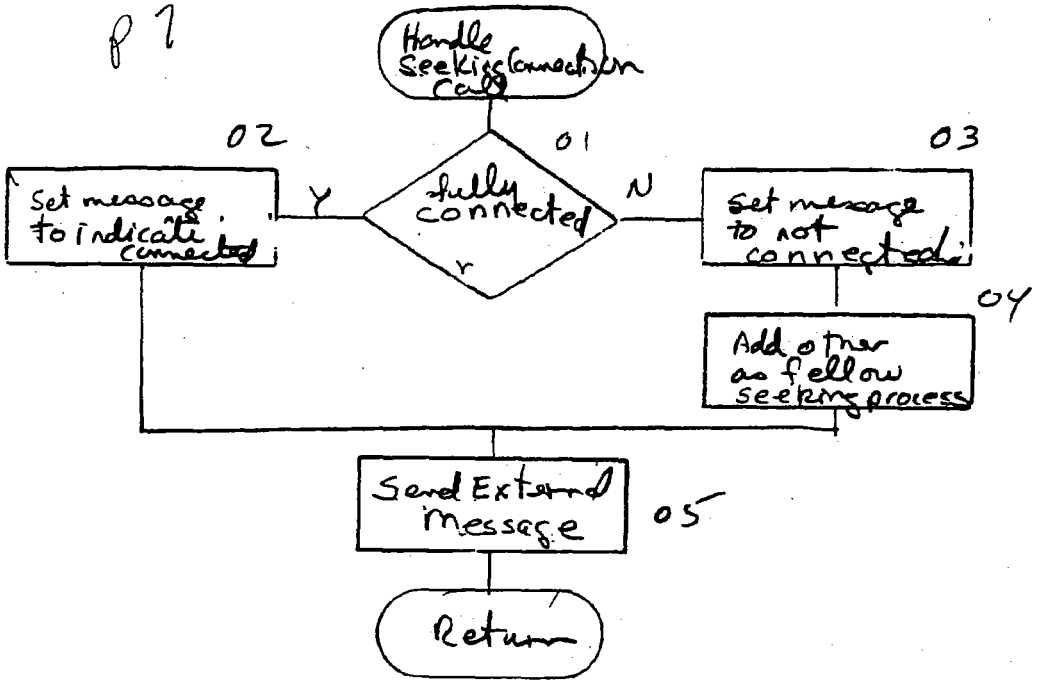
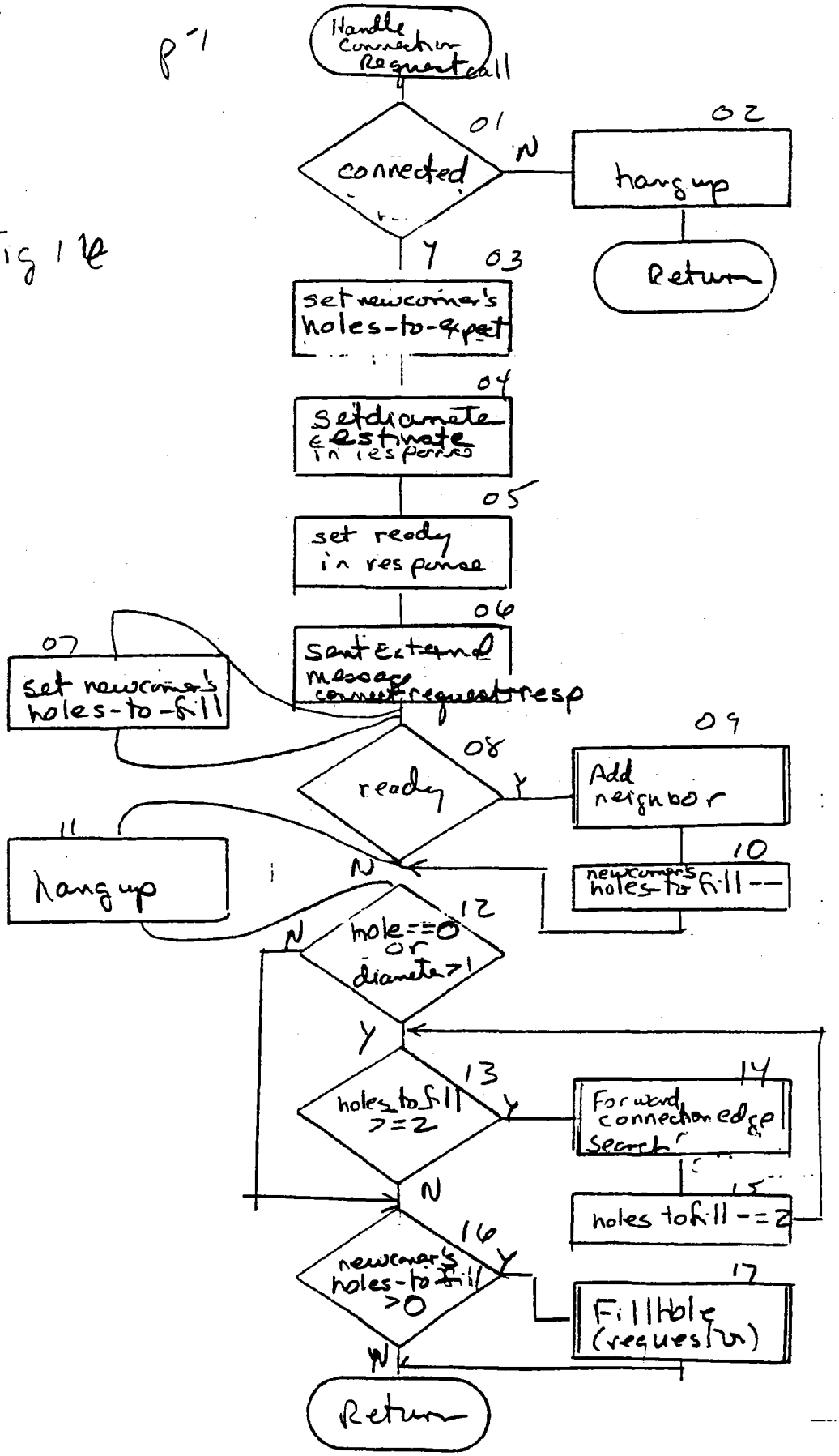

Handle Seeking Connection Call
01
02
03
04
05
Fully connected
Y
N
Set message to indicate connected
Set message to not connected
Add other as fellow seeking process
Send Extend message
Return

Fig 15

p-1

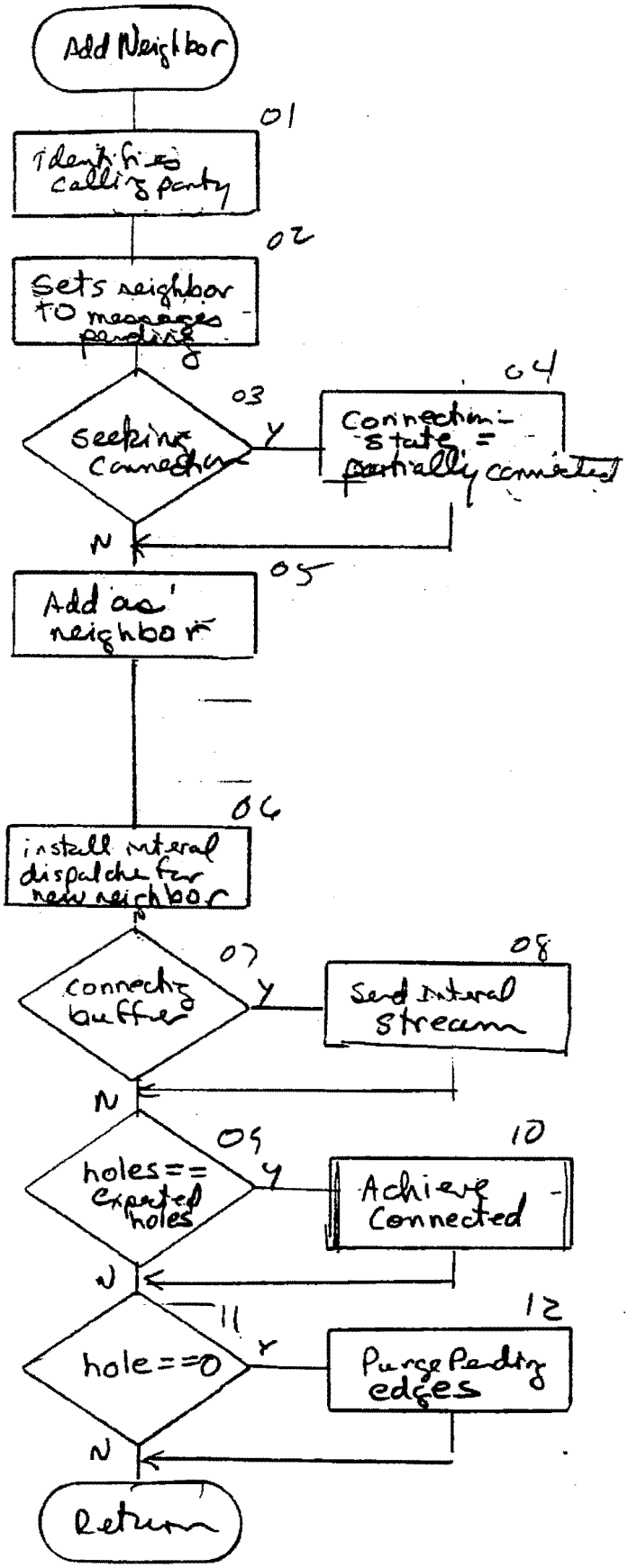
Fig 1e



Handle Connection Request call
01
connected
N
02
hangup
Return
Y
03
set newcomer's holes-to-expect
04
set diameter estimate in response
05
set ready in response
06
sent extend message connect request resp
07
set newcomer's holes-to-fill
08
ready
Y
09
Add neighbor
10
newcomer's holes-to-fill --
11
hangup
N
12
hole==0 or diameter>1
N
Y
13
holes to fill >=2
Y
14
Forward connection edge search
15
holes to fill -=2
N
16
newcomer's holes-to-fill >0
Y
17
Fill Hole (requestor)
N
Return

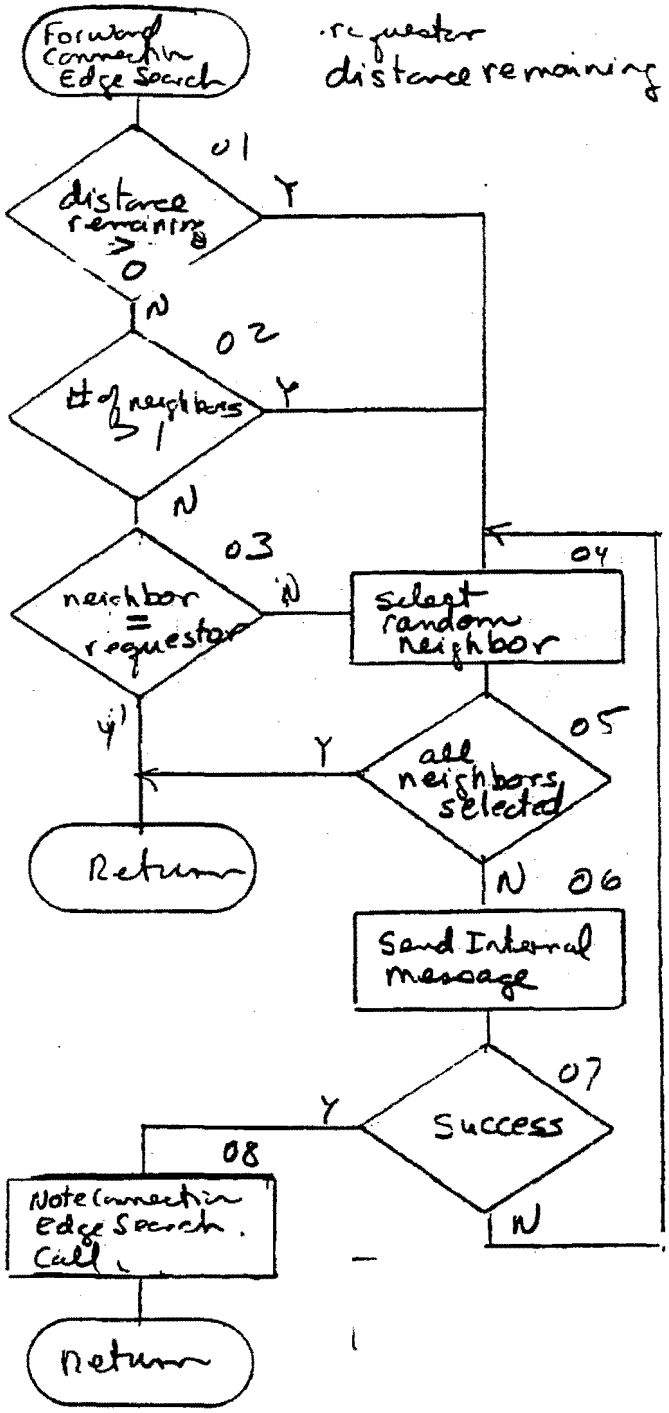
p9

Fig 17


Add Neighbor
01
Identifies calling party
02
Sets neighbor TO messages pending
03
Seeking connection
Y
04
Connection state = partially connected
N
05
Add as neighbor
06
install interal dispatcher for new neighbor
07
connect buffer
Y
08
Send Interal Stream
N
09
holes == expected holes
Y
10
Achieve Connected
N
11
hole == 0
Y
12
Purge Pending edges
N
Return

15

Fig 18


Forward Connection Edge Search
requester
distance remaining
01
distance remaining > 0
Y
N
02
# of neighbors > 1
Y
N
03
neighbor = requester
N
04
select random neighbor
Y
05
all neighbors selected
Y
N
Return
06
Send Internal message
07
Success
Y
N
08
Note Connection Edge Search Call
Return

p. 8

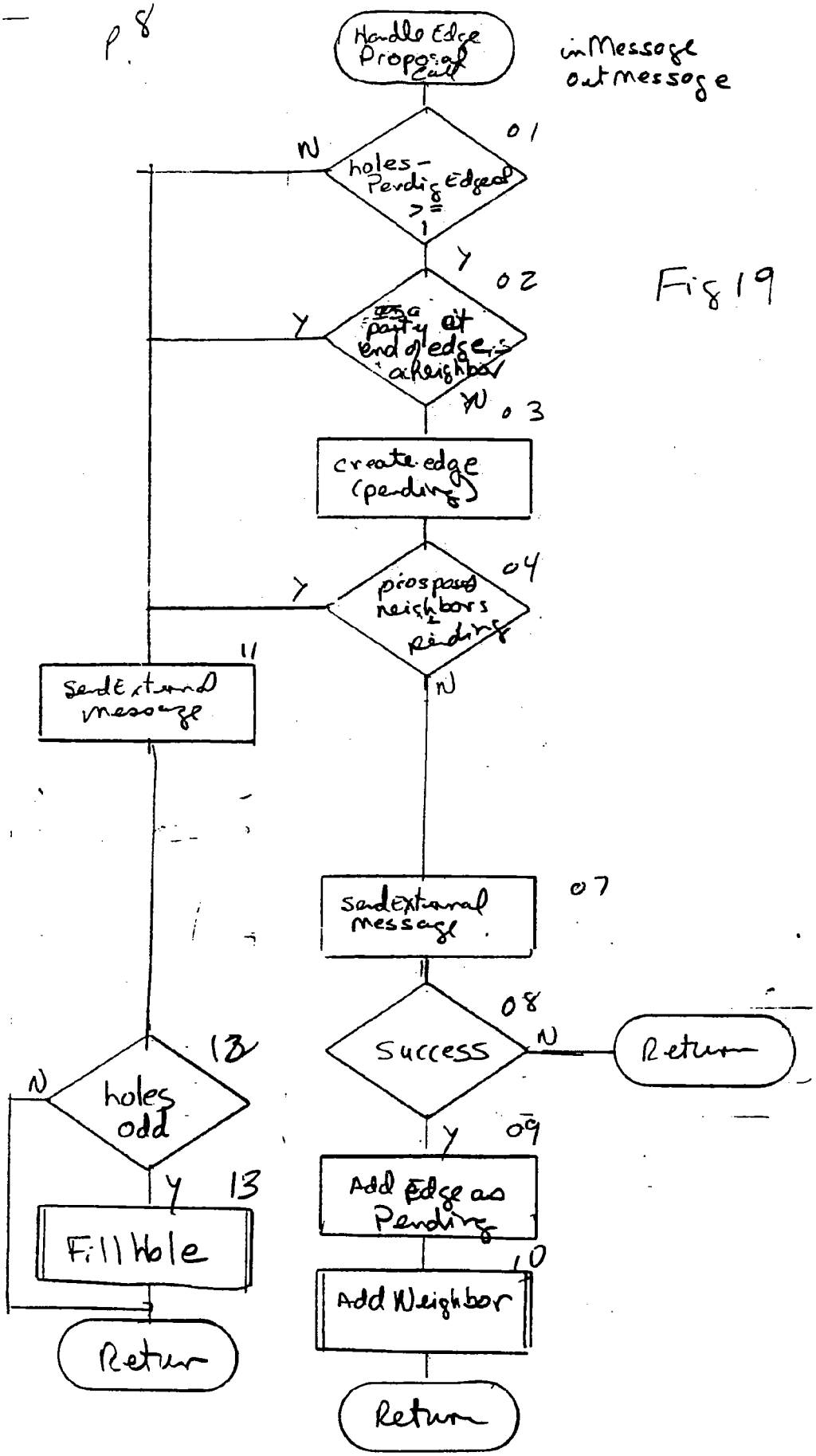

Handle Edge Proposal Call
inMessage
outmessage
01
N
holes - Pending Edges >= 1
Y
02
Y
party at end of edge a neighbor
N
03
create edge (pending)
04
Y
prospect neighbors pending
N
11
Send External message
07
Send External message
08
Success
N
Return
Y
09
Add Edge as Pending
10
Add Neighbor
Return
12
N
holes odd
Y
13
Fill Hole
Return

Fig 19

Fig 20

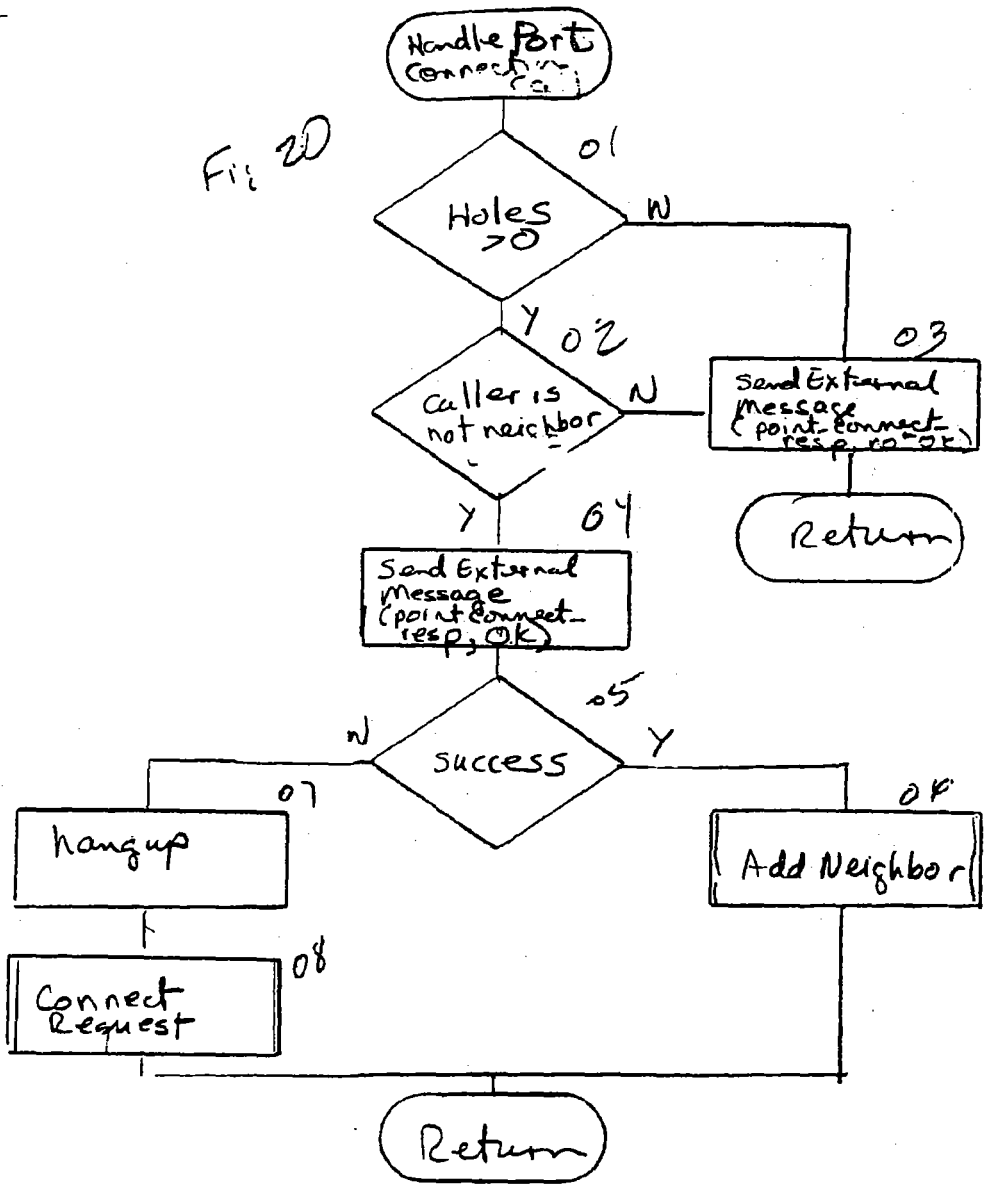
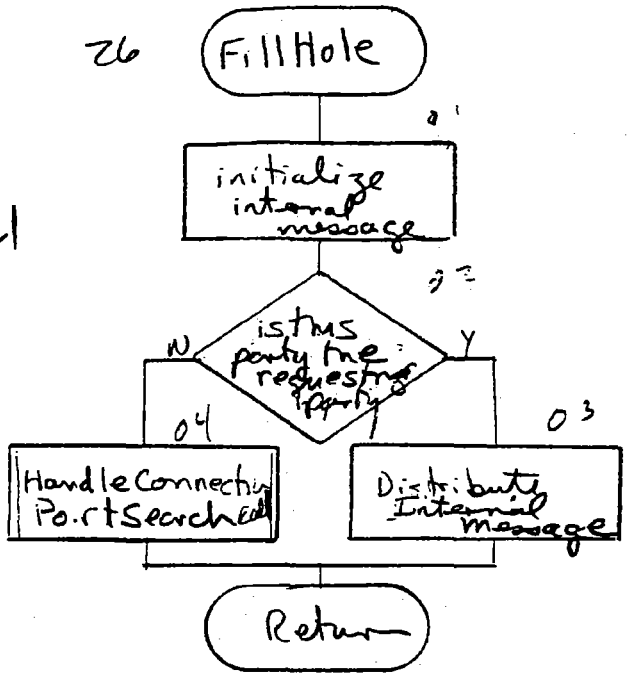

Handle Port Connect
01
Holes >0
N
Y
02
Caller is not neighbor
N
03
Send External Message (point_connect_resp, no_OK)
Return
Y
04
Send External Message (point_connect_resp, OK)
05
success
N
Y
07
hangup
08
Connect Request
06
Add Neighbor
Return

Fig 21


26
FillHole
initialize internal message
is this party the requesting party
N
Y
04
03
Handle Connection PortSearch
Distribute Internal message
Return

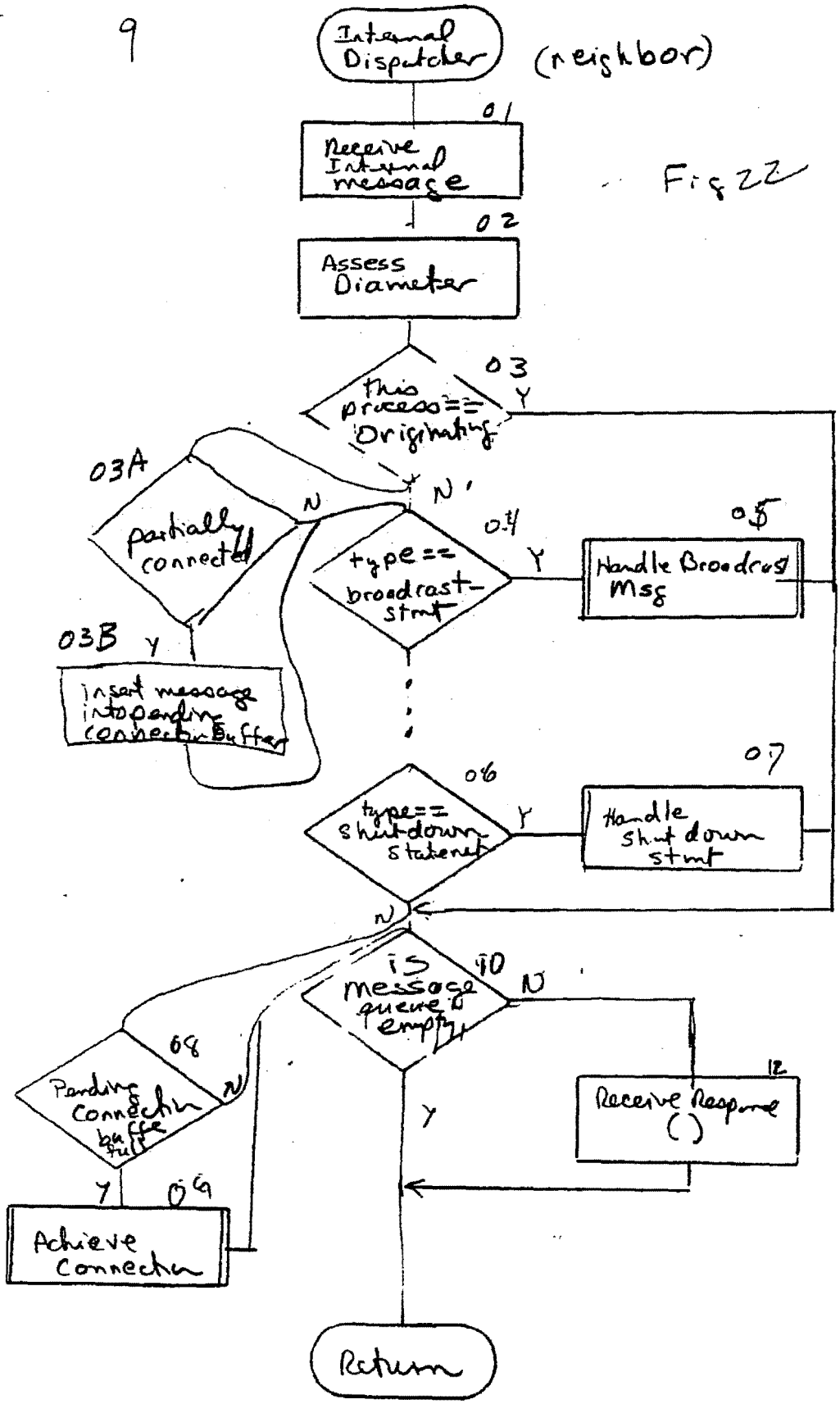

Internal Dispatcher
(neighbor)
01
Receive Internal message
02
Assess Diameter
03
this process == Originating
Y
N'
03A
partially connected
N
03B
Y
insert message into pending connector Buffer
04
type == broadcast-stmt
Y
05
Handle Broadcast Msg
06
type == shut-down statement
Y
07
Handle shut down stmt
N
10
is message queue empty
N
08
Pending Connection buffer full
N
Y
09
Achieve Connection
12
Receive Response ( )
Y
Return

Fig 22

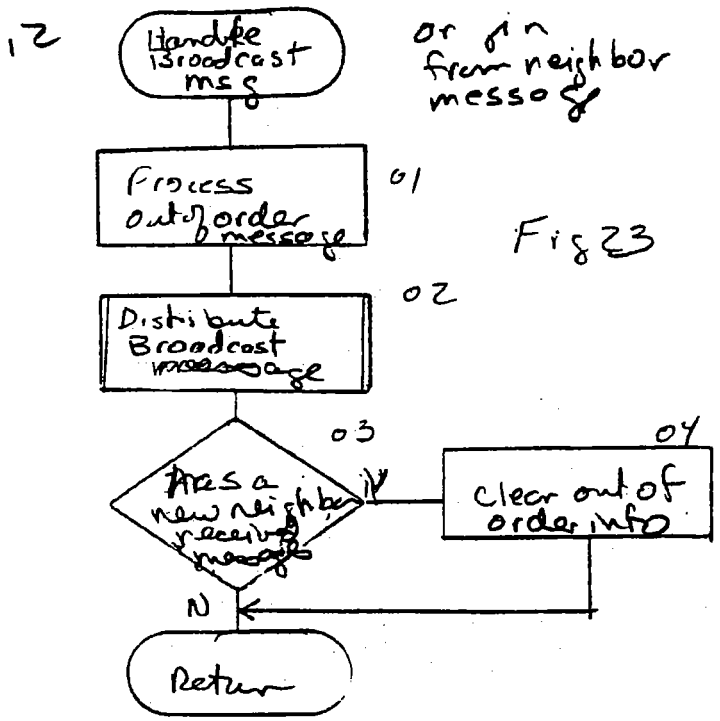

12
Handle Broadcast msg
or from neighbor message
Process out of order message
01
Distribute Broadcast message
02
Is a new neighbor receiving message
03
Y
Clear out of order info
04
N
Return

Fig 23

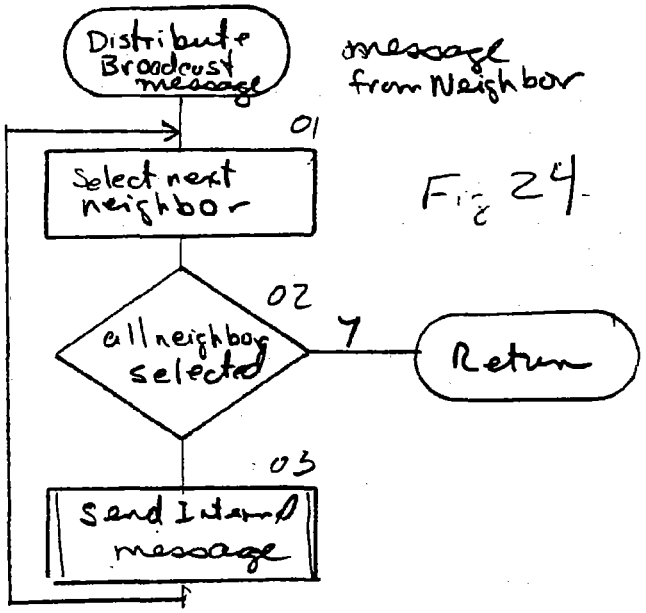

Distribute Broadcast message
message from Neighbor
01
Select next neighbor
02
all neighbor selected
Y
Return
03
Send Internal message

Fig 24.

13

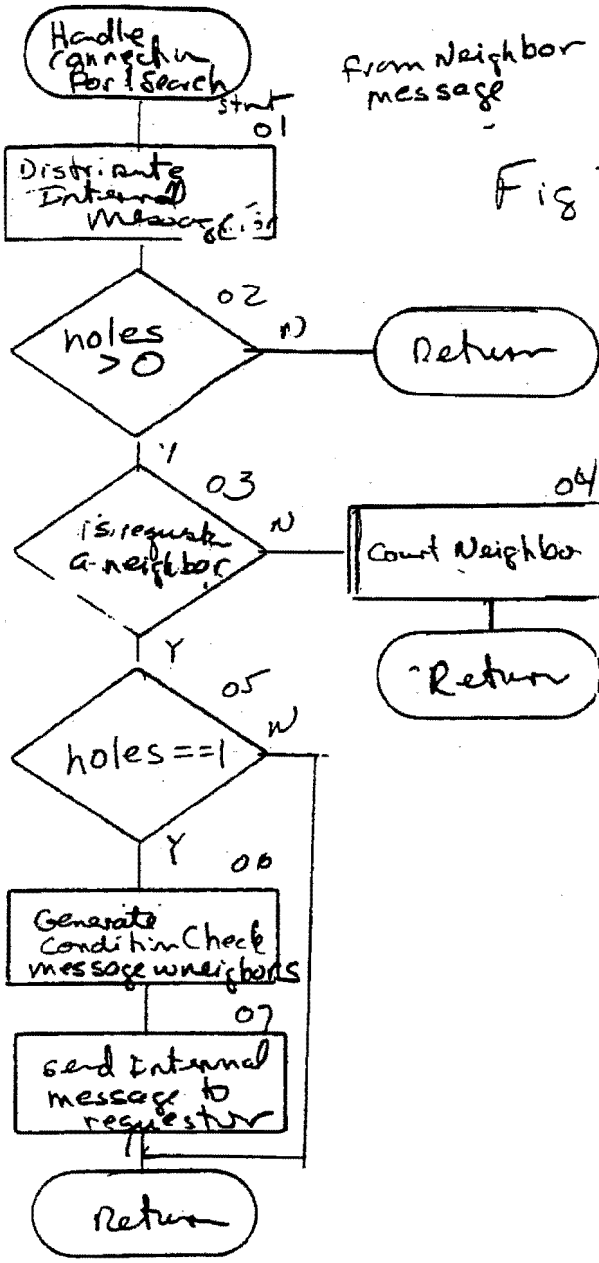

Handle connection for Search
start
01
Distribute Internal Message
02
holes >0
N
Return
Y
03
is request a neighbor
N
04
Count Neighbor
Return
Y
05
holes == 1
N
Y
06
Generate Condition Check message to neighbors
07
send Internal message to requestor
Return
from Neighbor message
21

Fig 26

21

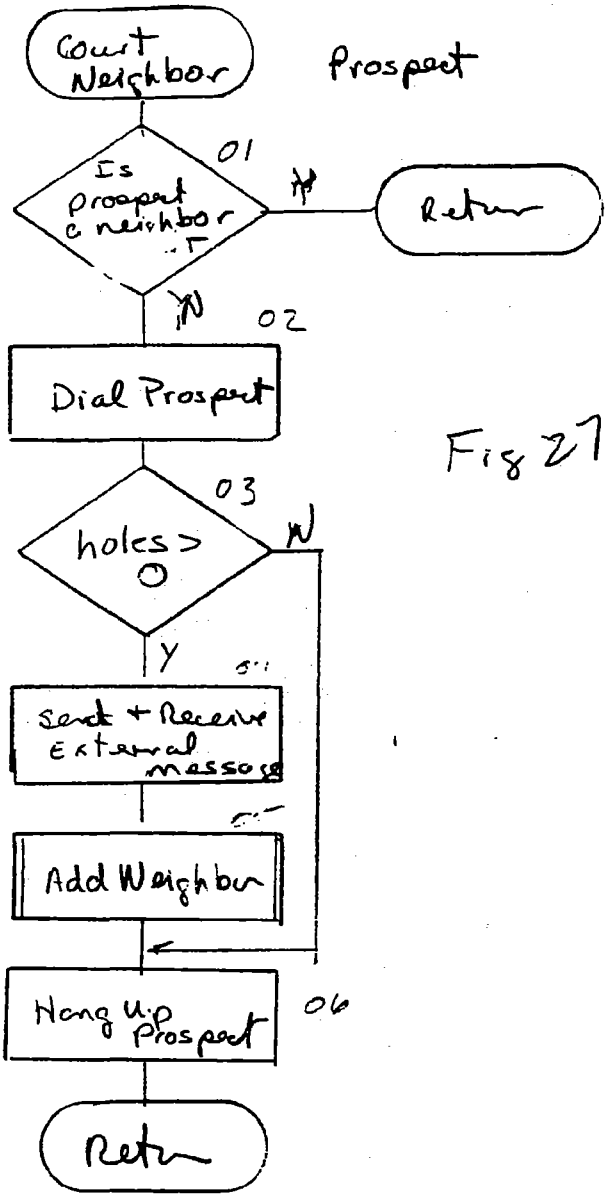

Court Neighbor
Prospect
Is Prospect a neighbor
01
Y
Return
N
02
Dial Prospect
03
holes > 0
N
Y
Send + Receive External messages
Add Neighbor
Hang up Prospect
06
Return

Fig 27

DISTRIBUTED AUCTION SYSTEM

CROSS-REFERENCE TO RELATED APPLICATIONS

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

TECHNICAL FIELD

The described system relates generally to a computer system for conducting an auction, and more particularly to conducting auctions in a distributed environment.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1181 of 1657

BACKGROUND

Because it facilitates electronic communications between vendors and purchasers, the Internet is increasingly being used to conduct "electronic commerce." The Internet comprises a vast number of computers and computer networks that are interconnected through communication channels. Electronic commerce refers generally to commercial transactions that are at least partially conducted using the computer systems of the parties to the transactions. For example, a purchaser can use a personal computer to connect via the Internet to a vendor's computer. The purchaser can then interact with the vendor's computer to conduct the transaction. Although many of the commercial transactions that are performed today could be performed via electronic commerce, the acceptance and wide-spread use of electronic commerce depends, in large part, upon the ease-of-use of conducting such electronic commerce. If electronic commerce can be easily conducted, then even the novice computer user will choose to engage in electronic commerce. Therefore, it is important that techniques be developed to facilitate conducting electronic commerce.

The Internet is also being used to conduct other types of commercial transactions. For example, some server computer systems have been developed to support the conducting of auctions electronically. To conduct an auction electronically, the seller of an item provides a definition of the auction via web pages to a server computer system. The definition includes a description of the item, an auction time period, and optionally a minimum bid. The server computer system then conducts the auction during the specified time period. Potential buyers can search the server computer system for an auction of interest. When such an auction is found, the potential buyer can view the bidding history for the auction and enter a bid for the item. When the auction is closed, the server computer system notifies the winning bidder and the seller (*e.g.*, via electronic mail) so that they can complete the transaction.

Although such auction servers facilitate the conducting of auctions electronically, the conducting of such auctions has several disadvantages. First, the reliability of the auction system depends upon the reliability of the auction server itself.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1182 of 1657

If the auction server were to fail, then the auctions could not be conducted. Thus, one failure can bring the entire auction system down. Second, the auctions conducted by the auction servers to do not closely model traditional non-computer auctions. In particular, the electronic auctions typically close at a fixed time whereas a non-computer auction typically closes when an auctioneer determines that no further bidding is likely. For example, an electronic auction may advertise that it will close at 5 p.m. on a certain day. Bidders can place bids up to that time. Traditional auctions, however, may have a set starting time, but their closing depends on bidding activity. Also, these electronic auctions, especially when web-based, do not provide for real-time notification of bidding activity. A bidder only finds out about being outbid in a couple of ways. The bidder may find out by periodically accessing the auction web page to see the current high bid. Such repeated accessing of the auction web page is cumbersome. Some auction servers may send out electronic mail messages when someone is outbid. Such electronic mail messages may not, however, arrive soon enough for the bidder to place a new bid.

It would be desirable to have an electronic auction system that would avoid these disadvantages of current server-based auction systems and more closely model traditional non-computer auctions.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram illustrating components of the auction system in one embodiment.

Figure 2 is a block diagram illustrating the components of a participant's computer in one embodiment.

Figure 3 is a block diagram illustrating a display of current auctions. Window 300 is displayed by the display status routine.

Figure 4 is a diagram illustrating the display all the auction-specific window.

Figure 5 is a flow diagram of routine to request the current state of the auctions.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1183 of 1657

Figure 6 is the flow diagram of routine that receives a current state request message.

Figure 7 is a flow diagram of routine that receives the current state message.

Figure 8 is a flow diagram illustrating the processing of the submit bid routine in one embodiment.

Figure 9 is a flow diagram illustrating the processing of the receive bid message routine in one embodiment.

Figure 10 is a flow diagram illustrating a routine that processes the expiration of the going timer.

Figure 11 is a flow diagram illustrating a routine that processes a received going message.

Figure 12 is a flow diagram illustrating a routine that processes the expiration of the gone timer.

Figure 13 is a block diagram illustrating a routine that processes a received on message.

Figure 14 is a flow diagram illustrating and auction agent in one embodiment.

DETAILED DESCRIPTION

A method and system for conducting electronic auctions with a distributed auctioneer is provided. In one embodiment, each participant's computer includes an auctioneer component for opening auctions, for accepting bids, and for closing auctions. Thus, the auction system does not depend on a central auction server for coordinating the bidding at an auction. The auction system is, in a sense, server-less. The auction system uses a broadcast channel to communicate between the participants of an auction. Each participant's computer is connected to the broadcast channel and executes an auction participant program. The auction participant program allows a participant to place a bid on the item being auctioned, to receive and display bids of other participants, and to

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1184 of 1657

coordinate the closing of the auction. When a participant places a bid on the item being auctioned, the auction participant program broadcasts a bid message on the broadcast channel. Each auction participant program connected to the broadcast channel receives the bid message and displays the current high bid to its participant. The auction participant program whose participant submitted the high bid coordinates the closing of the auction in accordance with closing rules. For example, when the auction participant program determines that its participant has not been outbid for a certain time period, then the auction participant program may broadcast a "going" message. The going message corresponds to an auctioneer who warns participants that the auction is about to close. If the auction participant program determines that its participant has not been outbid for certain time period after sending the going message, then the auction participant program may broadcast a "gone" message. When the auction participant programs receive the gone message, they notify their participants that the auction is closed. The auction participant program whose participant placed the winning bid then communicates with an auction listing server to complete the transaction. Because the participants are connected through a broadcast channel, each participant receives notification of each bid as it is placed. In addition, the auction system's reliability is not dependent on a central auction server. If any one of the participant's computer fails, then the other participants can continue on with the auction. In one embodiment, the auction system is implemented using the broadcast channel as described in U.S. Patent Application No. _____, entitled "A Broadcasting Network," which is being filed concurrently and which is hereby incorporated by reference. One skilled in the art will, however, appreciate that the auction system can be used with other underlying communication networks.

The auction system may include an auction listing server computer, an auction monitor computer, and participants' computers. The auction listing server computer may provide a web site through which sellers can list their items to be auctioned. When an item is listed, the seller can provide a picture of the item to be auctioned (if appropriate), the minimum bid for the item, and a start time for the auction. Potential bidders can access web pages of the auction listing server to view the listed

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1185 of 1657

auctions. Potential bidders may also download the auction participant program from the auction listing server to their computers. When a user wants to participate in a certain auction, the participant runs the auction participant program which may provide a list of the current auctions being conducted along with the status of each auction. The participant can select a certain auction and place a bid at that auction. Because the two participants may place a bid for the same amount for an item at approximately the same time, the auction participant program awards the bid to the participant based on a random number generated by the bidder's auction participant program. When a bid is placed, the auction participant program automatically generates and includes a random number with a bid message as it is broadcast. Whenever an auction participant program receives a bid for the same amount as the current high bid, the auction participant program awards the bid to the participant with the highest randomly generated number. The auction monitor computer may also be connected to the broadcast channel. The auction monitor tracks the status of the auction by monitoring the bids placed at the auction. The auction monitor may provide the status of the auctions to the auction listing server and to auction participant programs as they join the auction.

Figure 1 is a block diagram illustrating components of the auction system in one embodiment. The auction system includes an auction listing server 101, participant computers 102, and an auction monitor 103. Each computer may include a central processing unit, memory, input devices (*e.g.*, a 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 media that may contain computer instructions that implement the auction system. The computer-readable media may also include computer data transmission media, such as wire-based or wireless communications mechanisms. The participant computers may include a browser for accessing web pages provided by the auction listing server. The participant computers and the auction monitor are connected to the broadcast channel 105. The participant computers, the auction monitor computer, and the auction listing server are interconnected via the Internet 104. The participant computers may use a browser to access auction information provided by the

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1186 of 1657

auction listing server. The auction listing server may include a web engine 106, a create auction component 107, a close auction component 108, and auction database 109. The create auction component is used by a seller to create an auction for an item. The close auction component is used by a winning bidder to affect payment for the item being purchased. The auction listing server may also include a component for registering participants and a participant database. The auction database defines the auctions and may contain the current state of the auction as provided by the auction monitor. One skilled in the art will appreciate that various different communication mechanisms may be used by the auction system. For example, the broadcast channel may actually be implemented using the Internet itself. In addition, multiple auctions may be conducted simultaneously on the broadcast channel. In such cases, each message that is broadcast will include an auction identifier. Alternatively, each auction may have its own broadcast channel. The auction listing server may provide broadcast channel information (*e.g.*, application and session identifier) to each auction participant program. The messages may be encrypted, or otherwise secured, to ensure that only an authorized auction participant program participates in an auction.

Figure 2 is a block diagram illustrating the components of a participant's computer in one embodiment. The participant computer includes a broadcaster component 201, auction participant program 202, and an auction database 203. The broadcaster component controls the connection to, broadcasting message on to, and receiving messages off of the broadcast channel. The auction participant program controls the participating in an auction by sending messages on to and receiving messages off of the broadcast channel using the broadcaster component. The functions of the broadcaster component and the broadcast channel are described in the U.S. Patent Application No. _____, entitled "A Broadcasting Network," which is hereby incorporated by reference. The auction database contains current state information for the auctions. The auction participant program includes a monitor sub-component 105, a message handlers 106, a display state sub-component 107, and a subunit bid sub-component 108. The monitor sub-component monitors the messages sent

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1187 of 1657

on the broadcast channel and invokes the appropriate message handler routine. The display state sub-component displays the current state of the auctions. The submit bid sub-component is invoked when a participant wants to submit a bid at an auction.

Figure 3 is a block diagram illustrating a display of currently defined auctions. Window 300 is displayed by a display state routine. The window includes a sub-window 301 for each auction. Each sub-window may include information describing the auction. When a user selects a sub-window, then the display state routine displays an auction-specific window. Figure 4 is a diagram illustrating the display of an auction-specific window. Window 400 includes an item picture 401, an item description area 402, an auction description area 403, and a place bid button 404. The item picture area may contain the picture of the item being auctioned. The item description area includes a description of the item being auction. The auction description area contains information describing the current status of the auction. For example, the actual auction state may be the start time of the auction, an indication that the auction is in progress, an indication that the auction is "going," and an indication that the auction is closed. The auction description area may also include the minimum bid, the current bid, and a suggested bid amount which may be overridden. When the participant selects the place bid button, the auction participant program submits the bid amount.

Figures 5-14 are flow diagrams illustrating the processing of the auction participant program. The processing in these flow diagrams is illustrated in reference to a single auction. One skilled in the art would appreciate that the processing could be modified to accommodate multiple simultaneous auctions. Figure 5 is a flow diagram of a routine to request the current state of the auction. This routine may be invoked when the auction participant program first starts executing. When the auction participant program starts, it may contact the auction listing server to retrieve the current state of the auction. Alternatively, as shown in block 501, the request current state routine may broadcast a current state request message on the broadcast channel. The auction participant program will receive an indication of the current state of the auction in response. The auction participant program stores that state information in its auction

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1188 of 1657

database. Figure 6 is the flow diagram of routine that receives and processes a current state request message. Each auction participant program may ignore this request if the auction monitor is configured to respond to the request. Alternatively, the auction participant program with the current high bid at the auction may respond by broadcasting a message that includes the current state of the auction. In decision block 601, if this participant has the current high bid, then the routine continues at block 602, else the routine returns. In block 602, the routine broadcasts the state of the auction and then returns. Figure 7 is a flow diagram of a routine that receives the current state message. In block 701, the routine updates the auction state in the auction database and returns.

Figure 8 is a flow diagram illustrating the processing of the submit bid routine in one embodiment. This routine validates the bid amount and then broadcasts that bid. The routine also sets a timer to indicate when a going message should be broadcast to notify the other participants that the auction will end if no participant places a higher bid. In decision block 801, if the bid is valid, then the routine continues at block 802, else the routine returns. The routine determines whether a bid is valid by ensuring that the bid is greater than the current high bid. The routine may also check whether the auction is still open. The auction may have closed since the time the participant selected the place bid button. In block 802, the routine generates a random number that is to be included in the bid message. This random number is used by the receiving participants in case two bids of the same amount are received by those participants. If so, the participants award the bid to the bidder with the highest random number. In block 803, the routine creates a bid message that includes the identification of the participant, the bid amount, and the random number. In the situation where messages for multiple auctions are being broadcast on the broadcast channel, the bid message may also include the auction identifier. In block 804, the routine broadcasts the bid message on the broadcast channel. In block 805, the routine starts a timer for sending the going message. The routine then returns.

Figure 9 is a flow diagram illustrating the processing of the receive bid message routine in one embodiment. This routine is invoked when the auction participant

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1189 of 1657

program receives a bid message from the broadcast channel. This routine validates the bid, updates the auction state, and clears any timers. In decision block 901, if the auction is currently open, the routine continues at block 902, else the routine returns. In decision block 902, if the received bid is greater than or equal to the current high bid, then the routine continues at block 903, else the received bid has already been outbid and the routine returns. In decision block 903, if received bid equals the current high bid, then two participants have bid the same amount and the routine continues at block 904, else the routine continues at block 905. In decision block 904, if the random number included in the received bid message is greater than the random number that was included with the bid message with the current high bid, then the bid will be awarded to the participant who sent the bid message and the routine continues at block 905, else the routine returns. In block 905, the routine replaces the current high bid in the auction database and may update the display. In block 906, the routine clears any timers that may have been set to indicate the end of the auction. The routine then returns.

Figure 10 is a flow diagram illustrating a routine that processes the expiration of the going timer. In block 1001, the routine broadcasts a going message. The going message may identify the participant and the current high bid. In block 1002, the routine sets a timer for the sending of the gone message, which indicates the auction is now closed. The routine then returns. Figure 11 is a flow diagram illustrating a routine that processes a received going message. In decision block 1101, if the going message corresponds to a bid that has already been superseded, then the routine returns, else the routine continues at block 1102. In block 1102, the routine updates the state of the auction, which may include updating the display. The routine then returns.

Figure 12 is a flow diagram illustrating a routine that processes the expiration of the gone timer. In block 1201, the routine broadcasts a gone message, which may identify the participant who is sending the message along with the bid amount. In block 1202, the routine updates the status of the auction to indicate that it is closed. Figure 13 is a block diagram illustrating a routine that processes a received gone message. In block 1301, the routine updates the state of the auction to indicate that it is

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1190 of 1657

closed. In one embodiment, the auction participant program may also broadcast a suppress message prior to broadcasting the gone message. When a participant receives a suppress message, it can no longer submit a bid at that auction. If the participant who broadcasted the suppress message receives no bid message from another participant for certain period of time, it then broadcasts the gone message. If, however, a participant after receiving the suppress message does not receive a gone message within the certain period of time, it can assume that the auction is still open.

Figure 14 is a flow diagram illustrating an auction agent in one embodiment. The auction agent is a program that allows a participant to specify a maximum bid that they want a place for an item. The auction agent will automatically monitor the auction and place bids on behalf of the participant up to the maximum bid. The auction agent may use various techniques to disguise from the other participants that it is an automated agent. For example, the auction agent may delay the placing of a new bid when being outbid. The delay may be a randomly selected time period or specified in rules provided by the participant. In addition, the auction agent may wait until it receives a going message to place the new bid. In block 1401, the routine retrieves the current high bid from the auction database. In decision block 1402, if the current high bid is already greater than the maximum bid that is authorized for this agent, then the routine continues at block 1403, else the routine continues at block 1404. In block 1403, the routine notifies the participant that the participant has been outbid at the auction and then returns. In block 1404, the routine submits a bid that is the current bid plus the minimum bid increment. The submitted bid is broadcast on the broadcast channel. In block 1405, the routine waits for a message to be broadcast for the auction. This auction participant program will also broadcast going and gone messages as appropriate. In decision block 1406, if the message indicates that a new bid has been placed that is higher than the current bid, then the routine continues at block 1407, else the routine continues at block 1408 because this auction participant program has broadcast a gone message. In block 1407, the routine optionally delays and then loops to block 1402 to submit a new bid. In

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1191 of 1657

block 1408, the routine notifies the participant that the auction has been won and then returns.

Based on the description, it will be appreciated that although specific embodiments of the invention have been described for purposes of illustration, various modifications may be made without deviating from the spirit and scope of the invention. Accordingly, the invention is not limited except by the appended claims.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1192 of 1657

CLAIMS

1. An auction system comprising:

an auction listing server through which an auction for an item can be defined and information about defined auctions can be retrieved; and

an auction participant program that executes at each participant's computer, that receives bid messages that are broadcast on a broadcast channel, that broadcasts bid messages when the participant submits a bid, and that, when the participant has submitted the highest bid, determines when the auction ends.

2. The auction system of claim 1 wherein a participant computer is connected to at least three other participant computers.

3. The auction system of claim 1 including:

an auction monitor computer that monitors the state of the auction based on messages sent on the broadcast channel.

4. The auction system of claim 1 wherein multiple auctions are conducted simultaneously on the broadcast channel.

5. The auction system of claim 1 wherein each auction is conducted on a separate broadcast channel.

6. The auction system of claim 1 wherein each broadcast bid message includes a randomly generated number for use in awarding bids when two or more participants bid the same amount.

7. The auction system of claim 1 wherein the auction participant program broadcasts a going message when it decides that the auction is about to end.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1193 of 1657

8. The auction system of claim 7 wherein the going message is sent by the auction participant program that has submitted the current high bid.

9. An auction system comprising a plurality of computer systems, each computer system capable of sending a message to each other computer system, each computer system including a component that receives bid messages from other computer systems, that sends bid messages to other computer systems, that sets a current high bid based on the bid messages, and that determines when to close an auction when the computer system has submitted the current high bid.

10. The auction system of claim 9 wherein the computer systems are connected via a broadcast channel.

11. The auction system of claim 9 wherein the computer systems are interconnected to form an m-regular graph, where m is 4 or larger.

12. The auction system of claim 9 wherein a computer system determines to close an auction after it has sent a going message to the other computer systems.

13. The auction system of claim 12 wherein the determination is made a certain time period after sending the going message.

14. The auction system of claim 12 including sending a gone message when the computer system determines to close an auction.

15. The auction system of claim 9 wherein each computer system resolves bids of equal amount based on a random number included with a bid message.

16. A method in a computer system for resolving equal bids at an auction, the method comprising:

receiving a first bid and a first tiebreaker;

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1194 of 1657

indicating that the first bid is the current high bid at the auction;

receiving a second bid and a second tiebreaker; and

when the first bid and the second bid are equal,

comparing the first tiebreaker to the second tiebreaker; and

indicating that the second bid is the current high bid at the auction based on the comparison.

17. The method of claim 16 wherein the first tiebreaker and the second tiebreaker are randomly generated numbers.

18. The method of claim 16 wherein the computer system is connected to a broadcast channel.

19. The method of claim 16 wherein the auction has no central auction server.

20. The method of claim 16 wherein each participant in the auction is a peer computer system.

21. The method of claim 16 wherein the second bid is indicated as the current high bid when the second tiebreaker is larger than the first tiebreaker.

22. The method of claim 16 wherein computer systems participating in the auction are interconnected in a m-regular graph, where m is 4 or greater.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1195 of 1657

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1196 of 1657

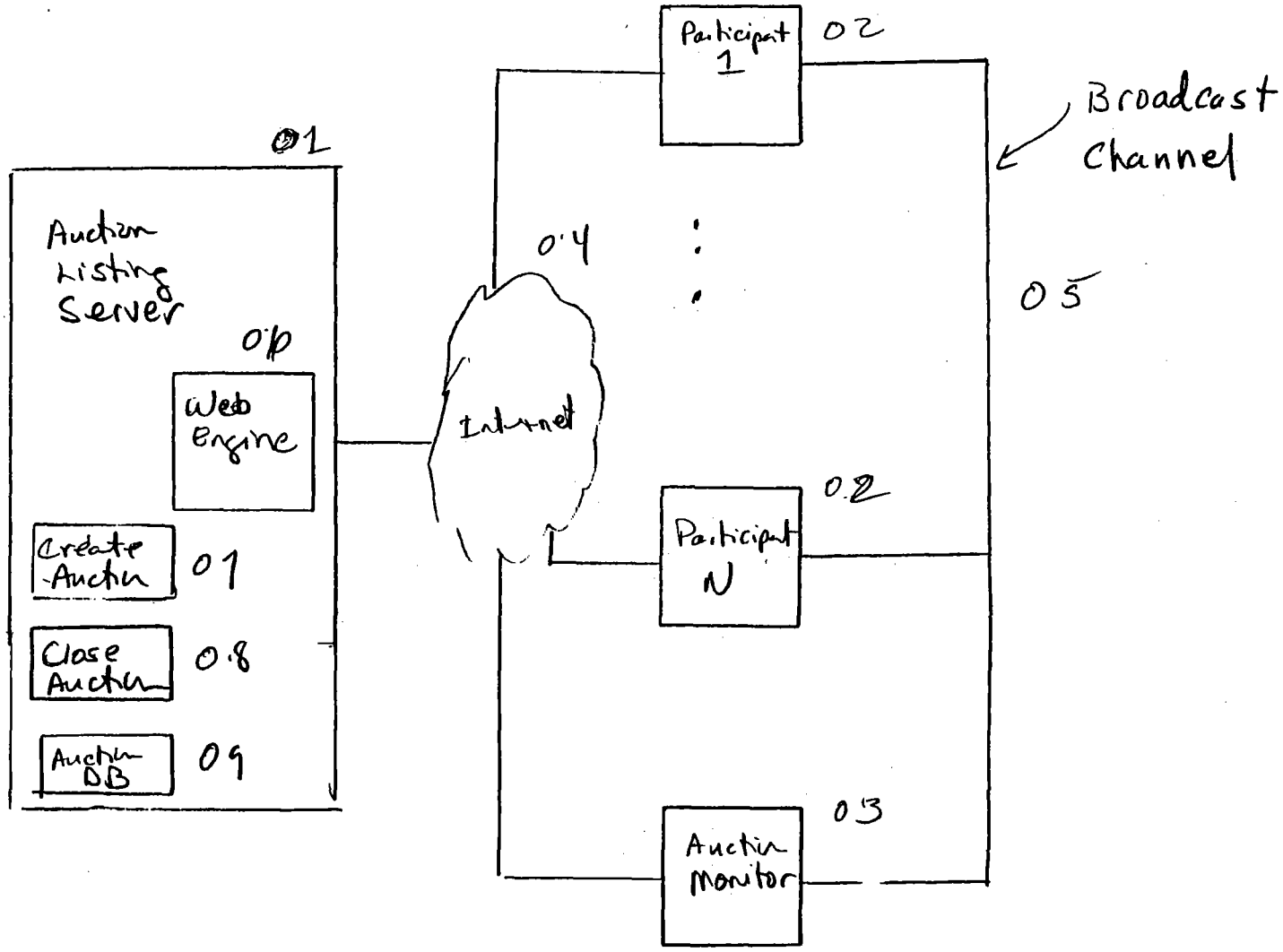


Fig 1

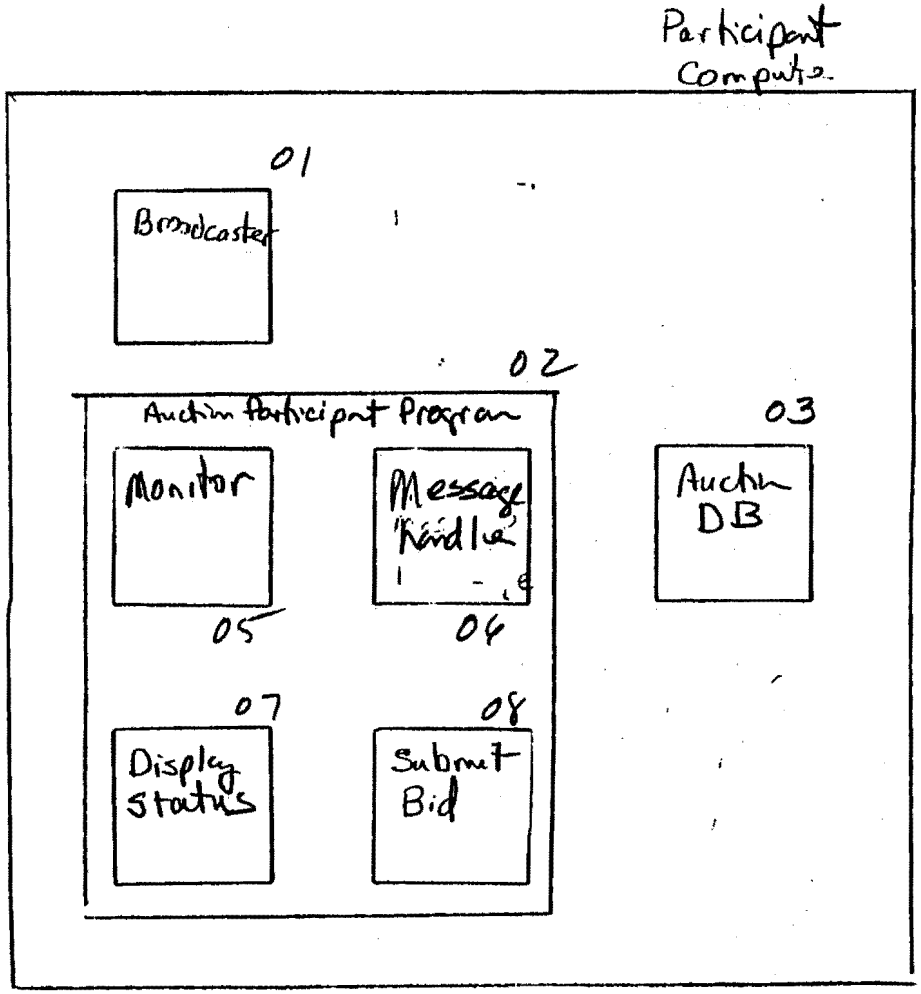


Fig 2

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1197 of 1657

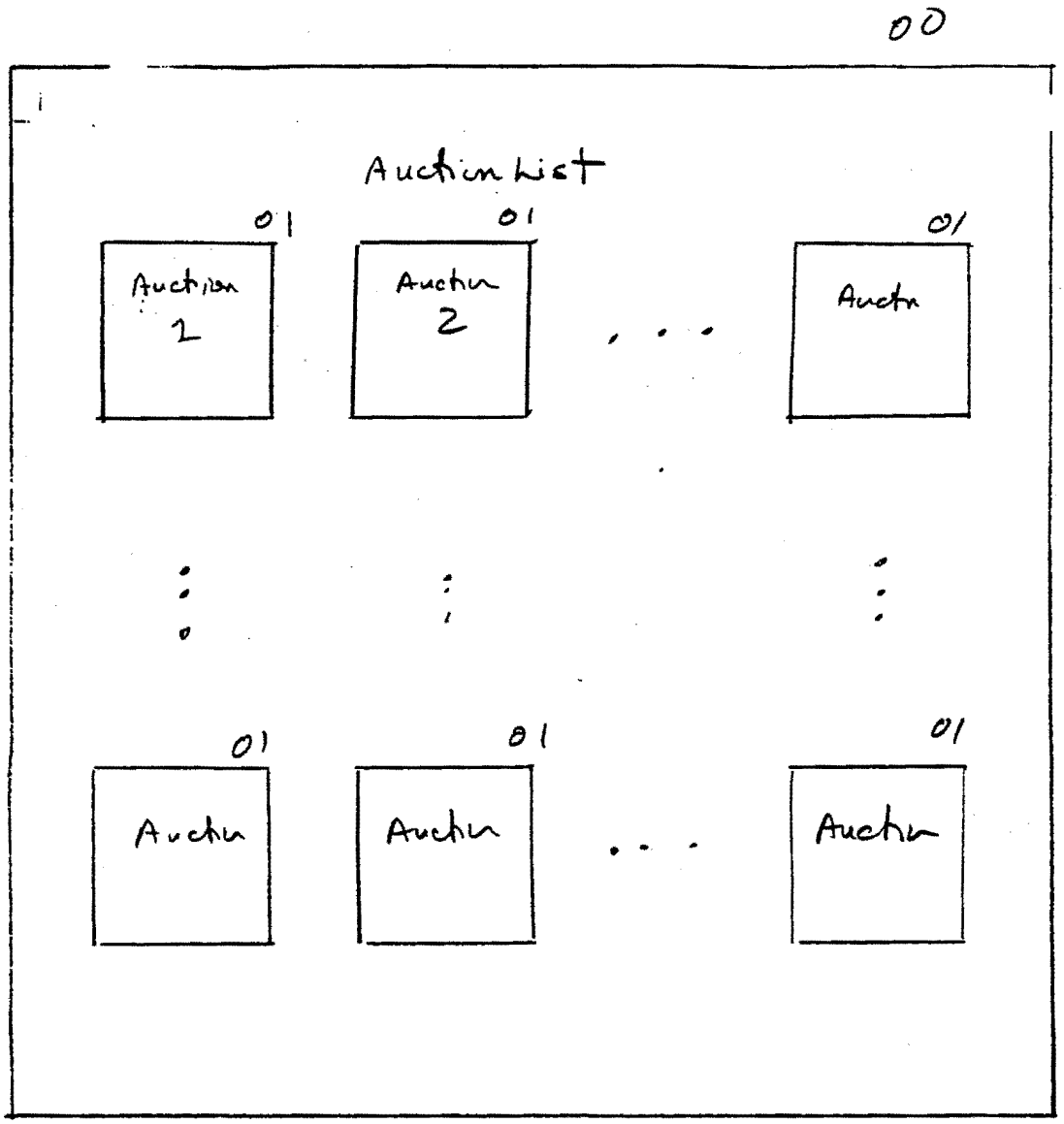

00
Auction List
01
Auction 1
01
Auction 2
01
Auctn
01
Auctn
01
Auctn
01
Auctn

Fig 3

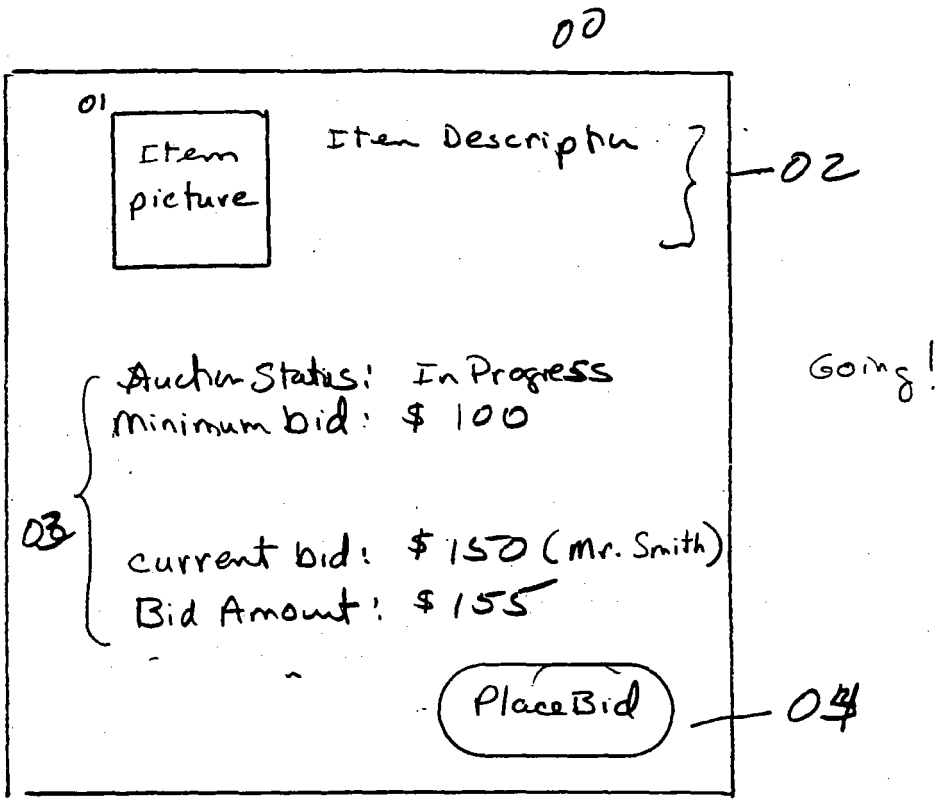

00
01
Item picture
Item Description
02
Auction Status: In Progress
Minimum bid: $ 100
03
current bid: $ 150 (Mr. Smith)
Bid Amount: $ 155
Going!
Place Bid
04

Fig 4

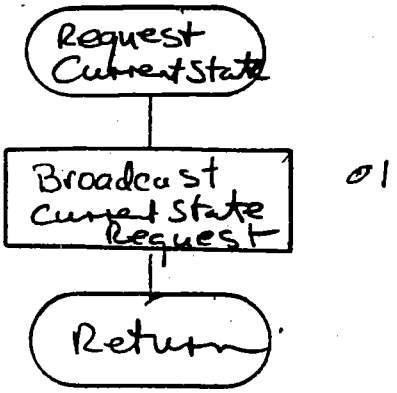

Request Current State
Broadcast Current State Request
Return
01

Fig 5

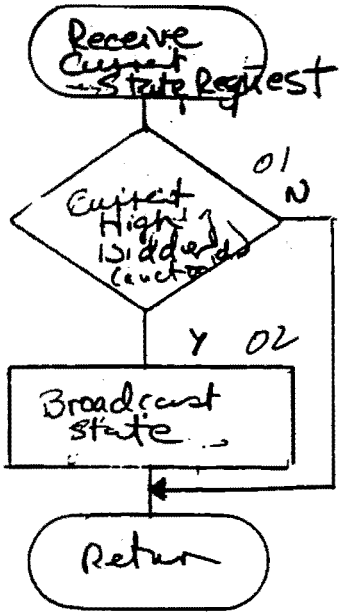
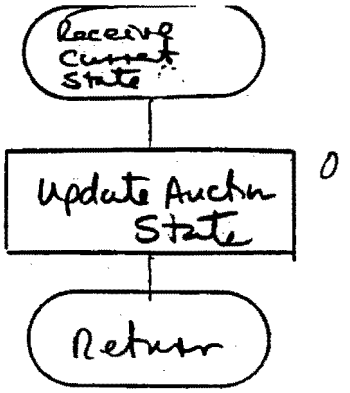

Receive Current State Request
01
N
Y 02
Broadcast state
Return

Fig 6



F.g 7

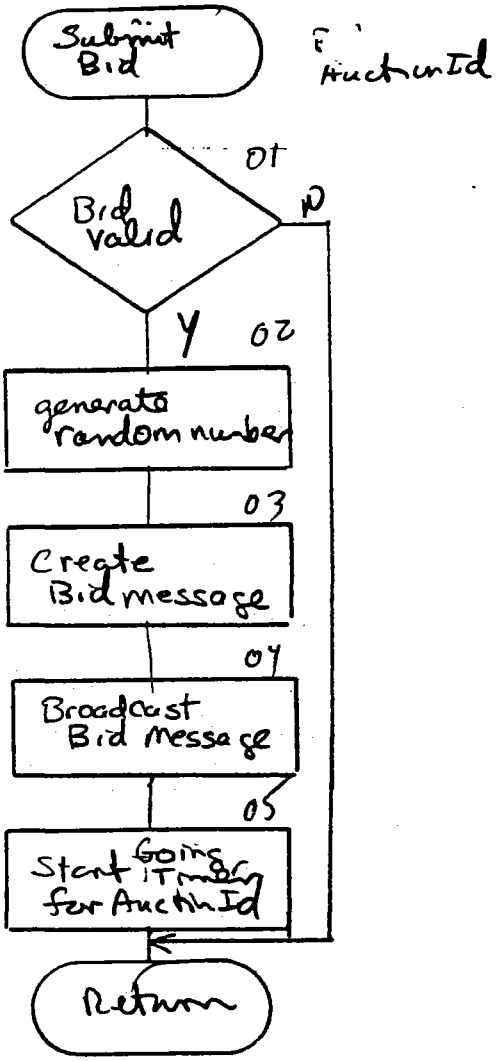

Submit Bid
AuctionId
01
Bid valid
N
Y
02
generate random number
03
Create Bid message
04
Broadcast Bid message
05
Start Going Timer for AuctionId
Return

Fig 8

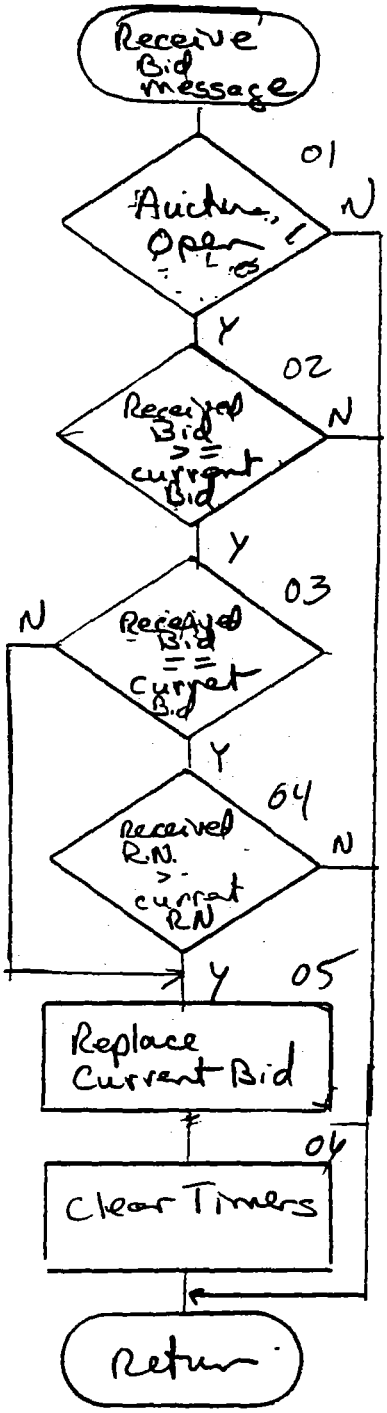

Receive Bid message
01
Auction Open
N
Y
02
Received Bid >= current Bid
N
Y
03
Received Bid == Current Bid
N
Y
04
Received R.N. > current R.N
N
Y
05
Replace Current Bid
06
Clear Timers
Return

Fig 9

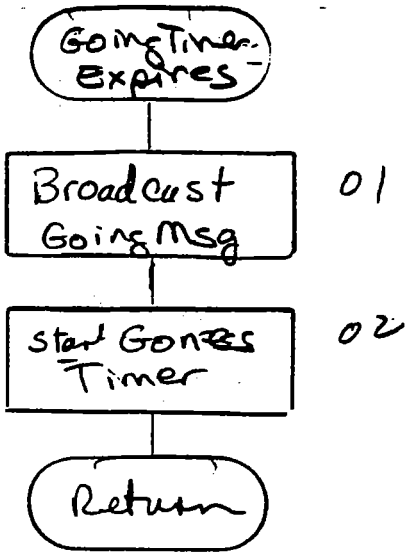

Going Timer Expires
Broadcast Going Msg
01
Start Going Timer
02
Return

Fig 10

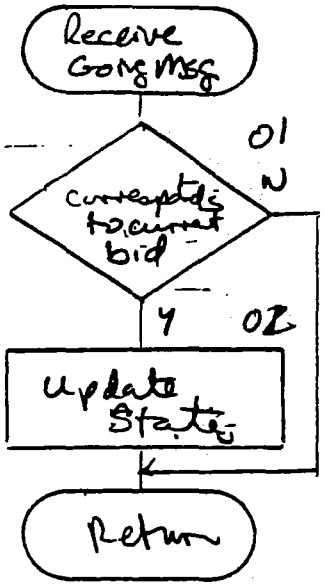


Fig 11

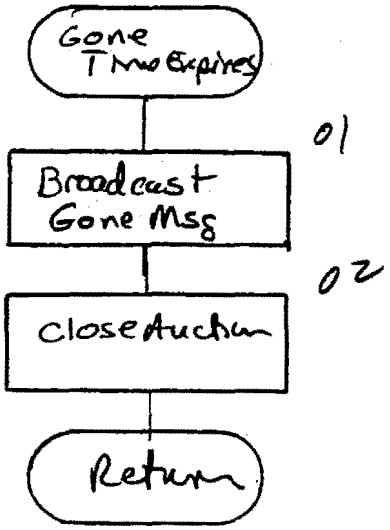

Gone Timer Expires
01
Broadcast Gone Msg
02
closeAuction
Return

Fig 12

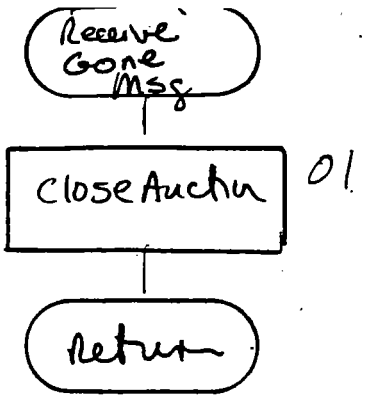


Fig 13

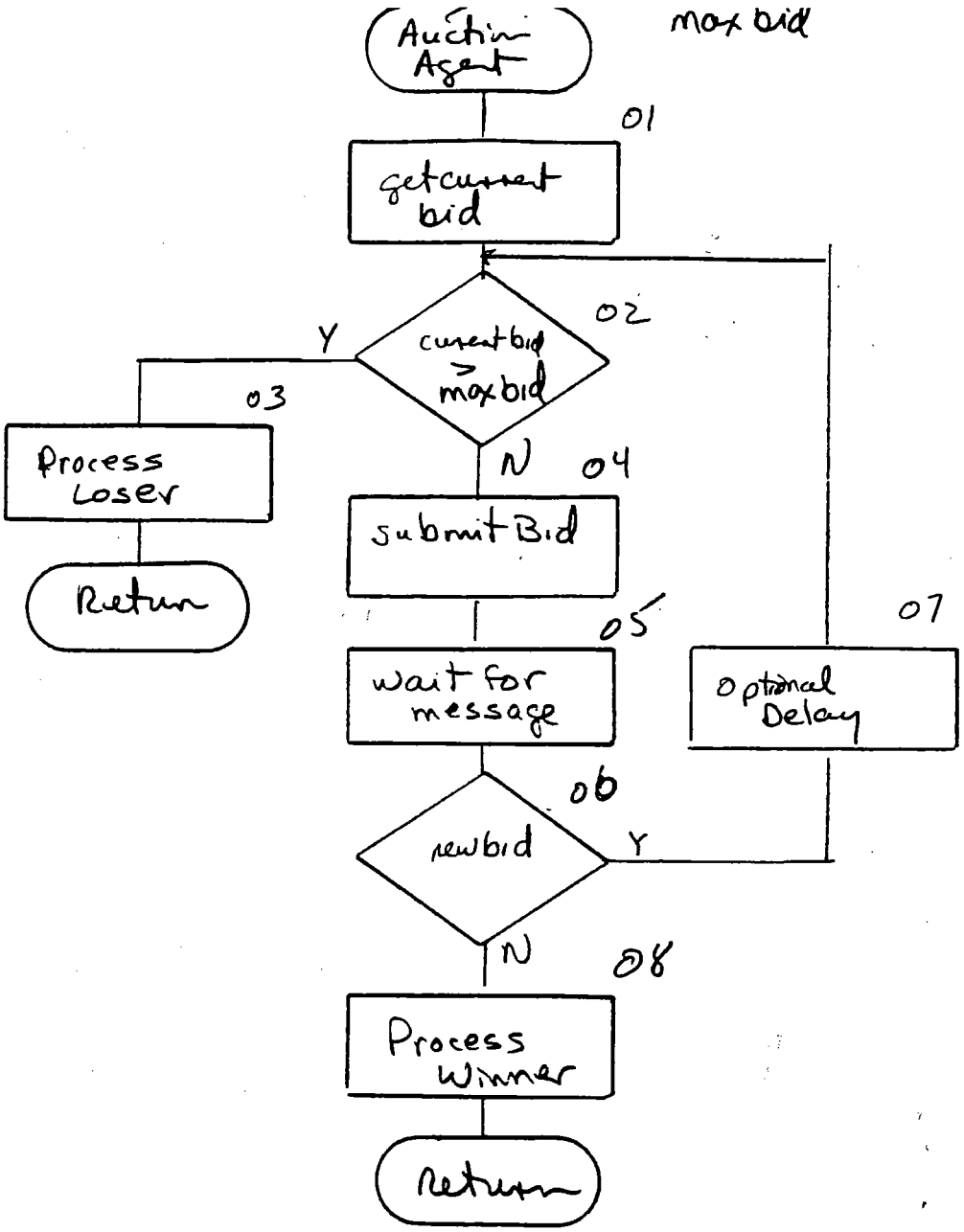

Auction Agent
max bid
01
get current bid
02
current bid > max bid
Y
03
Process Loser
Return
N
04
submit Bid
05
wait for message
07
optional Delay
06
new bid
Y
N
08
Process Winner
Return

Fig 14

CONTACTING A BROADCAST CHANNEL

CROSS-REFERENCE TO RELATED APPLICATIONS

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

TECHNICAL FIELD

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

BACKGROUND

There are a wide variety of computer network communications techniques such as point-to-point network protocols, client/server middleware, multicasting network

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1210 of 1657

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

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

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

The multicasting network protocols allow the sending of broadcast messages to multiple recipients of a network. The current implementations of such multicasting network

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1211 of 1657

protocols tend to place an unacceptable overhead on the underlying network. For example, UDP multicasting would swamp the Internet when trying to locate all possible participants. IP multicasting has other problems that include needing special-purpose infrastructure (*e.g.*, routers) to support the sharing of information efficiently.

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

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

BRIEF DESCRIPTION OF THE DRAWINGS

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

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

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

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

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1212 of 1657

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

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

Figure 5C illustrates the neighbors with empty ports condition.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1213 of 1657

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1214 of 1657

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

DETAILED DESCRIPTION

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1215 of 1657

divide the graph into disjoint sub-graphs, that is two separate broadcast channels. This property is referred to as being 4-connected. Thus, the graph is both 4-regular and 4-connected.

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

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

Composing the Graph

To connect to the broadcast channel, the computer seeking the connection first locates a computer that is currently fully connected to the broadcast channel and then

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1216 of 1657

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

Since the broadcast channel is a 4-regular graph, each of the identified computers is already connected to four computers. Thus, some connections between computers need to be broken so that the seeking computer can connect to four computers. In one embodiment, the broadcast technique identifies two pairs of computers that are currently connected to each other. Each of these pairs of computers breaks the connection between them, and then each of the four computers (two from each pair) connects to the seeking computer. Figures 3A and 3B illustrate the process of a new computer Z connecting to the broadcast channel. Figure 3A illustrates the broadcast channel before computer Z is connected. The pairs of computers B and E and computers C and D are the two pairs that are identified as the neighbors for the new computer Z. The connections between each of these pairs is broken, and a connection between computer Z and each of computers B, C, D, and E

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1217 of 1657

is established as indicated by Figure 3B. The process of breaking the connection between two neighbors and reconnecting each of the former neighbors to another computer is referred to as "edge pinning" as the edge between two nodes may be considered to be stretched and pinned to a new node.

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1218 of 1657

seeking computer actually communicates through that transfer-to port, which is the external port. The call is transferred so that other computers can place calls to that computer via the call-in port. The seeking computer then communicates via that external port to request the portal computer to assist in connecting the seeking computer to the broadcast channel. The seeking computer could identify the call-in port number of a portal computer by successively dialing each port in port number order. As discussed below in detail, the broadcast technique uses a hashing algorithm to select the port number order, which may result in improved performance.

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1219 of 1657

Broadcasting Through the Graph

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

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

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

When the broadcast channel is in a steady state (*i.e.*, no computers connecting or disconnecting from the broadcast channel), out-of-order messages are not a problem because each computer will eventually receive both messages and can queue messages until all earlier ordered messages are received. If, however, the broadcast channel is not in a

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1220 of 1657

steady state, then problems can occur. In particular, a computer may connect to the broadcast channel after the second message has already been received and forwarded on by its new neighbors. When a new neighbor eventually receives the first message, it sends the message to the newly connected computer. Thus, the newly connected computer will receive the first message, but will not receive the second message. If the newly connected computer needs to process the messages in order, it would wait indefinitely for the second message.

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1221 of 1657

same originating computer through another neighbor. If the second set of messages contains a message that is ordered earlier than the messages of the first set received, then the newly connected computer may ignore that earlier ordered message if the computer already processed those later ordered messages.

Decomposing the Graph

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1222 of 1657

computer that is also short a connection receives the connection request, it communicates with the requesting computer through its external port to establish a connection between the two computers. Figure 5B illustrates the disconnecting of a computer from the broadcast channel in an unplanned manner. In this illustration, computer H has disconnected in an unplanned manner. When each of its neighbors, computers A, E, F, and I, recognizes the disconnection, each neighbor broadcasts a port connection request indicating that it needs to fill an empty port. As shown by the dashed lines, computers F and I and computers A and E respond to each other's requests and establish a connection.

It is possible that a planned or unplanned disconnection may result in two neighbors each having an empty internal port. In such a case, since they are neighbors, they are already connected and cannot fill their empty ports by connecting to each other. Such a condition is referred to as the "neighbors with empty ports" condition. Each neighbor broadcasts a port connection request when it detects that it has an empty port as described above. When a neighbor receives the port connection request from the other neighbor, it will recognize the condition that its neighbor also has an empty port. Such a condition may also occur when the broadcast channel is in the small regime. The condition can only be corrected when in the large regime. When in the small regime, each computer will have less than four neighbors. To detect this condition in the large regime, which would be a problem if not repaired, the first neighbor to receive the port connection request recognizes the condition and sends a condition check message to the other neighbor. The condition check message includes a list of the neighbors of the sending computer. When the receiving computer receives the list, it compares the list to its own list of neighbors. If the lists are different, then this condition has occurred in the large regime and repair is needed. To repair this condition, the receiving computer will send a condition repair request to one of the neighbors of the sending computer which is not already a neighbor of the receiving computer. When the computer receives the condition repair request, it disconnects from one of its neighbors (other than the neighbor that is involved with the condition) and connects to the computer that sent the condition repair request. Thus, one of the original neighbors involved in the condition will have had a port filled. However, two computers are still in need of a connection, the other original neighbor and the computer that is now disconnected from the computer that received the condition repair request. Those two computers send out port connection requests. If those two computers are not neighbors, then they will connect to

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1223 of 1657

each other when they receive the requests. If, however, the two computers are neighbors, then they repeat the condition repair process until two non-neighbors are in need of connections.

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

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

Figures 5E and 5F further illustrate the neighbors with empty ports condition. Figure 5E illustrates the neighbors with empty ports condition in the small regime. In this

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1224 of 1657

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

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

Port Selection

As described above, the TCP/IP protocol designates ports above number 2056 as user ports. The broadcast technique uses five user port numbers on each computer: one external port and four internal ports. Generally, user ports cannot be statically allocated to an application program because other applications programs executing on the same computer may use conflicting port numbers. As a result, in one embodiment, the computers connected to the broadcast channel dynamically allocate their port numbers. Each computer could simply try to locate the lowest number unused port on that computer and use that port as the call-in port. A seeking computer, however, does not know in advance the call-in port number of the portal computers when the port numbers are dynamically allocated. Thus, a seeking computer needs to dial ports of a portal computer starting with the lowest port number when locating the call-in port of a portal computer. If the portal computer is

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1225 of 1657

connected to (or attempting to connect to) the broadcast channel, then the seeking computer would eventually find the call-in port. If the portal computer is not connected, then the seeking computer would eventually dial every user port. In addition, if each application program on a computer tried to allocate low-ordered port numbers, then a portal computer may end up with a high-numbered port for its call-in port because many of the low-ordered port numbers would be used by other application programs. Since the dialing of a port is a relatively slow process, it would take the seeking computer a long time to locate the call-in port of a portal computer. To minimize this time, the broadcast technique uses a port ordering algorithm to identify the port number order that a portal computer should use when finding an available port for its call-in port. In one embodiment, the broadcast technique uses a hashing algorithm to identify the port order. The algorithm preferably distributes the ordering of the port numbers randomly through out the user port number space and only selects each port number once. In addition, every time the algorithm is executed on any computer for a given channel type and channel instance, it generates the same port ordering. As described below, it is possible for a computer to be connected to multiple broadcast channels that are uniquely identified by channel type and channel instance. The algorithm may be "seeded" with channel type and channel instance in order to generate a unique ordering of port numbers for each broadcast channel. Thus, a seeking computer will dial the ports of a portal computer in the same order as the portal computer used when allocating its call-in port.

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1226 of 1657

possible that eight seeking computers could be simultaneously dialing ports in different sequences which would reduce the chances of dialing a busy port.

Locating a Portal Computer

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1227 of 1657

then the one seeking computer can share that it has searched to a depth of eight with another seeking computer. If that other seeking computer has searched to a depth of, for example, only four, it can skip searching through depths five through eight and that other seeking computer can advance its searching to a depth of nine.

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

Identifying Neighbors for a Seeking Computer

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

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1228 of 1657

traveled far enough will detect this condition of already being a neighbor and send the message to a randomly selected neighbor.

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

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

External Data Representation

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1229 of 1657

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

M-Regular

In the embodiment described above, each fully connected computer has four internal connections. The broadcast technique can be used with other numbers of internal connections. For example, each computer could have 6, 8, or any even number of internal connections. As the number of internal connections increase, the diameter of the broadcast channel tends to decrease, and thus propagation time for a message tends to decrease. The time that it takes to connect a seeking computer to the broadcast channel may, however, increase as the number of internal connections increases. When the number of internal connectors is even, then the broadcast channel can be maintained as m-regular and m-connected (in the steady state). If the number of internal connections is odd, then when the broadcast channel has an odd number of computers connected, one of the computers will have less than that odd number of internal connections. In such a situation, the broadcast network is neither m-regular nor m-connected. When the next computer connects to the broadcast channel, it can again become m-regular and m-connected. Thus, with an odd number of internal connections, the broadcast channel toggles between being and not being m-regular and m-connected.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1230 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1231 of 1657

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.

The following tables list messages sent by the broadcaster components.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1232 of 1657

EXTERNAL MESSAGES

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

INTERNAL MESSAGES

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1233 of 1657

	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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1234 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1235 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1236 of 1657

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

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 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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1237 of 1657

returns. In block 1203, the routine receives the external message from the external port. In decision block 1204, if the type of the message indicates that a seeking process is calling (*i.e.*, seeking_connection_call), then the routine continues at block 1205, else the routine returns. In block 1205, the routine sends an external message (*i.e.*, seeking_connection_resp) to the other seeking process indicating that this process is also is seeking a connection. In decision block 1206, if the sending of the external message is successful, then the routine continues at block 1207, else the routine returns. In block 1207, the routine adds the other seeking process to a list of fellow seeking processes and then returns. This list may be used if this process can find no process that is fully connected to the broadcast channel. In which case, this process may check to see if any fellow seeking process were successful in connecting to the broadcast channel. For example, a fellow seeking process may become the first process fully connected to the broadcast channel.

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1238 of 1657

request call routine in block 1406, else the routine continues at block 1407. In decision block 1407, if the message type is edge proposal call (*i.e.*, edge_proposal_call), then the routine invokes the handle edge proposal call routine in block 1408, else the routine continues at block 1409. In decision block 1409, if the message type is port connect call (*i.e.*, port_connect_call), then the routine invokes the handle port connection call routine in block 1410, else the routine continues at block 1411. In decision block 1411, if the message type is a connected statement (*i.e.*, connected_stmt), the routine invokes the handle connected statement in block 1112, else the routine continues at block 1212. In decision block 1412, if the message type is a condition repair statement (*i.e.*, condition_repair_stmt), then the routine invokes the handle condition repair routine in block 1413, else the routine loops to block 1414 to process the next message. After each handling routine is invoked, the routine loops to block 1414. In block 1414, the routine hangs up on the external port and continues at block 1401 to receive the next message.

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1239 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1240 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1241 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1242 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1243 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1244 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1245 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1246 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1247 of 1657

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

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1248 of 1657

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.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1249 of 1657

CLAIMS

1. A method in a computer for locating a computer through which to connect to a network, the method comprising:

providing an identification of a portal computer, the portal computer having communications ports with one of the communications ports being enabled for communications when the portal computer is in a state to coordinate the connection of a seeking computer to the network;

repeatedly selecting a communications port of the portal computer and attempting to communicate with the selected communications port until the communications with the selected communications port is successful; and

using the selected communications port to request that the portal computer coordinate the connecting of the computer to the network.

2. The method of claim 1 wherein the communications ports are selected in an order that is the same as used by the portal computer when it selected a communications port.

3. The method of claim 1 wherein the communications ports are selected based on a hashing algorithm ordering.

4. The method of claim 3 wherein the hashing algorithm ordering provides an ordering in which each communications port is selected without re-selecting a communications port.

5. The method of claim 3 wherein the hashing algorithm ordering is modified to reduce conflicts with other seeking computer that use the same hashing algorithm.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1250 of 1657

6. The method of claim 5 wherein a number of the first communications ports ordered by the hashing algorithm are reordered.

7. The method of claim 1 wherein the identification of a plurality of portal computers is provided and when a communications port is selected, attempting to communicate with each of the identified portal computers through the selected communications port before selecting the next communications port.

8. The method of claim 1 wherein the communications ports are TCP/IP ports.

9. A method in a computer system for locating a communications port, each communications port having a port number, the method comprising:

providing an ordering of the communications ports that is not port number sequential; and

until a communications port through which a connection can be established is found,

selecting the next communications port in the provided order; and

determining whether a connection can be established through the selected communications port.

10. The method of claim 9 wherein the ordering is provided by a hashing algorithm.

11. The method of claim 9 wherein the communications ports are TCP/IP port.

12. The method of claim 9 wherein the communications ports are ports of another computer.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1251 of 1657

13. The method of claim 12 wherein the other computer uses the same provided ordering of communications ports when selecting its communications port.

14. The method of claim 9 including reordering some portions of the provided ordering of the communications ports.

15. A data structure transmitted on a communications channel comprising a sequence of messages, each message in the sequence identifying a communications port of a computer system, whereby a hashing algorithm is used to order the identification of the communications ports in the sequence of messages.

16. The data structure of claim 15 wherein each message requests a connection to a receiving computer via the identified communications port.

17. The data structure of claim 15 wherein the messages are TCP/IP messages.

18. The data structure of claim 15 wherein the communications channel is the Internet.

19. The data structure of claim 15 wherein the sequence of messages is used to locate a portal computer through which a sending computer can request a connection to a broadcast channel.

20. A computer network having a plurality of participants, each participant having connections to neighbor participants, wherein a participant locates a communications port of a portal computer by repeatedly selecting a communications port of the portal computer and attempting to communicate with the selected communications port until the communications with the selected communications port is successful.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1252 of 1657

21. The computer network of claim 20 wherein the selecting of the communications ports is ordered according to a function.

22. The computer network of claim 21 wherein the portal computer uses the same function to order its selection of a communications port.

23. The computer network of claim 20 wherein an originating participant sends data to the other participants by sending the data through each of its connections to its neighbor participants, wherein when each participant sends data that it receives from a neighbor participant to its other neighbor participants.

24. The computer network of claim 20 wherein each participant is connected to 4 other participants.

25. The computer network of claim 20 wherein each participant is connected to an even number of other participants.

26 The computer network of claim 20 wherein the network is m-regular, where m is the number of neighbor participants of each participant.

27. The computer network of claim 26 wherein the network is m-connected, where m is the number of neighbor participants of each participant.

28. The computer network of claim 20 wherein the network is m-regular and m-connected, where m is the number of neighbor participants of each participant.

29. The computer network of claim 20 wherein all the participants are peers.

30. The computer network of claim 20 wherein the connections are peer-to-peer connections.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1253 of 1657

31. The computer network of claim 20 wherein the connections between neighbor computers are point-to-point.

32. The computer network of claim 20 wherein the connections are TCP/IP connections.

33. A component in a computer system for locating a communications port of a portal computer, comprising:

means for identifying the portal computer, the portal computer having a dynamically selected communications port for communicating with other computers; and

means for identifying the communications port of the identified portal computer by repeatedly trying to establish a connection with the identified portal computer through communications ports until a connection is successfully established.

34. The component of claim 33 wherein the communications ports are trying in an order that is the same as used by the portal computer when it dynamically selects a communications port.

35. The component of claim 33 wherein the communications ports are tried based on a hashing algorithm ordering.

36. The component of claim 35 wherein the hashing algorithm ordering provides an ordering in which each communications port is tried without re-trying a communications port.

37. The component of claim 35 wherein the hashing algorithm ordering is modified to reduce conflicts with other computers that use the same hashing algorithm.

38. The component of claim 37 wherein a number of the first communications ports ordered by the hashing algorithm are reordered.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1254 of 1657

39. The component of claim 33 including:

means for identifying a plurality of portal computers; and

means for trying to establish a connection with each of the identified portal computers through a certain communications port before trying the next communications port.

40. The component of claim 33 wherein the communications ports are TCP/IP ports.

IPR2016-00726 - ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR, Ex. 1002, Vol. 4, p. 1255 of 1657

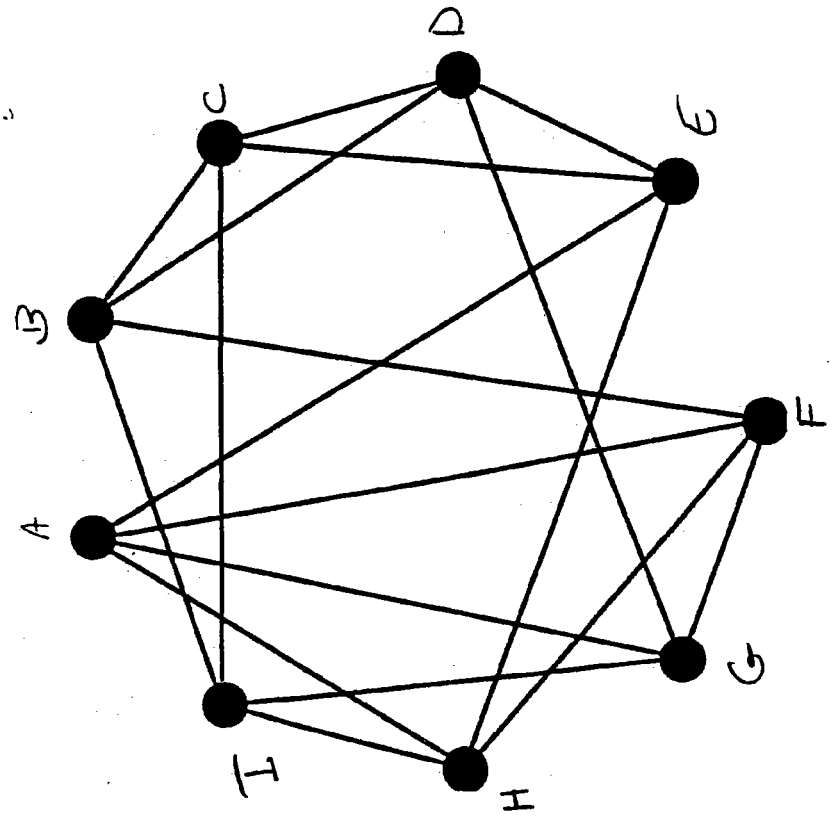


Fig 1