

CERTIFICATE OF SERVICE

The undersigned hereby certifies that a copy of this **REQUEST FOR REEXAMINATION UNDER 35 U.S.C. §§ 302-307 AND 37 C.F.R. § 1.510 FOR** U.S. Patent 5,822,523 together with all exhibits and attachments and supporting documentation on a CD, has been served via first class mail on June 11, 2010 upon the following:

DANIEL DEVITO
SKADDEN, ARPS, SLATE, MEAGHER & FLOM LLP
FOUR TIMES SQUARE
NEW YORK NY 10036

JORDAN ALTMAN
SHEARMAN & STERLING LLP
IP DOCKETING
599 LEXINGTON AVENUE
NEW YORK, NY 10022

RAJIV P. PATEL, ESQ.
FENWICK & WEST LLP
TWO PALO ALTO SQUARE
PALO ALTO, CA 94306

/Sonal Dash/
Sonal Dash

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

INFORMATION DISCLOSURE STATEMENT BY APPLICANT (Not for submission under 37 CFR 1.99)	Application Number		
	Filing Date		
	First Named Inventor	DANIEL J. SAMUEL	
	Art Unit		
	Examiner Name		
	Attorney Docket Number	8330.003	

U.S.PATENTS						
Examiner Initial*	Cite No	Patent Number	Kind Code ¹	Issue Date	Name of Patentee or Applicant of cited Document	Pages,Columns,Lines where Relevant Passages or Relevant Figures Appear
	1	5736982		1998-04-07	Suzuki et al.	
	2					
	3					
	4					
	5					

If you wish to add additional U.S. Patent citation information please click the Add button.

U.S.PATENT APPLICATION PUBLICATIONS						
Examiner Initial*	Cite No	Publication Number	Kind Code ¹	Publication Date	Name of Patentee or Applicant of cited Document	Pages,Columns,Lines where Relevant Passages or Relevant Figures Appear
	1					

If you wish to add additional U.S. Published Application citation information please click the Add button.

FOREIGN PATENT DOCUMENTS						
--------------------------	--	--	--	--	--	--

INFORMATION DISCLOSURE STATEMENT BY APPLICANT (Not for submission under 37 CFR 1.99)	Application Number		
	Filing Date		
	First Named Inventor	DANIEL J. SAMUEL	
	Art Unit		
	Examiner Name		
	Attorney Docket Number	8330.003	

Examiner Initial*	Cite No	Foreign Document Number ³	Country Code ² j	Kind Code ⁴	Publication Date	Name of Patentee or Applicant of cited Document	Pages, Columns, Lines where Relevant Passages or Relevant Figures Appear	T ⁵
	1							<input type="checkbox"/>

If you wish to add additional Foreign Patent Document citation information please click the Add button

NON-PATENT LITERATURE DOCUMENTS

Examiner Initials*	Cite No	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc), date, pages(s), volume-issue number(s), publisher, city and/or country where published.	T ⁵
	1	Server2.5pl4.tar.gz ("Server Code") and BRMH-1.7.tar.gz ("Client Code") (source code dated no later than August 1994) ("Netrek")	<input type="checkbox"/>
	2	J. OIKARINEN ET AL. RFC 1459, "Internet Relay Chat Protocol", published May 1993 ("IRC RFC").	<input type="checkbox"/>
	3	R. FRIEDMAN ET AL. "Packing Messages as a Tool for Boosting the Performance of Total Ordering Protocols", Dept. of Science of Cornell University, published July 7, 1995 ("Friedman").	<input type="checkbox"/>
	4	DANIEL J. VAN HOOK, JAMES O. CALVIN, MICHAEL K. NEWTON, and DAVID A. FUSCO, "An Approach to DIS Scaleability," 11th DIS Workshop, 26-30 Sept. 1994 ("Van Hook").	<input type="checkbox"/>
	5	IEEE 1278-1993 "IEEE Standard for Information Technology- Protocols for Distributed Interactive Simulation Applications", approved March 18, 1993, and published in 1993 ("DIS")	<input type="checkbox"/>
	6	T. A. FUNKHOUSER, "RING: A Client-Server System for Multi-User Virtual Environments," Association of Computing Machinery, 1995 Symposium on Interactive 3D Graphics, Monterey CA, April 9-12, 1995 ("RING").	<input type="checkbox"/>
	7	ANDY MCFADDEN, "The History of Netrek", published January 1, 1994 ("McFadden").	<input type="checkbox"/>

INFORMATION DISCLOSURE STATEMENT BY APPLICANT (Not for submission under 37 CFR 1.99)	Application Number		
	Filing Date		
	First Named Inventor	DANIEL J. SAMUEL	
	Art Unit		
	Examiner Name		
	Attorney Docket Number	8330.003	

	8	MICHAEL R. MACEDONIA, "Exploiting Reality with Multicast Groups", published September 1995 ("Macedonia")	<input type="checkbox"/>
--	---	--	--------------------------

If you wish to add additional non-patent literature document citation information please click the Add button

EXAMINER SIGNATURE

Examiner Signature		Date Considered	
--------------------	--	-----------------	--

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through a citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

¹ See Kind Codes of USPTO Patent Documents at www.USPTO.GOV or MPEP 901.04. ² Enter office that issued the document, by the two-letter code (WIPO Standard ST.3). ³ For Japanese patent documents, the indication of the year of the reign of the Emperor must precede the serial number of the patent document. ⁴ Kind of document by the appropriate symbols as indicated on the document under WIPO Standard ST.16 if possible. ⁵ Applicant is to place a check mark here if English language translation is attached.

PAT-A



US005822523A

United States Patent [19]
Rothschild et al.

[11] **Patent Number:** **5,822,523**
[45] **Date of Patent:** **Oct. 13, 1998**

[54] **SERVER-GROUP MESSAGING SYSTEM FOR INTERACTIVE APPLICATIONS**

[75] **Inventors:** **Jeffrey J. Rothschild; Marc P. Kwiatkowski**, both of Los Gatos; **Daniel J. Samuel**, Sunnyvale, all of Calif.

[73] **Assignee:** **Mpath Interactive, Inc.**, Mountain View, Calif.

[21] **Appl. No.:** **595,323**

[22] **Filed:** **Feb. 1, 1996**

[51] **Int. Cl.⁶** **H04H 1/02**

[52] **U.S. Cl.** **395/200.17; 395/200.1; 395/200.09**

[58] **Field of Search** **395/200.1, 200.01, 395/200.09, 200.17, 200.05, 793; 370/85.13, 60**

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,470,954	9/1984	Cotton et al.	370/60
5,079,767	1/1992	Perlman	370/94.3
5,150,464	9/1992	Sidhu et al.	395/200.01
5,309,433	5/1994	Cidon et al.	370/60
5,309,437	5/1994	Perlman et al.	370/85.13
5,329,619	7/1994	Pa�e et al.	395/200.01
5,361,256	11/1994	Doeringer et al.	370/60
5,475,819	12/1995	Miller et al.	395/200.01
5,517,494	5/1996	Green	370/60

FOREIGN PATENT DOCUMENTS

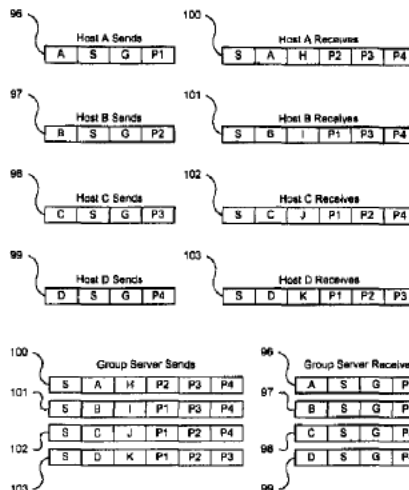
0637142	1/1995	European Pat. Off.
WO 95/10908	4/1995	WIPO
WO 95/10911	4/1995	WIPO

Primary Examiner—William M. Treat
Assistant Examiner—Zarni Maung
Attorney, Agent, or Firm—H. C. Chan; Wilson Sonsini Goodrich & Rosati

[57] **ABSTRACT**

A method for deploying interactive applications over a network containing host computers and group messaging servers is disclosed. The method operates in a conventional unicast network architecture comprised of conventional network links and unicast gateways and routers. The hosts send messages containing destination group addresses by unicast to the group messaging servers. The group addresses select message groups maintained by the group messaging servers. For each message group, the group messaging servers also maintain a list of all of the hosts that are members of the particular group. In its most simple implementation, the method consists of the group server receiving a message from a host containing a destination group address. Using the group address, the group messaging server then selects a message group which lists all of the host members of the group which are the targets of messages to the group. The group messaging server then forwards the message to each of the target hosts. In an interactive application, many messages will be arriving at the group server close to one another in time. Rather than simply forward each message to its targeted hosts, the group messaging server aggregates the contents of each of messages received during a specified time period and then sends an aggregated message to the targeted hosts. The time period can be defined in a number of ways. This method reduces the message traffic between hosts in a networked interactive application and contributes to reducing the latency in the communications between the hosts.

6 Claims, 11 Drawing Sheets



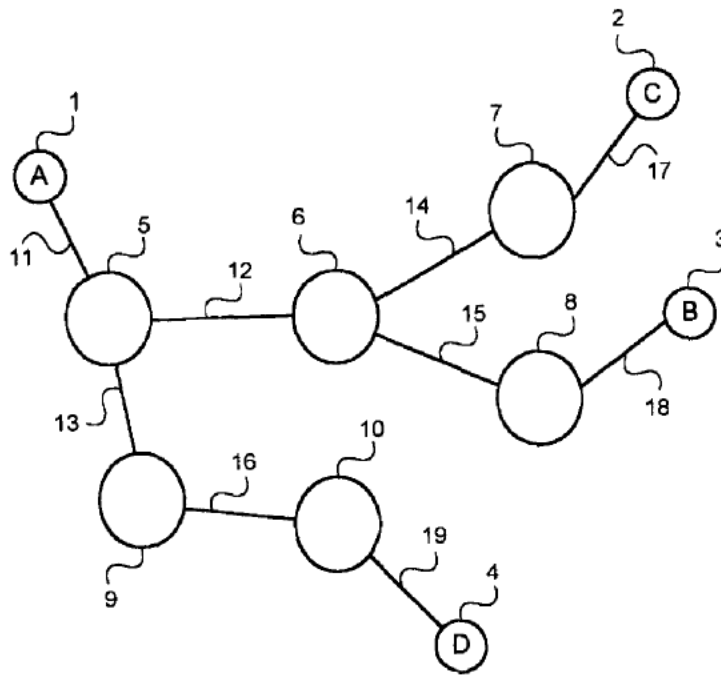


Figure 1
Prior Art

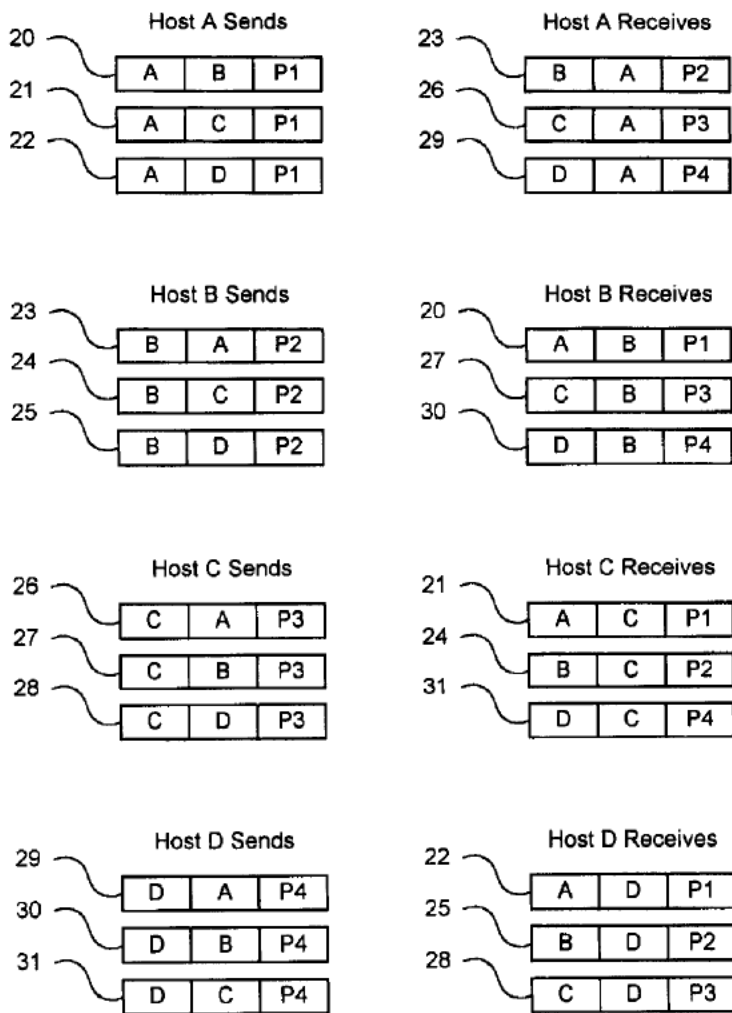


Figure 2
Prior Art

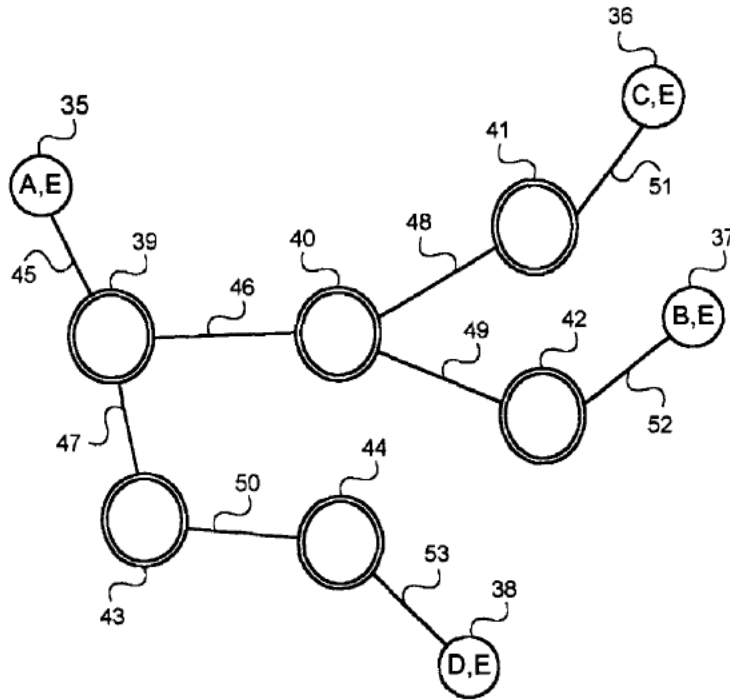


Figure 3
Prior Art

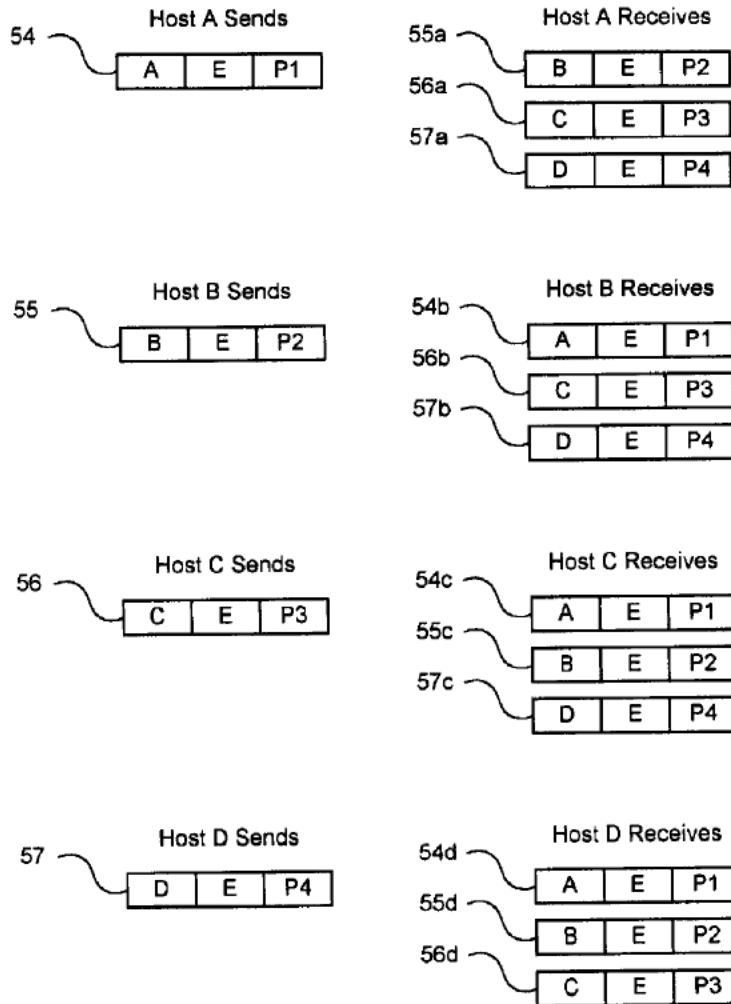


Figure 4
Prior Art

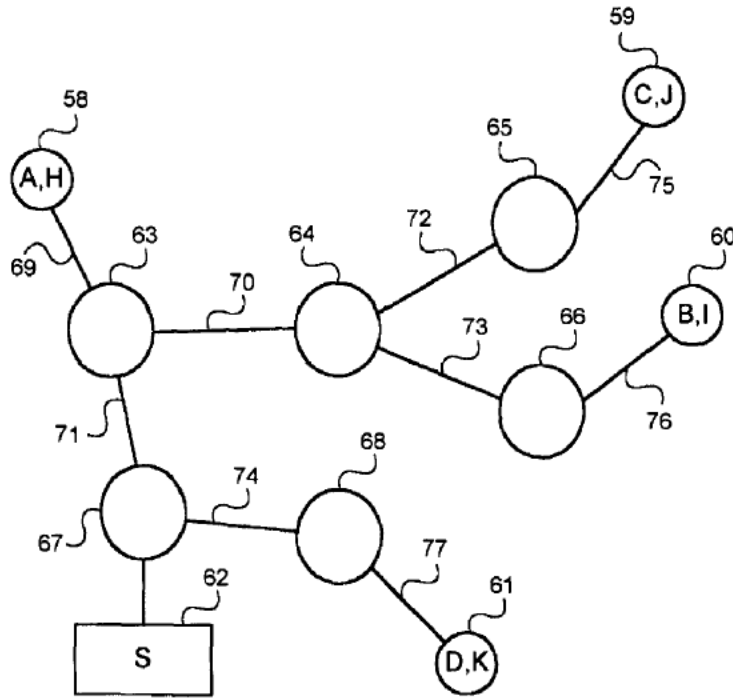


Figure 5

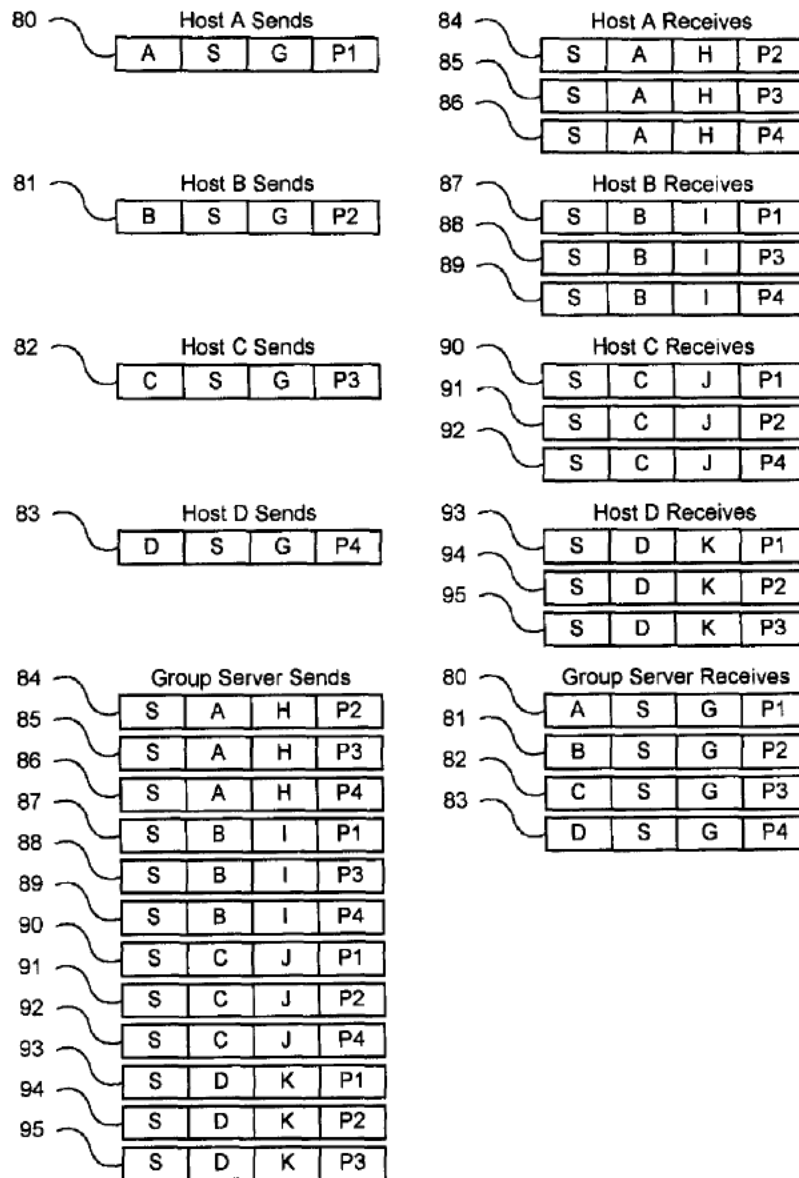


Figure 6

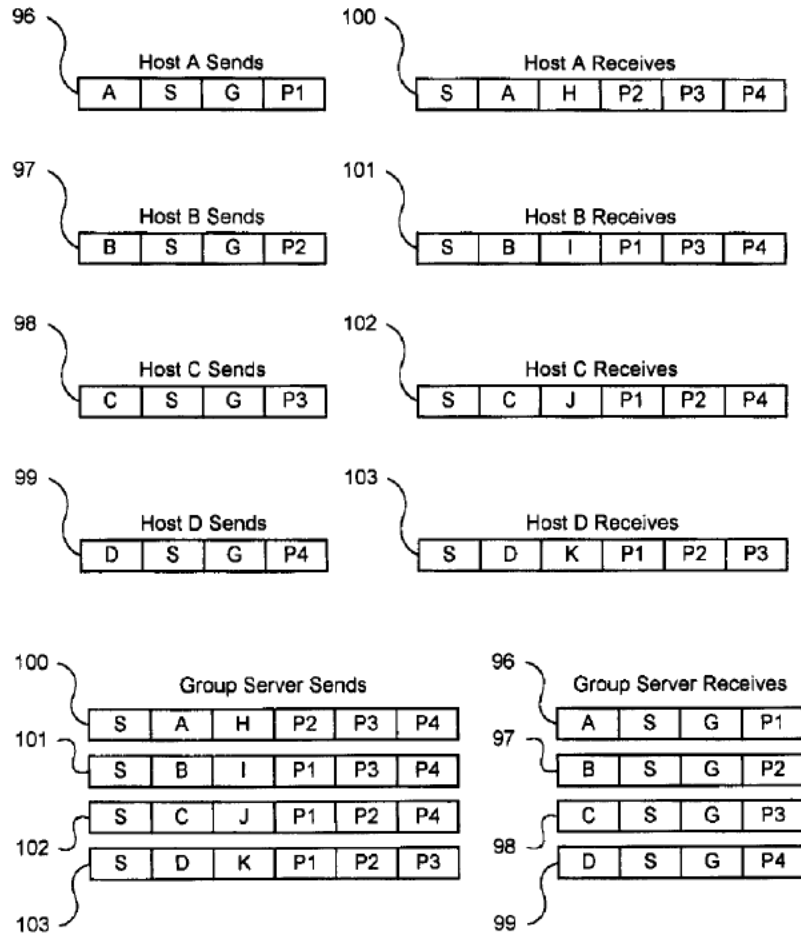


Figure 7

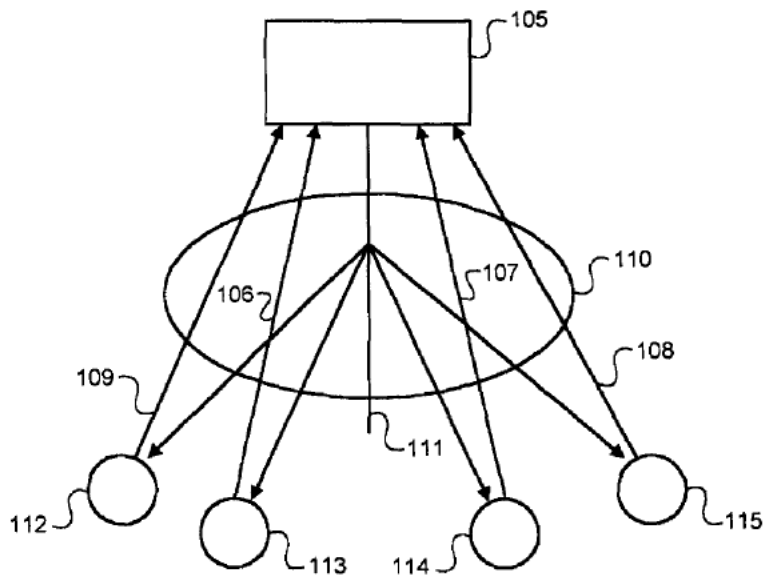


Figure 8
Prior Art

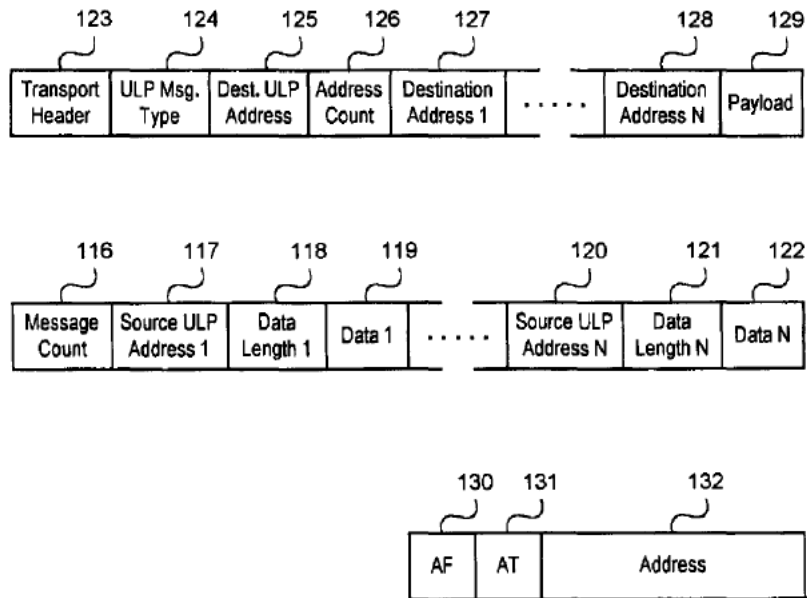


Figure 9

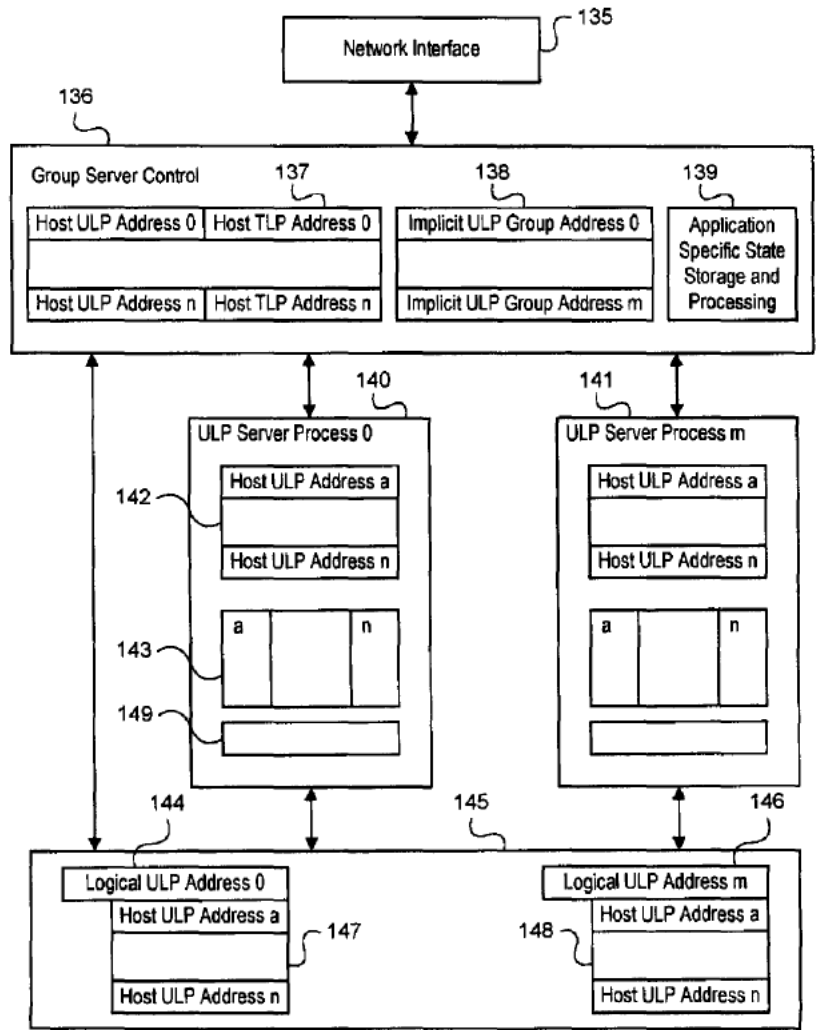


Figure 10

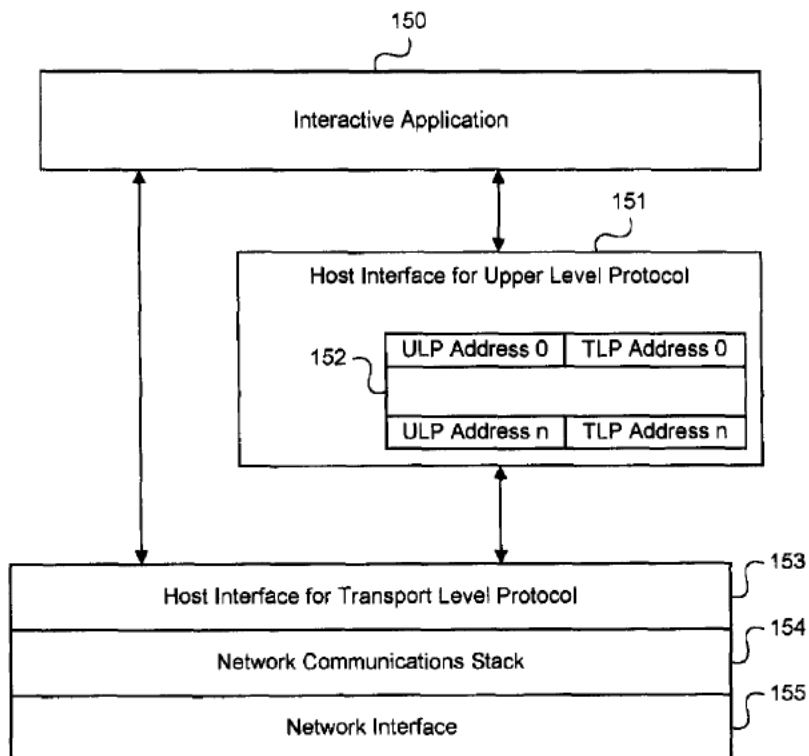


Figure 11

1

SERVER-GROUP MESSAGING SYSTEM FOR INTERACTIVE APPLICATIONS

FIELD OF THE INVENTION

The present invention relates to computer network systems, and particularly to server group messaging systems and methods for reducing message rate and latency.

BACKGROUND OF THE INVENTION

There are a wide range of interactive applications implemented on computer systems today. All are characterized by dynamic response to the user. The user provides input to the computer and the application responds quickly. One popular example of interactive applications on personal computers (PCs) are games. In this case, rapid response to the user may mean redrawing the screen with a new picture in between 30 ms and 100 ms. Interactive applications such as games control the speed of their interaction with the user through an internal time base. The application uses this time base to derive rates at which the user input is sampled, the screen is redrawn and sound is played.

As computers have become more powerful and common, it has become important to connect them together in networks. A network is comprised of nodes and links. The nodes are connected in such a way that there exists a path from each node over the links and through the other nodes to each of the other nodes in the network. Each node may be connected to the network with one or more links. Nodes are further categorized into hosts, gateways and routers. Hosts are computer systems that are connected to the network by one link. They communicate with the other nodes on the network by sending messages and receiving messages. Gateways are computer systems connected to the network by more than one link. They not only communicate with other nodes as do hosts, but they also forward messages on one of their network links to other nodes on their other network links. This processing of forwarding messages is called routing. In addition to sending and receiving messages and their routing functions, gateways may perform other functions in a network. Routers are nodes that are connected to the network by more than one link and whose sole function is the forwarding of messages on one network link to the other network links to which it is connected. A network consisting of many network links can be thought of as a network of sub-networks with gateways and/or routers connecting the sub-networks together into what is called an internet. Today the widely known example of a world wide internet is the so called "Internet" which in 1995 has over 10 million computers connected full time world-wide.

With so many computers on a single world-wide network, it is desirable to create interactive networked applications that bring together many people in a shared, networked, interactive application. Unfortunately, creating such shared networked, interactive applications runs into the limitations of the existing network technology.

As an example, consider a game designed to be deployed over a network which is to be played by multiple players simultaneously. The game could be implemented in software on a PC connected to a network. A rate set by its internal time base, it would sample the inputs of the local user, receive messages from the network from the PCs of the other players and send messages out to the PCs of the other players. A typical rate will be ten times per second for a time period of 100 ms. The messages sent between the PCs would contain information that was needed to keep the game

2

consistent between all of the PCs. In a game that created the illusion of a spatial environment where each player could move, the packets could contain information about the new positions of the players as they moved. Today there are many commercial example of PC games that can be played between multiple players on Local Area Networks (LANs) or by two players over dial-up phone lines using modems. The network messages sent by such games contain a wide variety of information specific to the game. This can include position and velocity information of the objects in the game along with special actions taken by a player that effect the other players in the game.

The case of a two player game played over a modem is particularly simple. If the message rate is 10 messages per second, each PC sends 10 messages per second to the other PC and receives 10 messages per second. The delay introduced by the modems and phone line is small and will not be noticed in most games. Unfortunately, the case of two players is uninteresting for networked interactive applications. With the same game played with 8 players on a LAN, the message rate increases. Each PC must send 7 messages, one to each of the other 7 players every time period and will receive 7 messages from the other players in the same time period. If the messaging time period is 100 ms, the total message rate will be 70 messages sent per second and 70 messages received per second. As can be seen the message rate increases linearly with the number of players in the game. The message rates and data rates supported by popular LANs are high enough to support a large number of players at reasonable message sizes. Unfortunately, LANs are only deployed in commercial applications and cannot be considered for deploying a networked interactive application to consumer users.

The wide area networks available today to consumer users all must be accessed through dial-up phone lines using modems. While modem speeds have increased rapidly, they have now reached a bit rate of 28.8 Kbits/sec which is close to the limit set by the signal-to-noise ratio of conventional phone lines. Further speed increases are possible with ISDN, but this technology is not ready for mass market use. Other new wide area networking technologies are being discussed that would provide much higher bandwidth, but none are close to commercial operation. Therefore, in deploying a networked, interactive application to consumers, it is necessary to do so in a way that operates with existing networking and communications infrastructures.

In the example of the 8 player networked game, consider a wide area network implementation where the PCs of each of the players is connected to the network with a 28.8 Kbit/sec modem. Assume that the network used in this example is the Internet so that all of the network protocols and routing behavior is well defined and understood. If the game uses TCP/IP to send its messages between the PCs in the game, the PPP protocol over the dial-up phone lines can be advantageously used to compress the TCP/IP headers. Even so, a typical message will be approximately 25 bytes in size. Sent through the modem, this is 250 bits. The messages are sent 10 times per second to each of the other PCs in the game and received 10 times per second from the other PCs. This is 35.0 Kbits/sec which exceeds the capabilities of the modem by 20%. If the messages are reduced to 20 bytes, just 8 players can be supported, but this approach clearly cannot support networked interactive applications with large numbers of participants. There are other problems beyond just the bandwidth of the network connection. There is the loading on each PC caused by the high packet rates and there is the latency introduced by the

3

time needed to send all of the outbound packets. Each packet sent or received by a PC will require some amount of processing time. As the packet rate increases with the number of players in the game, less and less of the processor will be available for running the game software itself. Latency is important in an interactive application because it defines the responsiveness of the system. When a player provides a new input on their system, it is desirable for that input to immediately affect the game on all of the other players systems. This is particularly important in any game where the game outcome depends on players shooting at targets that are moved by the actions of the other players. Latency in this case will be the time from when a player acts to move a target to the time that the target has moved on the screens of the other players in the game. A major portion of this latency will come from the time needed to send the messages to the other seven players in the game. In this example the time to send the messages to the other 7 players will be approximately 50 ms. While the first player of the seven will receive the message quickly, it will not be until 50 ms have passed that the last player of the seven will have received the message.

Internet Protocol Multicasting

As mentioned before, the Internet is a widely known example of a wide area network. The Internet is based on a protocol appropriately called the Internet Protocol (IP). In the OSI reference model for layers of network protocols, IP corresponds to a layer 3 or Network layer protocol. It provides services for transmission and routing of packets between two nodes in an internet. The addressing model provides a 32 bit address for all nodes in the network and all packets carry source and destination addresses. IP also defines the routing of packets between network links in an inter-network. Gateways and routers maintain tables that are used to lookup routing information based on the destination addresses of the packets they receive. The routing information tells the gateway/router whether the destination of the packet is directly reachable on a local network link connected to the gateway/router or if not, the address of another gateway/router on one of the local network links to which the packet should be forwarded. On top of IP are the layer 4 transport protocols TCP and UDP. UDP provides datagram delivery services to applications that does not guarantee reliable or in-order delivery of the datagrams. TCP is a connection oriented service to applications that does provide reliable delivery of a data stream. It handles division of the stream into packets and ensures reliable, in-order delivery. See the Internet Society RFCs: RFC-791 "Internet Protocol", RFC-793 "Transmission Control Protocol" and RFC-1180 "A TCP/IP Tutorial". IP, TCP and UDP are unicast protocols: packets, streams or datagrams are transmitted from a source to a single destination.

As an example, consider FIGS. 1 and 2. FIG. 1 shows a conventional unicast network with hosts 1, 2, 3 and 4 and network links 11, 12, 13, 14, 15, 16, 17, 18 and 19 and routers 5, 6, 7, 8, 9 and 10. In this example, each host wants to send a data payload to each of the other hosts. Host 1 has network address A, host 2 has network address C, host 3 has network address B and host 4 has network address D. Existing network protocols are typically based on packet formats that contain a source address, destination address and a payload. This is representative of commonly used wide area network protocols such as IP. There are other components in an actual IP packet, but for sake of this example, only these items will be considered. FIG. 2 shows the example packets that are sent by the hosts to one another using a conventional unicast network protocol such as IP. Host 1 send packets 20, to host

4

3, packet 21 to host 2 and packet 22 to host 4. Host 1 wants to send the same data P1 to each of the other three hosts, therefore the payload in all three packets is the same. Packet 20 travels over network links 11, 12, 15 and 18 and through routers 5, 6, and 8 to reach host 3. In a similar fashion host 3 sends packets 23 to host 1, packet 24 to host 2 and packet 25 to host 4. Host 2 and host 4 send packets 26, 27, 28 and 29, 30, 31 respectively to the other three hosts. All of these packets are carried by the unicast network individually from the source host to the destination host. So in this example each host must send three packets and receive three packets in order for each host to send its payload to the other three hosts.

As can be seen, each host must send a packet to every other host that it wishes to communicate with in an interactive application. Further, it receives a packet from every other host that wishes to communicate with it. In an interactive application, this will happen at a regular and high rate. All of the hosts that wish to communicate with one another will need to send packets to each other eight to ten times per second. With four hosts communicating with one another as in this example, each host will send three messages and receive three messages eight to ten times per second. As the number of hosts in the application that need to communicate with one another grows, the message rate will reach a rate that cannot be supported by conventional dial-up lines. This makes unicast transport protocols unsuitable for delivering interactive applications for multiple participants since their use will result in the problem of high packet rates that grow with the number of participants.

Work has been done to attempt to extend the IP protocol to support multicasting. See RFC-1112 "Host Extensions for IP Multicasting". This document describes a set of extensions to the IP protocol that enable IP multicasting. IP multicasting supports the transmission of a IP datagram to a host group by addressing the datagram to a single destination address. Multicast addresses are a subset of the IP address space and identified by class D IP addresses—these are IP addresses with "1110" in the high order 4 bits. The host group contains zero or more IP hosts and the IP multicasting protocol transmits a multicast datagram to all members of the group to which it is addressed. Hosts may join and leave groups dynamically and the routing of multicast datagrams is supported by multicast routers and gateways. It is proper to describe this general approach to multicast messaging as "distributed multicast messaging". It is a distributed technique because the job of message delivery and duplication is distributed throughout the network to all of the multicast routers. For distributed multicast messaging to work in a wide area network, all of the routers handling datagrams for multicast hosts must support the routing of multicast datagrams. Such multicast routers must be aware of the multicast group membership of all of the hosts locally connected to the router in order to deliver multicast datagrams to local hosts. Multicast routers must also be able to forward multicast packets to routers on their local network links. Multicast routers must also decide to which if any local routers they must forward multicast datagrams. When a multicast datagram is received, by a multicast router, its group address is compared to a list for each local multicast router of group addresses. When there is a match, the datagram is then forwarded to that local multicast router. Therefore, the multicast routers in the network must maintain an accurate and up to date list of group addresses for which they are to forward datagrams to. These lists are updated when hosts join or leave multicast groups. Hosts do this by sending messages using Internet

Group Management Protocol (IGMP) to their immediately-neighbor-
ing multicast routers. A further attribute of distributed
multicast messaging is that the routers must propagate
the group membership information for a particular group
throughout the network to all of the other routers that will be
forwarding traffic for that group. RFC-1112 does not
describe how this is to be done. Many different approaches
have been defined for solving this problem that will be
mentioned later in descriptions of related prior art. Despite
their differences, all of these approaches are methods for
propagation of multicast routing information between the
multicast routers and techniques for routing the multicast
datagrams in an inter-network supporting distributed multi-
cast messaging.

The distributed multicast messaging approach has a number
of undesirable side effects. The process of propagation of
group membership information to all of the relevant routers
is not instantaneous. In a large complex network it can even
take quite a period of time depending on the number of
routers that must receive that updated group membership
information and how many routers the information for the
group membership update must pass through. This process
can easily take many seconds and even minutes depending
on the specifics of the algorithm that is used. RFC-1112
mentions this problem and some of the side effects that must
be handled by an implementation of a practical routing
algorithm for multicast messaging. One problem results
when groups are dynamically created and destroyed. Since
there is no central authority in the network for assigning
group addresses, it is easily possible in a distributed network
for there to be duplication of group address assignment. This
will result in incorrect datagram delivery, where hosts will
receive unwanted datagrams from the duplicate group. This
requires a method at each host to filter out the unwanted
datagrams. Another set of problems result from the time
delay from when a group is created, destroyed or its mem-
bership changed to when all of the routers needed to route
the datagrams to the member hosts have been informed of
these changes. Imagine the case where Host N joins an
existing group by sending a join message to its local router.
The group already contains Host M which is a router. Router
hops away from Host N in the network. Shortly after
Host N has sent its join message, Host M sends a datagram
to the group, but the local router of Host M has not yet been
informed of the change in group membership and as a result
the datagram is not forwarded to one of the particular
network links connected to the local router of Host M that
is the only path in the network from that router that ulti-
mately will reach Host N. The result is that Host N will
receive no datagrams addressed to the group from Host M
until the local router of M has its group membership
information updated. Other related problems can also occur.
When a host leaves a group, messages addressed to the
group will continue for some time to be routed to that host
up to the local router of that host. The local router will know
at least not to route the datagram onto the local network of
that host. This can still result in a great deal of unnecessary
datagrams being carried in a large network when there are
many active message groups with rapidly changing mem-
berships.

Finally, distributed multicast messaging does not suffi-
ciently reduce the message rate between the hosts. With
distributed multicast messaging, each host need only send
one message addressed to the message group in order to send
a message to all of other hosts in the group. This is an
improvement over conventional unicast messaging where
one message would need to be sent to each of the other hosts

in a group. However, distributed multicast messaging does
nothing to reduce the received message rate at each of the
hosts when multiple hosts in a group are sending messages
to the group closely spaced in time. Let us return to the
example of a group of ten hosts sending messages seven
times per-second to the group. With conventional unicast
messaging, each host will need to send 9 messages to the
other hosts, seven times per-second and will receive 9
messages, seven times per-second. With distributed multi-
cast messaging, each host will need to send only one
message to the group containing all of the hosts seven times
per-second, but will still receive 9 messages, seven times
per-second. It is desirable to further reduce the number of
received messages.

An example of distributed multicasting is shown in FIGS.
3 and 4. FIG. 3 shows a network with multicast routers 39,
40, 41, 42, 43 and 44 and hosts 35, 36, 37, 38 and network
links 45, 46, 47, 48, 49, 50, 51, 52 and 53. The four hosts
have unicast network addresses A, B, C, D and are also all
members of a message group with address E. In advance the
message group was created and each of the hosts joined the
message group so that each of the multicast routers is aware
of the message group and has the proper routing informa-
tion. A network protocol such IP with multicast extensions
is assumed to be used in this example. Host 35 sends packet
54 with source address A and destination multicast address
E to the entire message group. In the same manner host 37
sends packet 55 to the group, host 36 sends packet 56 to the
group and host 38 sends packet 57 to the group. As the
packets are handled by the multicast routers they are repli-
cated as necessary in order to deliver them to all the
members of the group. Let us consider how a packets sent
by host 35 is ultimately delivered to the other hosts. Packet
54 is carried over network link 45 to multicast router 39. The
router determines from its routing tables that the multicast
packet should be sent onto network links 46 and 47 and
duplicates the packet and sends to both of these network
links. The packet is received by multicast routers 40 and 43.
Multicast router 43 sends the packet onto network link 50
and router 40 sends its onto links 48 and 49. The packet is
then received at multicast routers 44, 42 and 41. Router 41
sends the packet over network link 51 where it is received
by host 36. Router 42 sends the packet over network link 52
to host 37 and router 44 sends the packet over link 53 to host
38. A similar process is followed for each of the other
packets sent by the hosts to the multicast group E. The final
packets received by each host are shown in FIG. 4.

While distributed multicasting does reduce the number of
messages that need to be sent by the hosts in a networked
interactive application, it has no effect on the number of
messages that they receive. It has the further disadvantages
of poor behavior when group membership is rapidly chang-
ing and requires a special network infrastructure of multi-
cast routers. It also has no support for message aggregation and
cannot do so since message delivery is distributed. Distribu-
ted multicasting also has no support for messages that
define logical operations between message groups and uni-
cast host addresses.

All of these problems can be understood when placed in
context of the design goals for distributed multicast mes-
saging. Distributed multicast messaging was not designed
for interactive applications where groups are rapidly created,
changed and destroyed. Instead it was optimized for applica-
tions where the groups are created, changed and destroyed
over relatively long time spans perhaps measured in many
minutes or even hours. An example would be a video
conference where all the participants agreed to connect the

conference at a particular time for a conference that might last for an hour. Another would be the transmission of an audio or video program from one host to many receiving hosts, perhaps measured in the thousands or even millions. The multicast group would exist for the duration of the audio/video program. Host members would join and leave dynamically, but in this application it would be acceptable for there to be a significant time lag from joining or leaving before the connection was established or broken.

While IP and multicast extensions to IP are based on the routing of packets, another form of wide area networking technology called Asynchronous Transfer Mode (ATM) is based on switching fixed sized cells through switches. Unlike IP which supports both datagram and connection oriented services, ATM is fundamentally connection oriented. An ATM network consists of ATM switches interconnected by point-to-point links. The host systems are connected to the leaves of the network. Before any communication can occur between the hosts through the network, a virtual circuit must be setup across the network. Two forms of communication can be supported by an ATM network. Bi-directional point-to-point between two hosts and point-to-multipoint in one direction from one host to multiple hosts. ATM, however, does not directly support any form of multicasting. There are a number of proposals for layering multicasting on top of ATM. One approach is called a multicast server, shown in FIG. 8. Host systems 112, 113, 114, 115 setup point-to-point connections 106, 107, 108 and 109 to a multicast server 105. ATM cells are sent by the hosts to the multicast server via these links. The multicast server sets up a point-to-multipoint connection 111 to the hosts which collectively constitute a message group. Cells sent to the server which are addressed to the group are forwarded to the point-to-multipoint link 111. The ATM network 110 is responsible for the transport and switching for maintaining all of the connections between the hosts and the server. The cells carried by the point-to-multipoint connection are duplicated when necessary by the ATM switches at the branching points in the network tree between and forwarded down the branching network links. Therefore, the network is responsible for the replication of the cells and their payloads, not the server. This method has the same problems as distributed multicasting when used for an interactive application. Each host still receives individual cells from each of the other hosts, so there is no aggregation of the payloads of the cells targeted at a single host. There is no support for addressing cells to hosts based on logical operations on the sets of members of host groups.

Related Prior Art

There are a number of existing patents and European patent applications that are related to the area of the invention. These can be organized into two separate categories: multicast routing/distribution and source to destination multicast streams.

Multicast routing and distribution

These patents are U.S. Pat. No. 4,740,954 by Cotton et al, U.S. Pat. No. 4,864,559 by Perlman, U.S. Pat. No. 5,361,256 by Doeringer et al, U.S. Pat. No. 5,079,767 by Perlman and U.S. Pat. No. 5,309,433 by Cidon et al. Collectively these patents cover various algorithms for the routing and distribution of the datagrams in distributed multicast networks. None deal with the problems described previously for this class of multicast routing and message distribution such as poor behaviors when the message groups change rapidly. In all of these patents, messages are transmitted from a host via a distributed network of routers to a plurality of destination hosts which are members of a group. Since these patents

deal only with variants of distributed multicasting they provide no means to reduce the received message rate, no method to aggregate messages and provide no method in the messages to perform logical operation on message groups. Source to destination multicast streams

These are PCTs and a European patent application. They are EP 0 637 149 A2 by Perlman et al, PCT/US94/11282 by Danneels et al and PCT/US94/11278 by Sivakumar et al. These three patent applications deal with the transmission of data streams from a source to a group of destinations. In none of these patent applications, is a method described for transmitting data between multiple members of a group. In all of these applications, the data transmission is from a source to a plurality of designations. Since these patent applications deal only with point-to-multipoint messaging, they can provide no means to reduce the received message rate, no method to aggregate messages and provide no method in the messages to perform logical operation on message groups.

SUMMARY OF THE INVENTION

The present invention relates to facilitating efficient communications between multiple host computers over a conventional wide area communications network to implement an interactive application such as a computer game between multiple players. In such an application, the hosts will be dynamically sending to each other information that the other hosts need in order to keep the interactive application operating consistently on each of the hosts. The invention is comprised of a group messaging server connected to the network that maintains a set of message groups used by the hosts to communicate information between themselves. The invention further comprises a server-group messaging protocol used by the hosts and the server. The server-group messaging protocol is layered on top of the Transport Level Protocol (TLP) of the network and is called the Upper Level Protocol (or ULP). In the OSI reference model the ULP can be thought of as a session layer protocol built on top of a transport or applications layer protocol. The ULP protocol uses a server-group address space that is separate from the address space of the TLP. Hosts send messages to addresses in the ULP address space to a group messaging server using the underlying unicast transport protocol of the network. The ULP address space is segmented into unicast addresses, implicit group messaging addresses and logical group messaging addresses. The implicit and logical group messaging addresses are collectively called group messaging addresses.

Host systems must first establish connections to a group messaging server before sending messages to any ULP addresses. The process of establishing this connection is done by sending TLP messages to the server. The server establishes the connection by assigning a unicast ULP address to the host and returning this address in an acknowledgment message to the host. Once connected, hosts can inquire about existing message groups, join existing message groups, create new message groups, leave message groups they have joined and send messages to ULP addresses known by the server. Each message group is assigned either an implicit or logical ULP address depending on its type.

FIG. 5 shows an example of a wide area network with a group messaging server ("GMS"). Hosts 58 has TLP address A and ULP address H, host 59 has TLP address C and ULP address J, host 60 has TLP address B and ULP address I and host 61 has TLP address D and ULP address K. The network is a conventional unicast network of network links 69, 70, 71, 72, 73, 74, 75, 76, and 77 and unicast routers 63, 64, 65,

9

66, 67, and 68. The group messaging server 62 receives messages from the hosts addressed to a message group and send the contents of the messages to the members of the message group. FIG. 6 shows an example of datagrams sent from the hosts to a message group that they are members of. As before, a TLP such as IP (where the message header contain the source and destination TLP addresses) is assumed to be used here. Host 58 sends message 80 which contains the TLP source address A of the host and the destination TLP address S for the GMS 62. The destination ULP address G is an implicit ULP address handled by the GMS and the payload P1 contains both the data to be sent and the source ULP address H of the host. It is assumed that prior to sending their ULP messages to the GMS, that each host as already established a connection to the GMS and joined the message group G. Host 60 sends message 81 with payload P2 containing data and source ULP address I. Hosts 59 sends message 82 with payload P3 containing data and source ULP address J. Host 61 sends message 83 with payload P4 containing data and source ULP address K. The GMS receives all of these messages and sees that each message is addressed to implicit message group G with members H, I, J, and K. The GMS can either process the message with or without aggregating their payloads. FIG. 6 shows the case where there is no aggregation and FIG. 7 shows the case with aggregation.

Without aggregation, the GMS generates the outbound messages 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, and 95 which it sends to the hosts. The datagrams have TLP headers with the source and destination TLP addresses of the GMS and the hosts respectively. The next field in the datagrams is the destination ULP of the datagram. Datagrams 84, 85, and 86 are sent to host 58 with TLP address A and ULP address H. Datagrams 87, 88, and 89 are sent to host 60 with TLP address B and ULP address I. Datagrams 90, 91 and 92 are sent to host 59 with TLP address C and ULP address J. Datagrams 93, 94 and 95 are sent to host 61 with TLP address D and ULP address K respectively. As can be seen from the payloads that each host has received, each host has received the payloads from the other three hosts. Note that each host has not received a copy of its own original message. This is because the GMS has performed echo suppression. This is selectable attribute of the GMS since in some applications it is useful for the hosts to receive and echo of each message that they send to a group that they are also members of. In the example of FIG. 6, it has been shown how the present invention can achieve the same message delivery as distributed multicasting without its disadvantages. Without aggregation, the present invention enables a host to send a single message to multiple other hosts that are members of a message group. It reduces the message traffic that a host must process in an interactive application by reducing the number of messages that each host must send to the others. Without aggregation, however, there is no reduction in the number of messages received by the hosts. Without aggregation we can achieve the same message rate as distributed multicasting without the need for a network with multicast routers, we can use a conventional unicast network such as the Internet. The present invention also avoids the problems that dynamic group membership causes for distributed multicasting. Group membership can be changed very rapidly. Groups can be created, joined and left by single unicast messages from hosts to the GMS. These messages will be point-to-point messages and will not have to propagate in throughout the network nor have to cause routing table changes in the routers. This ability to rapidly and accurately change group membership is critical to the

10

implementation of networked interactive applications. Consider a computer game for multiple players that supports hundreds of players that are spread throughout a three dimensional space created by the game. At any time only a few players will be able to see and effect one another in the game since other players will be in other areas that are out of sight. Using conventional phone lines to carry the data from each players computer to the network, it will not be possible to send all actions of each player to all of the other players, but because only a few players will be in close proximity at any one time, it will not be necessary to do so. It is only necessary to send data between the players that are in close proximity to one another. These "groups" of players naturally map onto the message groups of the invention. As players move about the three dimensional space of the game, game will cause them to join and leave message groups as necessary. If this does not happen rapidly it will limit the interactivity of the game or cause inconsistent results for the different players in the game.

The invention also allows aggregating message payloads of multiple messages destined to a single host into a single larger message. This can be done because of the GMS where all of the messages are received prior to being sent to the hosts. FIG. 7 shows an example of how this works. The hosts send their messages to the GMS in exactly the same fashion as in FIG. 6 using the same addresses previously defined in FIG. 5. Host 58 sends message 96, host 60 sends message 97, host 59 sends message 98 and host 61 sends message 99. The GMS receives all of these messages and creates four outbound messages 100, 101, 102 and 103. The process by which these messages will be explained in detail in the detailed description of the invention. Each message is destined to a single host and contains an aggregated payload with multiple payload items. Message 100 has a destination ULP address H for host 58 and aggregated payload P2, P3 and P4 from the messages from hosts 59, 60 and 61. Message 101 is targeted at host 60, message 102 is targeted at host 59 and message 103 is targeted at host 61. As can be seen, each host sends one message and receives one message. The received message is longer and contains multiple payloads, but this is a significant improvement over receiving multiple messages with the wasted overhead of multiple message headers and message processing time. Overall the invention has dramatically reduced the amount of data that must be sent and received by each host. Since the bit rate over conventional phone lines using a modem is low, a reduction in the amount of data that must be sent and received directly translates into improved time and latency for message communications between the hosts.

Hosts create, join and leave message groups using control messages in the ULP protocol to the GMS. Hosts may also read and write application specific state information that is stored in the GMS. When hosts send messages to other hosts, the message must be at least addressed to an implicit group address. The ULP implicit address will always be the primary address in a message from one host to another. The message may optionally specify auxiliary destination addresses. In many cases the implicit ULP address will be the only destination ULP address in the message. The GMS will handle delivery of the ULP messages addressed to the implicit message group to all of the hosts that are members of the group. A ULP send message may optionally specify an address list of auxiliary addresses in addition to the primary destination of the implicit ULP address. This auxiliary address list can contain only unicast and logical ULP addresses. The address list can also specify set operators to be performed between the sets of host ULP addresses

11

defined by the unicast addresses and logical groups. Once the address list has been processed to yield a set of hosts, this set is intersected with the set of hosts that are members of the implicit message group specified by the primary implicit ULP address in the message. This ability to perform logical set operators on message groups is very useful in interactive applications. It allows a single ULP message to selectively deliver a message to hosts that fit a set of computed criteria without the sending host having to know anything about the members of the groups in the address list. Recall the example of a networked game with hundreds of players in a three dimensional environment created by the game. Consider an implicit message group consisting of all of the game players in a certain area of the game where all of the players can interact with one another. Consider that the players are organized into multiple teams. Logical message groups could be created for each team within the game. To send a message to all the players within the area that were on one team, a ULP message would be sent to the ULP implicit message group for all the players in the area with an auxiliary address of the logical message group for all the players on the selected team. The GMS would perform the proper set intersection prior to sending the resulting messages to the targeted hosts. The result of this will be that the message will only be delivered to the players on the selected team in the selected area of the game.

In summary, the present invention deals with the issues of deploying an interactive application for multiple participants on wide area networks by providing a method for reducing the overall message rate and reducing latency. This invention uses a server group messaging approach, as opposed to the above described "distributed multicast messaging" approach. The present invention overcomes the undesirable side effects of the distributed multicast messaging approach. Further, it reduces the message rate between the hosts. As pointed out in an example discussed above, with prior art distributed multicast messaging, each host will need to send only one message to the group containing all of the hosts seven times per-second, but will still receive 9 messages, seven times per-second. The present invention of server group messaging has each host sending one message, seven times per-second and receiving one message, seven times per-second.

The present invention is different from the multicast routing and distribution method disclosed in U.S. Pat. Nos. 4,740,954, 4,864,559, 5,361,256, 5,079,767 and 5,309,433. Since these patents deal only with variants of distributed multicasting they provide no means to reduce the received message rate, no method to aggregate messages and provide no method in the messages to perform logical operation on message groups. This differs from the present invention where messages from multiple hosts addressed to a message group are received by a group server which processes the contents of the messages and transmits the results to the destination hosts.

The present invention is also different from the source to destination multicast streams approach disclosed in EP 0 637 149 A2, PCT/US94/11282 and PCT/US94/11278. In all of these references, the data transmission is from a source to a plurality of designations, whereas the present invention describes data transmission from a sending host to a server host system and then from the server host to the destination hosts.

These and other features and advantages of the present invention can be understood from the following detailed description of the invention together with the accompanying drawings.

12

DESCRIPTION OF DRAWINGS

FIG. 1 shows a conventional unicast network consisting of hosts, network links and routers.

FIG. 2 shows the unicast datagrams on a conventional unicast network that would be needed to implement an interactive application between four hosts.

FIG. 3 shows a prior art multicast network consisting of hosts, network links and multicast routers.

FIG. 4 shows a multicast datagrams on a prior art multicast network that would be needed to implement an interactive application between four hosts.

FIG. 5 shows a unicast network equipped with a group messaging server in accordance with the present invention.

FIG. 6 shows the ULP datagrams without payload aggregation on a network according to the present invention that would be needed to implement an interactive application between four hosts.

FIG. 7 shows the ULP datagrams with payload aggregation on a network according to the present invention that would be needed to implement an interactive application between four hosts.

FIG. 8 shows a prior art ATM network with a multicast server.

FIG. 9 shows the detailed datagram format and address format for ULP messages in accordance with the present invention.

FIG. 10 shows the internal functions of the GMS according to the present invention.

FIG. 11 shows the host software interface and functions needed to support the ULP according to the present invention.

DETAILED DESCRIPTION OF THE INVENTION

The present invention provides a method for multiple host computers to efficiently communicate information to one another over a wide area network for the purposes of implementing an interactive application between multiple users. The method consists of three components: a host protocol interface, a protocol and a server. The protocol is between the host protocol interface and the server and is implemented on top of the network transport protocol of a wide area network. The protocol is called the Upper Level Protocol (ULP) since it is layered above the existing network Transport Level Protocol (TLP). In the OSI reference model the protocol can be described as a Session Layer protocol on top of the Transport Layer of the network. FIG. 11 shows the host protocol interface, 151, relative to the interactive application, 150, and the host interface for the Transport Level Protocol, 153. The network interface, 155, provides the physical connection for the host to the network. The network communications stack, 154, is the communications protocol stack that provides network transport services for the host and the host interface for the Transport Level Protocol, 153, is an interface between host application software and the network transport services of the network communications stack.

The interactive application can send and receive conventional network messages using the host interface to the TLP. The interactive application also can send and receive ULP messages through the host interface for the ULP. Internal to the host interface for the ULP is a table, 152, of all ULP addresses which the host can send messages to. Each entry in the table contains a pair of addresses, a ULP address and

its corresponding TLP address. When the host sends a message to a ULP address, that message is encapsulated in a TLP message sent to the TLP address corresponding to that ULP address. This allows the ULP messages to be handled transparently by the transport mechanisms of the existing network. A core function of the ULP is group messaging where hosts send messages to message groups populated by multiple hosts. This allows a host to send a message to multiple hosts with one ULP message. Since the ULP is layered on top of the TLP, the group messaging functions of the ULP operate on a conventional unicast network where TLP messages can only be sent from one host to only one other host.

The group based messaging is implemented through the use of a server called a group messaging server. All ULP messages from the hosts are sent from the hosts to a group messaging server using the TLP protocol. The server processes the ULP portion of the messages and takes the necessary required by the ULP message. Control ULP messages are processed locally by the server and may be acknowledged to the sending host. ULP messages addressed to other hosts are processed by the group messaging server and then re-transmitted to the proper ULP destination hosts, again using the TLP protocol to encapsulate and transport these messages.

In FIG. 5, hosts 58, 59, 60 and 61 send messages to one another using the ULP over a conventional unicast network using a group messaging server 62. The network consists of conventional routers 63, 64, 65, 66, 67 and 68 connected with conventional network links 69, 70, 71, 72, 73, 74, 75, 76 and 77. Host 58 can send a message to hosts 59, 60 and 61 by sending a single ULP message to the group messaging server 62 where the ULP message specifies a destination address that is a ULP message group. The ULP message is encapsulated in a TLP message addressed to the group messaging server. This causes the message to be properly routed by router 63 to network link 71 to router 67 to the server 62. The group messaging server receives the ULP message and determines that the message is addressed to a message group containing hosts 59, 60 and 61 as members. The server sends the payload of the received message to each of the hosts in three new ULP messages individually sent to the three hosts. Since each message is encapsulated in a TLP message, the messages are properly carried over the conventional unicast network. The first ULP message is sent by the group messaging server to host 61. This message is carried by network links 71, 70, 72 and 75 and routers 67, 63, 64 and 65. The second ULP message is sent by the group messaging server to host 60. This message is carried by network links 71, 70, 73 and 76 and routers 67, 63, 64 and 66. The third ULP message is sent by the group messaging server to host 61. This message is carried by network links 74 and 77 and routers 67 and 68.

The invention can be implemented both in a datagram form and in a connection oriented form. To best understand the details of the invention, it is best to first consider a datagram implementation.

Datagram Transport Implementation

The ULP can be implemented as a datagram protocol by encapsulating addresses, message type information and the message payload within a datagram of the underlying network transport protocol. The general form of the ULP datagram message format is shown in FIG. 9 as elements 123, 124, 125, 126, 127, 128 and 129. The transport header 123 is the datagram header of the TLP that is encapsulating the ULP datagram. The ULP message type field 124 indicates whether it is a send or receive message, if it is a control

message or a state message. The following table shows the different message types. The ULP message type field must be present in a ULP datagram.

Message Types	
1	Send
2	Receive
3	Send Control
4	Receive Control
5	Send State
6	Receive State

Send messages are always sent from a host to a group messaging server. Messages from a group server to the hosts are always receive messages. Send Control messages are messages from hosts to a group messaging server requesting a control function be performed. Receive Control messages are acknowledgments from a group messaging server to the hosts in response to a prior Send Control messages. The Send and Receive State messages are special cases of the Send and Receive Control messages that allow hosts to read and write application specific state storage in the group messaging server. The specific control functions supported by the ULP will be explained later.

The destination ULP address 125 is required in ULP datagrams and specifies the primary destination of the ULP message. The address count field 126 is required in ULP send message types and is not present in ULP receive message types. When the address count field in a ULP send message is non-zero, it specifies the number of auxiliary destination addresses for the send message that follow the address count field. These auxiliary destination addresses are shown as items 127 and 128, but it is understood that there are as many auxiliary ULP destination addresses as specified by the address count field. Finally there is the payload 129.

The payload format for ULP datagrams is defined by items 116, 117, 118, 119, 120, 121 and 122. Item 116 is the message count and defines how many payload elements will be contained in the payload. A single payload element consists of a triplet of source ULP address, data length and data. Items 117, 118 and 119 comprise the first payload element of the payload. Item 117 is the ULP address of the source of the payload element, item 118 is the data length for the data in the payload element and item 119 is the actual data. Items 120, 121 and 122 comprise the last payload element in the payload. ULP send messages only support payloads with a single payload element, so the message count is required to be equal to one. ULP receive messages may have payloads with one or more payload elements.

ULP Address Space

The address space of the ULP is divided into three segments: unicast host addresses, implicit group addresses and logical group addresses. All source and destination addresses in ULP must be in this address space. The ULP address space is unique to a single group messaging server. Therefore each group messaging server has a unique ULP address space. Multiple group messaging servers may be connected to the network and hosts may communicate with multiple group messaging servers without confusion since each ULP datagram contains the header of the TLP. Different group messaging servers will have unique TLP addresses which can be used by the hosts to uniquely identify multiple ULP address spaces. The format for ULP addresses is shown in FIG. 9 comprised of items 130, 131 and 132. The address format field 130 is a variable length field used to allow multiple address lengths to be supported. The address type

15

field 131 indicates the type of ULP address: unicast host, implicit group or logical group. The encoding is as follows:

Address Type Encoding	
0 0	Unicast Host Address
0 1	Unicast Host Address
1 0	Implicit Group Address
1 1	Logical Group Address

The address format encoding determines the length of the address field and therefore the total length of the ULP address. This encoding is shown below. Note that when the address type specifies a unicast host address, the low bit of the address type field is concatenated to the address field to become the most significant bit of the address. This doubles the size of the address space for unicast host addresses which is useful since there will generally be more hosts than group messaging servers.

Address Format Encoding	
0	29 Bit Address Field
1 0	4 Bit Address Field
1 1 0	11 Bit Address Field

ULP unicast host addresses are assigned to each host when it first connects to a group messaging server. When a host sends a message to other ULP address, the unicast ULP address of the host will appear as the source ULP address in the received payload element. Unicast ULP host addresses can also be used as destination addresses only as auxiliary addresses in a ULP send message. They are not allowed to be used to as the primary ULP destination address. This means that hosts cannot send ULP directly to one another, but always must send the messages to one another through a group messaging server.

Implicit group addresses are created by a group messaging server in response to a control message to the server requesting the creation of an implicit message group. The host requesting the creation of the implicit message group becomes a member of the message group when it is created. Other hosts can send inquiry control messages to the group messaging server to learn of its existence and then send a implicit group join message in order to join the group. The group messaging server maintains a list of ULP addresses of hosts that are members of the implicit message group. Implicit ULP group addresses are the only ULP addresses allowed to be the primary destination of a ULP send message. Implicit ULP addresses will never appear as ULP source addresses in a payload element.

Logical ULP addresses are used both to address logical message groups and for specifying set operations between the group members of the auxiliary ULP addresses in a ULP send message. Logical message groups are created and joined similarly to implicit message groups, however, logical ULP addresses may only be used as auxiliary ULP addresses in a ULP send message. Logical ULP addresses will also never appear as source ULP addresses in a payload element. The support of set operations between message groups as part of a ULP send message will be explained in a later section on ULP send messages.

Group Messaging Server Internal Functions

The internal components of the group messaging server are shown in FIG. 10.

In the preferred embodiment, the group messaging server is a general purpose computer system with a network

16

interface to connect it to a wide area network. Item 135 is the network interface for the group messaging server and includes not only the hardware connection to the network but the communications protocol stack used to implement the TLP on the server.

Item 136 is an overall control function for the group messaging server. This control function is responsible for all ULP messages that are sent or received by the GMS. Internal to this control function are several important storage and processing functions. Item 137 is an address map for all hosts currently connected to the GMS. This address map is a list of the ULP host address of each host connected to GMS and its corresponding TLP address. This enables the control function to construct the necessary TLP headers for sending ULP messages to the hosts connected to the GMS. Item 138 is a list of all of the currently active implicit ULP addresses currently recognized by the GMS. Item 139 is an application specific state storage and processing function. Many interactive applications deployed over a network will be able to be implemented solely with host based processing. In these cases all data that needs to be sent between the hosts can be transported using the ULP. However, some applications will need maintain a centrally stored and maintained repository of application state information. This is useful when hosts may join or leave the application dynamically. When hosts join such an application, they will need a place from which they can obtain a snapshot of the current state of the application in order to be consistent with the other hosts that already where part of the application. To read and write this state storage area, the ULP supports send and receive state message types. Within these messages, there is the ability to access a state address space so that different portions of the state can be individually accessed. Application specific processing of state written into this state storage area can also be implemented.

Items 140 and 141 are two of multiple ULP server processes running on the GMS. These are software processes that are at the heart of the ULP. Each implicit ULP addresses recognized by the GMS has a one-to-one correspondence to a ULP server process and to a message group maintained by the process. Since all ULP send messages must have an implicit ULP address as the primary destination address of the message, every ULP send message is sent to and processed by a ULP server process. These processes are created by the GMS control function in response to ULP control messages to create new implicit ULP addresses. They are destroyed when the last host which is a member of its message group has left the message group. Internal to a ULP server process is a list, 142, of the ULP host addresses of the members of the message group, a set of message queues 143 for each host which is a member of the message group and a message aggregation function 149 which is used to aggregate multiple messages to a single host into a single message.

Item 145 maintains a list of all of the logical ULP addresses and message groups in the GMS. Items 144 and 146 represent two of multiple logical ULP addresses. For each logical ULP address, there is a corresponding list, 147 and 148 of the host ULP addresses of the members of the logical message group. The logical message groups are not tied to specific ULP server processes, but are global with a GMS to all of the ULP server processes.

Control Functions

The control functions consist of connect, disconnect, create group, close group, join group, leave group, query groups, query group members, query group attributes. These control functions are implemented by a ULP send and

17

receive control messages. The control functions are initiated by a host sending a ULP send control message to a GMS. These messages only allow a primary ULP destination address in the message and do not allow auxiliary addresses. The primary ULP address is interpreted as a control address space with a unique fixed address assigned to each of the control functions enumerated above. The contents of data in the payload supplies any arguments needed by the control function. Returned values from the control function are returned in a ULP receive control message that is addressed to the host that sent the original control message for which data is being returned. The detailed operation of these control functions is described below.

Connect

This control function allows a host to connect to a GMS. The destination ULP address in the message is a fixed address that indicates the connect function. The source ULP address and any data in the payload are ignored.

Upon receiving this message, the GMS control function, 136, creates a new host address and enters the host address in the host address map 136 along with the source ULP address from the ULP header of the message. Upon successful completion, the GMS control function responds with a receive control ULP message addressed to the host along with a function code in the data portion of the payload that indicates successful host connection. The destination ULP address in the message is the ULP address assigned to the host. The host saves this and uses it for any future messages to the GMS. If there is an error, the control function returns a message to the host with a function code in the data portion of the payload indicating failed host connection.

Disconnect

This function allows a host to disconnect from a GMS. The destination ULP address in the message is a fixed address that indicates the disconnect function. The source ULP address is used to remove the host from membership in any implicit or logical groups prior to disconnecting. Any data in the payload is ignored. The GMS control function also removes the entry for the host from the host address map. Upon successful completion, the GMS control function responds with a receive control ULP message addressed to the host along with a function code in the data portion of the payload that indicates successful host disconnection. The destination ULP address in the message is the ULP address assigned to the host. If there is an error, the control function returns a message to the host with a function code in the data portion of the payload indicating failed host disconnection.

Create implicit group

This function allows a host to create a new implicit message group and associated implicit ULP address and server process. The payload in the message may contain a single payload item whose data field holds attributes of the group. These attributes can be used to define any optional functions of the group. The destination ULP address in the message is a fixed address that indicates the create implicit group function. The GMS control function allocates a new implicit ULP address, adds it to the implicit ULP address list 138 and creates a new ULP server process 140. The host that sends this message is added to the membership list of the implicit group. This is done by adding the source ULP address in the message to the group membership list 142 in the ULP server process. Upon successful completion, the GMS control function responds with a receive control ULP message addressed to the host along with a function code in the data portion of the payload that indicates successful implicit group creation. The source ULP address in the payload is the ULP address assigned to the new implicit

18

group. If there is an error, the control function returns a message to the host with a function code in the data portion of the payload indicating failed implicit group creation.

Create logical group

This function allows a host to create a new logical message group and associated logical ULP address. The payload in the message may contain a single payload item whose data field holds attributes of the group. These attributes can be used to define any optional functions of the group. The destination ULP address in the message is a fixed address that indicates the create logical group function. The GMS control function allocates a new logical ULP address and adds it to the logical ULP address list 145. The host that sends this message is added to the membership list of the logical group. This is done by adding the source ULP address in the message to the group membership list 147 for the new logical message group 144. Upon successful completion, the GMS control function responds with a receive control ULP message addressed to the host along with a function code in the data portion of the payload that indicates successful logical group creation. The source ULP address in the payload is the ULP address assigned to the new logical group. If there is an error, the control function returns a message to the host with a function code in the data portion of the payload indicating failed implicit group creation.

Join group

This function allows a host to join an existing logical or implicit message group. The destination ULP address in the message is a fixed address that indicates the join group function. The data portion of the payload contains the ULP address of the group that is to be joined. The GMS control function looks at this address and determines if it is an implicit or logical ULP address. If it is an implicit ULP address, the GMS control function finds the ULP server process selected by the address in the message payload and adds the source ULP host address from the message to the group membership list 142. If it is a logical ULP address, the GMS control function finds the logical ULP address 144 selected by the address in the message payload and adds the source ULP host address from the message to the group membership list 147. Upon successful completion, the GMS control function responds with a receive control ULP message addressed to the host along with a function code in the data portion of the payload that indicates successful group join. The source ULP address in the payload is the ULP address of the group that was joined. If there is an error, the control function returns a message to the host with a function code in the data portion of the payload indicating failed implicit group creation.

Leave group

This function allows a host to leave an existing logical or implicit message group that it is a member of. The destination ULP address in the message is a fixed address that indicates the leave group function. The data portion of the payload contains the ULP address of the group that is to be left. The GMS control function looks at this address and determines if it is an implicit or logical ULP address. If it is an implicit ULP address, the GMS control function finds the ULP server process selected by the address in the message payload and removes from the group membership list 142 the source ULP host address from the message. If the host is the last member of the group, the ULP server process is terminated and the implicit ULP address is de-allocated. If it is a logical ULP address, the GMS control function finds the logical ULP address 144 selected by the address in the message payload and removes from the group membership

list 147 the source ULP host address from the. If the host is the last member of the group, the ULP address is de-allocated. Upon successful completion, the GMS control function responds with a receive control ULP message addressed to the host along with a function code in the data portion of the payload that indicates successful group leave. If there is an error, the control function returns a message to the host with a function code in the data portion of the payload indicating failed implicit group creation.

Query groups

This function allows a host to get a list of all implicit and logical message groups currently active on a GMS. The destination ULP address in the message is a fixed address that indicates the query groups function. Any data portion of the payload is ignored. Upon successful completion, the GMS control function responds with a receive control ULP message addressed to the host along with a payload with multiple payload elements. The first payload element contains a function code indicating successful query groups. The source ULP address in the first payload element is ignored. Each of the subsequent payload elements contain a ULP group address in the source address field of the payload element that is one of the active group addresses on the GMS. There is no data field in these subsequent payload elements. If there is an error, the control function returns a message to the host with a function code in the data portion of a payload with a single payload element indicating failed query groups.

Query group members

This function allows a host to get a list of all hosts that are members of a message group. The destination ULP address in the message is a fixed address that indicates the query group members function. The data portion of the payload carries the address of the message group for the query. Upon successful completion, the GMS control function responds with a receive control ULP message addressed to the host along with a payload with multiple payload elements. The first payload element contains a function code indicating successful query group members. The source ULP address in the first payload element is ignored. Each of the subsequent payload elements contain a ULP host address in the source address field of the payload element that is one of the active group addresses on the GMS. There is no data field in these subsequent payload elements. If there is an error, the control function returns a message to the host with a function code in the data portion of a payload with a single payload element indicating failed query group members.

Query group attributes

This function allows a host to get a list of the attributes of a message group. The destination ULP address in the message is a fixed address that indicates the query group attributes function. The data portion of the payload carries the address of the message group for the query. Upon successful completion, the GMS control function responds with a receive control ULP message addressed to the host along with a payload with two payload elements. The first payload element contains a function code indicating successful query group members. The second payload element contains the attributes of the message group. If there is an error, the control function returns a message to the host with a function code in the data portion of a payload with a single payload element indicating failed query group attributes.

Send Message Operation

In order to fully understand the operations of the send message function, a number of individual cases are worth considering.

Single implicit destination

The most simple case is a send message to a single implicit ULP address. In all send message datagrams, the destination ULP address 125 must be an implicit ULP address. In this case of a single implicit destination, this is the only destination address in the datagram. The auxiliary address count 126 is zero and there are no auxiliary destination addresses 127 or 128. The payload consists of a message count 116 of one, the ULP of the host sending the message in the source ULP address 117 and the data length 118 and data 119. Send message datagrams may only have a single payload item so their message count field 116 must always be one.

The host sends the send message onto the network with a TLP header addressing the data. The GMS that is the selected target of the message. The GMS receives the message and the GMS control function 136 determines that it is a send message datagram and looks up the implicit destination address in its implicit ULP address list 138. If the address does not exist, an error message is returned to the sending host with a ULP receive message datagram. If the address is valid, the GMS control function removes the TLP header from the datagram and sends the ULP portion to the ULP server process corresponding to the destination implicit ULP address. Assume for discussion that this is the ULP server process 140. The ULP server process 140 will extract the single payload item from the message 117, 118 and 119 and place the payload item in each of the message queues 143. There will be one message queue for each member of the message group served by the ULP server process 140. The members of the group will have their host ULP addresses listed in the host address list 142. Each message queue in a ULP server process will fill with payload items that are targeted at particular destination hosts. The mechanisms by which payload items are removed from the queues and sent to the hosts will be described later.

Auxiliary unicast destination

In this case in addition to an implicit destination 125, there is also a single auxiliary address 127 in the datagram. The auxiliary address count 126 is one and the auxiliary destination addresses 127 is a unicast host ULP address. The payload consists of a message count 116 of one, the ULP of the host sending the message in the source ULP address 117 and the data length 118 and data 119.

The host sends the send message onto the network with a TLP header addressing the datagram to the GMS that is the selected target of the message. The GMS receives the message and the GMS control function 136 determines that it is a send message datagram and looks up the implicit destination address in its implicit ULP address list 138 and the unicast host ULP auxiliary address in the host address map 137. If either of addresses does not exist, an error message is returned to the sending host with a ULP receive message datagram. If the addresses are valid, the GMS control function removes the TLP header from the datagram and sends the ULP portion to the ULP server process corresponding to the destination implicit ULP address. Assume for discussion that this is the ULP server process 140. The ULP server process extracts the auxiliary ULP address from the message and determines from the address that it is a unicast host ULP address. The server process then checks to see if this address is a member of the message group defined by the host address list 142. If it is not, no further action is taken and the payload item in the message is not placed in any of the message queues 143. If the host address is in the message group, the payload item in the message is placed in the single message queue correspond-

21

ing to that host. The net effect is that the ULP server process has performed a set intersection operation on the members of the message group selected by the implicit ULP destination address and defined by the group membership list 142 with the members of the set of hosts defined by the auxiliary address. The payload item is then sent only to the hosts that are members of this set intersection.

Auxiliary logical destination

In this case in addition to an implicit destination 125, there is also a single auxiliary address 127 in the datagram. The auxiliary address count 126 is one and the auxiliary destination addresses 127 is a logical ULP address. The payload consists of a message count 116 of one, the ULP of the host sending the message in the source ULP address 117 and the data length 118 and data 119.

The host sends the send message onto the network with a TLP header addressing the datagram to the GMS that is the selected target of the message. The GMS receives the message and the GMS control function 136 determines that it is a send message datagram and looks up the implicit destination address in its implicit ULP address list 138 and the logical ULP auxiliary address in list of logical ULP addresses 145. If either of addresses does not exist, an error message is returned to the sending host with a ULP receive message datagram. If the addresses are valid, the GMS control function removes the TLP header from the datagram and sends the ULP portion to the ULP server process corresponding to the destination implicit ULP address. Assume for discussion that this is the ULP server process 140. The ULP server process extracts the auxiliary ULP address from the message and determines from the address that it is a logical ULP address. Assume for this example that this logical ULP address is the logical address 144. The server process fetches the group membership list 147 corresponding to the logical address and performs a set intersection operation with the group membership list 142 of the server process. If there are no members of this set intersection, no further action is taken and the payload item in the message is not placed in any of the message queues 143. If there are members of the set intersection operation, the payload item in the message is placed in the queues corresponding to the hosts that are members of the set intersection.

Multiple auxiliary addresses with logical operations

In its most sophisticated form, a send message can perform set operations between the implicit message group of the ULP server process and multiple logical and unicast ULP addresses. This is done by placing multiple auxiliary destination ULP addresses in the message with logical operators imbedded in the address list. The address count 126 holds a count of the total auxiliary addresses in the address list 127 and 128. The auxiliary addresses are a mix of logical ULP addresses and unicast host ULP addresses. Two logical ULP addresses in the ULP address space are assigned the role of specifying set operations to be performed between the logical message groups and unicast host addresses in the message list. They are specially assigned addresses for the functions set intersection, set union. A third logical address is used to indicate set complement. The payload consists of a message count 116 of one, the ULP of the host sending the message in the source ULP address 117 and the data length 118 and data 119.

The host sends the send message onto the network with a TLP header addressing the datagram to the GMS that is the selected target of the message. The GMS receives the message and the GMS control function 136 determines that it is a send message datagram and looks up the implicit ULP

22

message in the implicit ULP address list 138 and all of the addresses in the address list either in the host ULP address map 137 or in the logical ULP address list 145 as appropriate. If any of addresses does not exist, an error message is returned to the sending host with a ULP receive message datagram. If the addresses are valid, the GMS control function removes the TLP header from the datagram and sends the ULP portion to the ULP server process corresponding to the destination implicit ULP address. Assume for discussion that this is the ULP server process 140. The ULP server process extracts the auxiliary ULP address list from the message and scans it from beginning to end. The scanning and processing of the set operators is done in post-fix fashion. This means that arguments are read followed by an operator that is then applied to the arguments. The result of the operator becomes the first argument of the next operation. Therefore at the start of scanning two addresses are read from the address list. The next address will be an operator that is applied to the arguments and the result of this operator is the first argument to be used by the next operator. From then on a single address is read from the address list followed by a logical ULP address which is operator on the two arguments consisting of the new argument and the results of the last operator. The logical address used to indicate set complement is not a set operator, by an argument qualifier since it can precede any address in the address list. The meaning of the set complement argument qualifier is relative to the group membership of implicit group address in the send message. If the set complement qualifier precedes a unicast host address which is not a member of the message group selected by the implicit ULP address in the send message, the effective argument is the set of all hosts that are members of the implicit message group. If the set complement qualifier precedes a unicast host address which is a member of the message group selected by the implicit ULP address in the send message, the effective argument is the set of all hosts that are members of the implicit message group except for the original unicast host address qualified by the complement function. If the set complement qualifier precedes a logical ULP address the effective argument is the set of all hosts that are members of the implicit message group specified by the send message except hosts that are members of the logical message group preceded by the set complement modifier. Once the entire address list has been processed to a single result set of hosts, a set intersection operation is performed on this set and the set of members of the implicit message group 142 defined by the implicit address in the send message. If there are no members of this set intersection, no further action is taken and the payload item in the message is not placed in any of the message queues 143. If there are members of the set intersection operation, the payload item in the message is placed in the queues corresponding to the hosts that are members of the set intersection.

Message Delivery and Aggregation

Once messages are entered into the message queues in the ULP server processes, there are a variety of ways that they can ultimately be delivered to the targeted hosts. In the invention, the delivery method is set on a per-ULP server process basis by attributes that are provided at the time that an implicit ULP message group and server process are created. It is important during the description of these methods to keep in mind that the invention is intended to provide an efficient means for a group of hosts to send messages to each other at a rapid rate during the implementation of a networked interactive application. Also assumed in the following description is that the GMS performs echo

23

suppression when a host sends a message to a group that it belongs to. This means that the host will not receive a copy of its own message to the group either as a single un-aggregated message or as a payload item in an aggregated message. This is controlled by a ULP server process attribute that can be changed to stop echo suppression, but echo suppression is the default.

Immediate Delivery

The most simple delivery method is to immediately deliver the payload items to their targeted hosts as soon as they are placed in the message queues. Each payload item in a message queue will contain a ULP source address, a data length and the data to be sent. To implement immediate delivery, the ULP server process will remove a payload item from a message queue for a particular host 143. The host address for this host will be obtained from the group membership list 142. The payload item and the destination host address will be sent to the GMS control function 136 where it will be used to create a ULP receive message sent to the destination host. The GMS control function 136 will use the destination ULP host address to look up the TLP address of the host from the host address map 137. This will be used to create a TLP header for the message 123. The ULP message type 124 will be ULP receive, the destination ULP address 125 will be the destination host, the address count will be 0 and there will be no auxiliary addresses. The payload in this case will have a message count 116 of 1 and the payload item comprised of fields 117, 118, and 119 will be the payload element taken from the message queue.

Immediate delivery is useful when the message rate between a group of hosts is low. Consider four hosts that are members of an implicit message group where each member of the group sends a message to every other member of the group at a fixed rate. With immediate delivery, each host will send three messages to the other members of the group and receive three messages from the other members of the group at the fixed rate. This is acceptable if the size of the group is small and the message rate is low. However, it is obvious that total message rate is the product of the underlying message rate and the total number of members of the group minus one. Clearly this will result in unacceptably high message rates for large groups and highly interactive message rates. A group of 20 members that had an underlying message rate of 10 messages per second would yield a total message rate at each host of 190 messages sent and 190 messages received every second. This message rate will be unsupportable over a conventional dial-up connection to a conventional wide area network such as the internet.

Aggregation

A key concept in the present invention is the aggregation of multiple messages in a message queue into a single ULP receive message to a host that contains multiple payload items in the payload. The ULP server process 140 removes payload items from a message queue 143 for a host and accumulates them in an aggregation buffer 149. The aggregation buffer has buffer areas for each host for which there is a message queue. These individual host areas within the aggregation buffer are called host aggregation buffers. The start and end of this aggregation period can be controlled in a number of ways that will be described in the next sections. At the end of the aggregation period, the each host aggregation buffer may hold multiple payload items. The host aggregation buffer will hold a message count of the payload items followed by the multiple payload items. The contents of a host aggregation buffer along with the ULP host address of the corresponding host are sent to the GMS control function 136 where it will be used to create a ULP receive

24

message sent to the destination host. The GMS control function 136 will use the destination ULP host address to look up the TLP address of the host from the host address map 137. This will be used to create a TLP header for the message 123. The ULP message type 124 will be ULP receive, the destination ULP address 125 will be the destination host, the address count will be 0 and there will be no auxiliary addresses. The payload in this case will have a message count 116 set by the message count value from the host aggregation buffer. The payload will contain all of the payload items from the host aggregation buffer.

The effect of aggregation will be to greatly reduce the total message rate received by the hosts. A single message to a host will be able to carry multiple payload items received from the other hosts during the aggregation period. This fits very well the interactive applications of this invention where groups of hosts will be sending messages to all the other hosts in the group at a periodic rate. Aggregation will be very effective in collecting together all of the messages from all of the other hosts into a single message for each member of the group. This reduces processing at each receiving host since a single message will be received rather than many separate messages. Aggregation will also reduce the total data rate to the hosts since aggregation eliminates the need for separate message headers for each payload item. The savings will be significant for small payload items since there will be only one message header comprising fields 123, 124 and 125 for multiple payload items. In cases where a group of hosts are sending messages to the group at a periodic rate, it is often the case in many interactive applications that the data being sent by each host to the group is very similar to the messages sent by the other hosts. This affords the opportunity within an aggregated payload of multiple payload items to apply a data compression method across the multiple data elements of the payload elements. A wide variety of known data compression methods will lend themselves to this application. The first data element in the first payload item can be sent in uncompressed form with each subsequent data element being compressed using some form of difference coding method. A variety of known data compression methods use the concept of a predictor with differences from the predicted value being encoded. The first data element in an aggregated payload can be used as this predictor with the subsequent data elements coded using such a data compression method. These conventional data compression methods do not assume any knowledge of the internal structure or function of portions of a data element to compress. It is also possible to make use of application specific coding techniques that take advantage of such knowledge to potentially achieve much higher coding efficiency.

Server Isochronous

One method by which the aggregation time period can be defined is called Server Isochronous or SI. In this method, A ULP Server Process defines a uniform time base for defining the aggregation time period. This time base is defined by three parameters: the time period, the aggregation offset and the transmit offset. These parameters are set by the attributes provided in the create implicit group control function at the time the implicit group and the ULP server process are created. The time period is a fixed time interval during which the ULP server process will accumulate messages in the message queues, aggregate the messages in the queues and send the aggregated messages to the targeted hosts. The aggregation offset defines the point after the start of the time period after which arriving messages will be stored in the message queues for delivery in the next time period.

25

Therefore, at the aggregation offset after the start of the time period, a snapshot will be taken of all of the messages in each message queue. New messages will continue to arrive and be entered into the queues after the aggregation offset. Only those messages in the queues before the aggregation offset point will be aggregated into outbound messages. The resulting aggregated messages will then be sent to their targeted hosts at the point in time which is the transmit offset after the start of the time period. The result is that messages arrive continuously and are stored in the message queues. Once per time period they are aggregated into single messages to each host which is the target of messages and once per time period these aggregated messages are sent to the hosts.

Another embodiment of the SI method is to allow the ULP server process to dynamically vary the time period based on some criteria such as the received message rates, and/or received data rate. The ULP server could use a function to define the aggregation period based on the number of messages received per second or the total number of payload bytes received per second. One reasonable function would be to shorten the aggregation period as the rate or received messages or data rate of the received payloads increased. This would tend to keep the size of the outbound messages from growing too much as received messages and/or received data rate grew. Other possible functions could be used that varied the aggregation period based on received message rates, received payload data rates or other parameters available to the ULP server process.

Host Synchronous

The host synchronous or HS method of defining the aggregation time period allows the definition of a flexible time period that is controlled by the hosts. It is based on the concept of a turn which is a host sending a message to one or more members of the implicit message group which is operating in HS mode. Once every host in the message group has taken a turn, the aggregation period ends. A snapshot of the contents of the message queues is taken, the contents of each of the queues is aggregated and the aggregated messages are sent to the hosts targeted by each message queue. A refinement to this technique qualifies which of the three ULP send message types to the group constitute a host turn: a send only to the implicit address of the group, a send to a unicast host address within the group or a send to a logical ULP address which shares members with the group. The attributes of the group not only will define HS aggregation, but one or more ULP send message types that will be considered a host turn. A further refinement sets the total number of turns that a host can take in a single aggregation time period. The default will be one turn, but multiple turns can be allowed. If a host attempts to take more turns than allowed, the messages are ignored.

This aggregation technique has the additional benefit of causing the hosts which are member of an HS implicit message group to have their processing functions synchronized when they are executing the same interactive application. Many networked interactive applications are based on a simple overall three step operational model: wait for messages from other hosts, process the messages and the local users inputs to update the local application, send messages to the other hosts. This basic application loop is repeated at a rate fast enough to provide an interactive experience such as 5 to 30 times per second. It is desirable to keep such applications synchronized so that the states of the applications is consistent on the different host machines. When such applications communicate using the HS model of the present invention their operations will become natu-

26

rally synchronized. The HS ULP server process will wait until all of the members of the message group has completed their turns and sent a message to the group before sending the aggregated messages to the members of the group. This will cause the applications on the hosts to wait until they have received the aggregated messages. They will all then start processing these messages along with the local user inputs. Even if they perform their processing at different speeds and send their next messages to the group at different times, the HS ULP server will wait until all have completed their processing and reported in with a message to the group. This will keep all of the host applications synchronized in that every host will be at the same application loop iteration as all of the others. This will keep the application state consistent on all of the hosts. Only network propagation delays from the GMS to the hosts and different processing speeds of the hosts will cause the start and completion of their processing to begin at different times. It is not a requirement in networked applications to keep all of the hosts precisely synchronized, only that that application state is consistent. The HS method provides a natural way to do this in the context of the present invention.

Preferred Embodiment

The detailed description of the invention has described a diagram implementation of the invention as the best way to explain the invention. The preferred embodiment of the invention is as follows.

In the preferred embodiment, the wide area network is the Internet and the TLP protocol is TCP/IP. The GMS is a general purpose computer system connected to the Internet and the hosts are personal computers connected to the Internet.

TCP/IP provides a number of advantages that provide for a more efficient applications interface on the hosts. TCP/IP supports the concept of source and destination port numbers in its header. The ULP can make use of the port numbers to identify source and destination ULP connections. Most ULP send messages will be from hosts to a implicit ULP group addresses and most ULP receive messages will be from the implicit ULP addresses to the ULP host addresses. All of these and the ULP message type field can be represented by source and destination port addresses within the TCP/IP header. This means that for most ULP messages, the ULP message encapsulated within the TCP/IP message need only contain the payload. There is the slight complication of the aggregated ULP receive messages sent from a ULP server process to a hosts. Here the destination port will be the host the source port will be for the implicit ULP group address and the payload will still contain the source host ULP addresses in each the payload items.

TCP/IP also supports header compression for low speed dial-up lines which is also important in this application. See RFC 1144. TCP/IP is a connection oriented protocol which provides reliable end-to-end transport. It handles re-transmission on errors and fragmentation and reassembly of data transparently to upper level protocols. Header compression allows much of the TCP/IP header to be omitted with each packet to be replaced by a small connection identifier. This connection ID will uniquely define a connection consisting of a source and destination IP address and source and destination TCP/IP port numbers.

At the interface to the application on the hosts, the preferred embodiment of the ULP is as a session layer protocol. In the preferred embodiment the application on a host opens a session with a ULP server process. This session is identified with a unique session ID on the host. The host application then sends data to the ULP host interface 151

tagged with this session ID. The session ID defines a host and implicit ULP pair including the TCP/IP TLP address of the GMS server that is running the particular ULP server process for the implicit ULP address. By binding the transport address of the GMS of a ULP server process to the session ID, we can transparently to the application support multiple group messaging servers on the network and a single host can have multiple active sessions with different physical group messaging servers. This avoids any address space collision problems that could arise from the fact that the ULP address space is unique to each GMS.

Alternate Embodiments

One possible extension to the invention is to extend the ULP to support a common synchronized time base on the GMS and the hosts that are connected to it. This would be most interesting in context of the SI message aggregation mode. The SI time base on the GMS could be replicated on all of the hosts and all of the hosts and the GMS could lock these time bases together. There are known methods to synchronize time bases on multiple computer systems. One such method is called NTP.

Another extension to the invention is to define ULP server processes that perform specific application specific processing on the contents of the messages that are received. A variety of different application specific processing functions can be defined and implemented. A particular function would be selected by attributes provided in the create implicit group function. These functions could process the data in the message payloads and replace the data elements in the payloads with processed results. Separately, or in combination with processing the message payloads, the processing could store either raw message payload data in the application specific state storage area or could store processed results.

Clearly, the host system need not be personal computers, but could also be dedicated game consoles or television set top boxes or any other device with a programmable controller capable of implementing the ULP protocol. The wide area network used to transport the ULP protocol need not be the Internet or based on IP. Other networks with some means for wide area packet or datagram transport are possible including ATM networks or a digital cable television network.

The invention now being fully described, it will be apparent to one of ordinary skill in the art that any changes and modifications can be made thereto without departing from the spirit or scope of the invention as set forth herein.

Accordingly, the present invention is to be limited solely by the scope of the appended claims.

What is claimed is:

1. A method for providing group messages to a plurality of host computers connected over a unicast wide area communication network, comprising the steps of:

providing a group messaging server coupled to said network, said server communicating with said plurality of host computers using said unicast network and maintaining a list of message groups, each message group containing at least one host computer;

sending, by a plurality of host computers belonging to a first message group, messages to said server via said unicast network, said messages containing a payload portion and a portion for identifying said first message group;

aggregating, by said server in a time interval determined in accordance with a predefined criterion, said payload portions of said messages to create an aggregated payload;

forming an aggregated message using said aggregated payload; and

transmitting, by said server via said unicast network, said aggregated message to a recipient host computer belonging to said first message group.

2. The method of claim 1 wherein said time interval is a fixed period of time.

3. The method of claim 1 wherein said time interval corresponds to a time for said server to receive at least one message from each host computer belonging to said first message group.

4. The method of claim 1 further comprising the step of creating, by one of said plurality of host computers, said first message group by sending a first control message to said server via said unicast network.

5. The method of claim 4 further comprising the step of joining, by some of said plurality of host computers, said first message group by sending control messages via said unicast network to said server specifying said first message group.

6. The method of claim 1 wherein said network is Internet and said server communicates with said plurality of host computers using a session layer protocol.

* * * * *

CC-A-B

Claim Chart comparing Claims 1–6 of U.S. Patent No. 5,822,523 to the disclosure in Netrek

Prior art cited in this chart:

- *Server2.5pl4.tar.gz* (“*Server Code*”) and *BRMH-1.7.tar.gz* (“*Client Code*”) (source code dated no later than August 1994).
- The History of Netrek, Andy McFadden (“McFadden”) (January 1, 1994).

Claims of the ‘523 Patent	Disclosure in Netrek
<p>1. A method for providing group messages to a plurality of host computers connected over a unicast wide area communication network, comprising the steps of:</p>	<p>“Netrek is a real-time graphical multiplayer arcade/strategy game played over the Internet. Players form into teams and fight for control of the galaxy, dogfighting and taking planets” McFadden at § 0.2</p> <p>“In Netrek, every player has a client program that connects to the server.” McFadden at § 2.1.2</p> <p>“3.3.1 Client/Server Recall that Xtrek handled all rendering from the server side. The X10 traffic was sent over TCP sockets from the server to the player's display. Smith added network code that separated the game into distinct client and server components. Each player ran a client program that communicated with the server using a vastly simpler protocol. The client handled all rendering locally, so the bandwidth requirements were greatly reduced.” McFadden at § 3.3.1</p> <pre>00001390 updateMessages () 00001391 { [...] 00001590 }</pre> <p>Server\ntserv\socket.c at lines 1390-590</p> <pre>00000603 updateClient ()</pre>

	<pre> 00000604 { [...]</pre> <pre> 00000688 flushSockBuf(); 00000689 repCount++; 00000690 }</pre> <p>Server\ntserv\socket.c at lines 603-90</p> <pre> 00001537 sendServerPacket(packet) 00001538 /* Pick a random type for the packet */ 00001539 struct player_spacket *packet; 00001540 {</pre> <p>brmh-1.7/socket.c at lines 1537-1540</p> <pre> 00001856 doRead(asock) 00001857 int asock; 00001858 { 00002044 }</pre> <p>Server\ntserv\socket.c at lines 603-90</p> <pre> 00000523 struct mesg_cpacket { 00000524 char type; /* CP_MESSAGE */ 00000525 char group; 00000526 char indiv; 00000527 char pad1; 00000528 char mesg[80]; 00000529 };</pre> <p>brmh-1.7/packets.h at lines 523-29</p>
<p>providing a group messaging server coupled to said network, said server communicating with said plurality of host</p>	<p>"In Netrek, every player has a client program that connects to the server." McFadden at § 2.1.2</p> <p>"3.3.1 Client/Server Recall that Xtrek handled all rendering from the server side. The X10 traffic was sent over TCP sockets from the server to the player's display.</p>

<p>computers using said unicast network and maintaining a list of message groups, each message group containing at least one host computer;</p>	<p>Smith added network code that separated the game into distinct client and server components. Each player ran a client program that communicated with the server using a vastly simpler protocol. The client handled all rendering locally, so the bandwidth requirements were greatly reduced."</p> <p>McFadden at § 3.3.1</p> <p>" 1. newstartd - This waits around for a connection from a client. It then forks and execs a ntserv."</p> <p>Server\docs\README at lines 99-100</p> <pre> 00000129 while (1) { 00000130 if ((port_idx=connectionAttemptDetected(num_progs))<0){ 00000131 fprintf(stderr, "Whoops. Bye.\n"); 00000132 exit(0); 00000133 } [...] 00000143 if (fd!=-1) write(fd, logname, strlen(logname)); 00000144 sleep(2); 00000145 } 00000146 else if (fork() == 0) { /* we are a clone */ 00000147 time(&curtime); 00000148 sprintf(logname, " %-32.32s %s", 00000149 peerhostname, 00000150 ctime(&curtime)); 00000151 if (fd!=-1) write(fd, logname, strlen(logname)); 00000152 if (fd!=-1) close(fd); 00000153 00000154 pr=&(prog[port_idx]); 00000155 switch (pr->nargs) { 00000156 case 0: execl(pr->prog,pr->progrname, peerhostname, 0); 00000157 break; 00000158 case 1: execl(pr->prog,pr->progrname, pr->arg[0], peerhostname, 0); 00000159 break;</pre>
---	---

```

00000160 case 2: execl(pr->prog,pr->progname, pr->arg[0], pr-
>arg[1],
00000161         peerhostname, 0);
00000162         break;
00000163 case 3: execl(pr->prog,pr->progname, pr->arg[0], pr-
>arg[1],
00000164         pr->arg[2], peerhostname, 0);
00000165         break;
00000166 case 4: execl(pr->prog,pr->progname, pr->arg[0], pr-
>arg[1],
00000167         pr->arg[2], pr->arg[3], peerhostname, 0);
00000168         break;
00000169 default: ;
00000170 }
00000171 fprintf(stderr,"Error in execl! -- %s\n",pr->prog);
00000172 reporterror();
00000173 }

```

Server\ntserv\newstartd.c at lines 129-173

```

00000179 int connectionAttemptDetected(num_progs)
00000180 int num_progs;
00000181 {
00000182     [...]
00000214 if(bind(sock, &addr, sizeof(addr)) < 0){
00000215     [...]
00000237     }
00000238 }
00000239 }
00000240 if(listen(sock, 1)<0) {
00000241     fprintf(stderr,"Listen failed: ");
00000242     reporterror();
00000243     sock = -1;
00000244 }
00000245 prog[i].sock=sock;

```

```

00000246 /* close(0); */
00000247     fprintf (stderr,"listening on %d, connection will start
%s \"%s\" %s %s %s %s\n",
00000248         prog[i].port, prog[i].prog, prog[i].progname,
00000249         prog[i].arg[0], prog[i].arg[1], prog[i].arg[2],
prog[i].arg[3]);
00000250     fflush (stderr);
00000251     }
00000252     }
00000253
00000254     while (1) /* Wait for a connection */
00000289         newsock = accept(sock, &naddr, &len);
00000290         if ((newsock < 0) && (errno == EINTR))
00000291             goto nointr;
00000292
00000293         if(newsock < 0){
00000294             fprintf(stderr,"accept error!");
00000295             reporterror();
00000296             fprintf(stderr,"No one calling!\n");
00000297             shutdown(sock, 2);
00000298             close(sock);
00000299             prog[i].sock = -1;
00000300             return(-1);
00000301         } else {
00000302             if (newsock != 0) {
00000303                 if (dup2(newsock, 0) == -1) {
00000304                     fprintf(stderr, "failed dup2\n");
00000305                     reporterror();
00000306                 }
00000307                 close(newsock);
00000308             }
00000309             return(i);
00000310         }
00000311     }

```

Server\newstartd\newstartd.c at lines 179-311

```
00000135  if (callHost) {
00000136      if (!connectToClient(host,xtrekPort)) {
00000137          exit(0);
00000138      }
00000139  } else {
00000140      sock=0;    /* Because we were forked by inetd! */
00000141      checkSocket();
00000142      initClientData();    /* "normally" called by
connectToClient() */
00000143  }
```

Server\ntserv\main.c at lines 135-43

```
00000442 connectToClient(machine, port)
00000443 char *machine;
00000444 int port;
00000445 {
[...]
00000452  if (sock!=-1) {
00000453  shutdown(sock,2);
00000454  sock= -1;
00000455  }
00000456  ERROR(3,("Connecting to %s through %d\n", machine, port));
00000457
00000458  if ((ns=socket(AF_INET, SOCK_STREAM, 0)) < 0) {
00000459      ERROR(1,("I cannot create a socket\n"));
00000460      exit(2);
00000461  }
[...]
00000479  if (connect(ns, &addr, sizeof(addr)) < 0) {
00000480  ERROR(3,("I cannot connect through port %d\n", port));
00000481  close(ns);
00000482  return(0);
```

```

00000483     }
[...
00000488 }
Server\ntserv\socket.c at lines 442-88

00001747 flushSockBuf()
00001748 {
[...
00001755 if (gwrite(sock, buf, t) != t) {
00001756     perror("std flush gwrite failed, client marked dead");
00001757     clientDead=1;
00001758 }
[...
00001782 if (gwrite(udpSock, udpbuf, t) != t){
00001783     perror("UDP flush gwrite failed, client marked dead once
more");
[...
00001791 }
[...
00001802 }
Server\ntserv\socket.c at lines 1747-802

00002607 gwrite(fd, wbuf, size)
00002608 int fd;
00002609 char *wbuf;
00002610 size_t size;
00002611 {
[...
00002625     while (bytes>0) {
00002626     n = write(fd, wbuf, bytes);
00002627     if (count++ > 100) {
00002628         ERROR(1,("Gwrite hosed: too many writes
(%d)\n",getpid()));
00002629         clientDead = 1;

```

```
00002630     return (-1);
00002631 }
[...]
00002671     }
00002672     return(orig);
00002673 }
```

Server\ntserv\socket.c at lines 2607-73

```
struct mesg_spacket {
    char type;      /* SP_MESSAGE */
    u_char m_flags;
    u_char m_recpt;
    u_char m_from;
    char mesg[MSG_LEN];
};
```

Server\ntserv\packet.h at lines 184-190

```
00000471 struct memory {
00000472     struct player  players[MAXPLAYER];
00000473     struct torp    torps[MAXPLAYER * MAXTORP];
00000474     struct plasmatorp  plasmatorps[MAXPLAYER * MAXPLASMA];
00000475     struct status  status[1];
00000476     struct planet  planets[MAXPLANETS];
00000477     struct phaser  phasers[MAXPLAYER];
00000478     struct mctl    mctl[1];
00000479     struct message messages[MAXMESSAGE];
00000480     struct team    teams[MAXTEAM + 1];
00000481     struct ship    shipvals[NUM_TYPES];
00000482 };
```

Server\ntserv\struct.h at lines 471-82

```
00000208 struct player {
[...]
00000218     int p_x;
```



```

00000219     int p_y;
[...]
```

```

00000226     short p_team;           /* Team I'm on */
[...]
```

```

00000260 #ifdef FULL_HOSTNAMES
00000261     char p_full_hostname[32]; /* full hostname 4/13/92 TC */
00000262 #endif
00000263 #ifdef PING
00000264     int p_avrt;           /* average round trip time */
00000265     int p_stdv;           /* standard deviation in round
trip time */
00000266     int p_pkls_c_s;       /* packet loss (client to
server) */
00000267     int p_pkls_s_c;       /* packet loss (server to
client) */
00000268 #endif
00000269 #ifdef DS
00000270     int p_timerdelay;     /* updates per second */
00000271     pid_t p_process;     /* process id number */
00000272 #endif
[...]
```

```

00000284 };
Server\ntserv\struct.h at lines 208-84
```

```

00000120 /* These are the teams */
[...]
```

```

00000132 #define ALLTEAM (FED|ROM|KLI|ORI)
00000133 #define MAXTEAM (ORI)           /* was ALLTEAM (overkill?)
6/22/92 TMC */
00000134 #define NUMTEAM 4
Server\ntserv\defs.h at lines 120-134
```

```

00001125 updateTorps()
00001126 {
```

```

[...]
```

```

00001132     for (i=0, torp=torps, tpi=clientTorpsInfo, tp=clientTorps;
00001133         i<MAXPLAYER*MAXTORP;
00001134         i++, torp++, tpi++, tp++) {
[...]
```

```

00001142         sendClientPacket(tpi);
[...]
```

```

00001151         sendClientPacket(tp);
[...]
```

```

00001191     }
00001192 }
Server\ntserv\socket.c at lines 1125-92
```

```

00001194 updatePlasmas()
00001195 {
[...]
```

```

00001201     for (i=0, torp=plasmatorps, tpi=clientPlasmasInfo,
tp=clientPlasmas;
00001202         i<MAXPLAYER*MAXPLASMA;
00001203         i++, torp++, tpi++, tp++) {
[...]
```

```

00001211         sendClientPacket(tpi);
[...]
```

```

00001219         sendClientPacket(tp);
[...]
```

```

00001254     }
00001255 }
Server\ntserv\socket.c at lines 1194-255
```

```

00001257 updatePhasers()
00001258 {
[...]
```

```

00001264     for (i=0, ph=clientPhasers, phase=phasers, pl=players;
00001265         i<MAXPLAYER; i++, ph++, phase++, pl++) {
```

```

[...]
00001274     sendClientPacket(ph);
[...]
00001290     sendClientPacket(ph);
[...]
00001293     }
00001294 }
Server\ntserv\socket.c at lines 1257-94

00001390 updateMessages()
00001391 {
[...]
00001563     if (cur->m_from==DOOSHMSG) msg.m_from=255; /* god */
00001564     if ((cur->m_from < 0) || (cur->m_from > MAXPLAYER))
00001565         sendClientPacket((CVOID) &msg);
00001566     else if (cur->m_flags & MALL && !(ignored[cur->m_from]
& MALL))
00001567         sendClientPacket((CVOID) &msg);
00001568     else if (cur->m_flags & MTEAM && !(ignored[cur->m_from]
& MTEAM)){
00001569         sendClientPacket((CVOID) &msg);
00001570     }
00001571     else if (cur->m_flags & MINDIV) {
[...]
00001584         sendClientPacket((CVOID) &msg);
[...]
00001586     }
[...]
00001590 }
Server\ntserv\socket.c at lines 1390-590

00000191     me = &players[pno];
[...]
00000222     updateSelf(); /* so he gets info on who he is */

```

```

00000223     updateShips();
00000224     updatePlanets();
00000225     flushSockBuf();
00000226
00000227     /* Get login name */
00000228
00000229     if ((pwent = getpwuid(getuid())) != NULL)
00000230 STRNCPY(login, pwent->pw_name, NAME_LEN);
00000231     else
00000232 STRNCPY(login, "Bozo", NAME_LEN);
00000233     login[NAME_LEN - 1] = '\0';
[...]
```

```

00000238     me->p_team=ALLTEAM;
[...]
```

```

00000273     me->p_full_hostname[sizeof(me->p_full_hostname) - 1] =
'\0';
[...]
```

```

00000296 /* give the player the motd and find out which team he wants
*/
00000297     if (me->p_status != PALIVE) {
00000298 me->p_x= -100000;
00000299 me->p_y= -100000;
00000300 updateSelf();
00000301 updateShips();
00000302 teamPick= -1;
00000303 flushSockBuf();
00000304 getEntry(&team, &s_type);
00000305 repCount=0;          /* Make sure he gets an update
immediately */
00000306     }
[...]
```

```

00000325     enter(team, 0, pno, s_type);
Server\ntserv\main.c at lines 183-325
```

```

00000034 enter(tno, disp, pno, s_type)
00000035 int tno;
00000036 int disp;          /* not used, so I used it 7/27/91 TC */
00000037 int pno;
00000038 int s_type;
00000039 {
[...]
```

```

00000056     STRNCPY(me->p_name, pseudo, NAME_LEN);
00000057     me->p_name[NAME_LEN - 1] = '\0';
00000058     getship(myship, s_type);
00000059
00000060     /* Alert client about new ship stats */
[...]
```

```

00000085     if ((s_type != STARBASE) && (s_type != ATT) && plkills>0)
    {
00000086     me->p_ship.s_plasmacost = -1;
00000087     }
00000088     me->p_updates = 0;
00000089     me->p_flags = PFSHIELD;
00000090     if (s_type==STARBASE) me->p_flags |= PFDOCKOK;
00000091     me->p_dir = 0;
00000092     me->p_desdir = 0;
00000093     me->p_speed = 0;
00000094     me->p_desspeed = 0;
00000095     me->p_subspeed = 0;
00000096     if ((tno == 4) || (tno == 5)) { /* change 5/10/91 TC new
case, indep */
00000097     me->p_team = 0;
00000098     placeIndependent(); /* place away from others 1/23/92 TC */
00000099     }
00000100     else {
00000101     me->p_team = (1 << tno);
00000102     for (;;) {
00000103         startplanet=tno*10 + random() % 10;

```

```

00000104     if (startplanets[startplanet]) break;
00000105 }
[...]
```

```

00000132 /*     if (!keeppeace) me->p_hostile = (FED|ROM|KLI|ORI);*/
00000133     if (!keeppeace) auto_peace();
00000134     me->p_hostile &= ~me->p_team;
00000135
00000136     /* join message stuff */
00000137
00000138     sprintf(me->p_mapchars, "%c%c", teamlet[me->p_team],
shipnos[me->p_no]);
00000139     if (lastteam != tno || lastrank != mystats->st_rank) {
00000140
[...]
```

```

00000187 #ifndef FULLHOSTNAMES
00000188     pmessage2(0, MALL | MJOIN, addrbuf, me->p_no,
00000189         "%.16s (%2.2s) promoted to %s (%.16s@%.16s)",
00000190         me->p_name,
00000191         me->p_mapchars,
00000192         ranks[me->p_stats.st_rank].name,
00000193         me->p_login,
00000194         me->p_monitor);
00000195 #else
00000196     pmessage2(0, MALL | MJOIN, addrbuf, me->p_no,
00000197         "%.16s (%2.2s) promoted to %s (%.16s@%.32s)",
00000198         me->p_name,
00000199         me->p_mapchars,
00000200         ranks[me->p_stats.st_rank].name,
00000201         me->p_login,
00000202         me->p_full_hostname);
00000203 #endif
00000204 }
[...]
```

```

00000232 }
```

	Server\ntserv\enter.c at lines 30-232
<p>sending, by a plurality of host computers belonging to a first message group, messages to said server via said unicast network, said messages containing a payload portion and a portion for identifying said first message group;</p>	<pre> 00001537 sendServerPacket(packet) 00001538 /* Pick a random type for the packet */ 00001539 struct player_spacket *packet; 00001540 { [...] 00001554 if (commMode == COMM_UDP) { 00001555 /* for now, just sent everything via TCP */ 00001556 } 00001557 if (commMode == COMM_TCP !udpClientSend) { 00001558 /* special case for verify packet */ 00001559 if (packet->type == CP_UDP_REQ) { 00001560 if (((struct udp_req_cpacket *) packet)->request == COMM_VERIFY) 00001561 goto send_udp; 00001562 } 00001563 /* 00001564 * business as usual (or player has turned off UDP transmission) 00001565 */ 00001566 if (gwrite(sock, (char *) packet, size) != size) { 00001567 printf("gwrite failed. Server must be dead\n"); 00001568 serverDead = 1; 00001569 } [...] 00001610 if (gwrite(udpSock, packet, size) != size) { [...] 00001623 } [...] 00001628 if (gwrite(sock, (char *) packet, size) != size) { 00001629 printf("gwrite failed. Server must be dead\n"); 00001630 serverDead = 1; 00001631 } </pre>

```

[...]
00001633     }
00001634 }
brmh-1.7/socket.c at lines 1537-634

00000026 struct player *me = NULL;
brmh-1.7/data.c at line 26

00000134 struct player {
[...]
00000144     int             p_x;
00000145     int             p_y;
[...]
00000152     short          p_team;    /* Team I'm on */
[...]
00000192 };
brmh-1.7/struct.h at lines 134-92

00000523 struct mesg_cpacket {
00000524     char             type;    /* CP_MESSAGE */
00000525     char             group;
00000526     char             indiv;
00000527     char             pad1;
00000528     char             mesg[80];
00000529 };
brmh-1.7/packets.h at lines 523-29

00000222 #define sendTorpReq(dir) sendShortPacket(CP_TORP, dir)
brmh-1.7/defs.h at line 222

00000293 struct torp_cpacket {
00000294     char             type;    /* CP_TORP */
00000295     unsigned char    dir;      /* direction to fire torp */
00000296     char             pad1;

```



```
0000297 char pad2;
0000298 };
```

brmh-1.7\packets.h at lines 293-99

```
0000121 struct packet_handler handlers[] = {
[...]
```

```
0000128     { sizeof(struct torp_cpacket), handleTorpReq },
[...]
```

```
0000194 #ifdef FEATURE_PACKETS
0000195     { sizeof(struct feature_cpacket), handleFeature },
0000196 #endif
0000197 };
```

Server\ntserv\socket.c at lines 121-97

```
0000167     /* Check to see if the handler is there and the request
is legal.
0000168     * The code is a little ugly, but it isn't too bad to
worry about
0000169     * yet.
0000170     */
0000171     packetsReceived[*bufptr]++;
0000172 #ifdef PING
0000173         if(asock == udpSock)
0000174             packets_received ++;
0000175 #endif
0000176     if (handlers[*bufptr].handler != NULL) {
0000177         if (((FD_ISSET(*bufptr, &inputMask)) &&
0000178             (me==NULL || !(me->p_flags & (PFWAR|PFREFITTING
0000179 #ifdef SB_TRANSWARP
0000180             | PFTWARP
0000181 #endif
0000182             )))) ||
0000183             *bufptr==CP_RESETSTATS || *bufptr==CP_UPDATES ||
0000184             *bufptr==CP_OPTIONS || *bufptr==CP_RESERVED ||
```

```

00001985 #ifdef PING                /* ping response always valid */
00001986                            *bufptr==CP_PING_RESPONSE ||
00001987 #endif
00001988 #ifdef RSA                      /* NEW -- fix ghostbust problem
*/
00001989                            *bufptr== CP_RSA_KEY ||
00001990 #endif
00001991 #ifdef FEATURE_PACKETS
00001992                            *bufptr == CP_FEATURE ||
00001993 #endif
00001994 #ifdef MESSAGES_ALL_TIME /* off for the moment */
00001995                            *bufptr == CP_MESSAGE ||
00001996 #ifdef SHORT_PACKETS
00001997                            *bufptr == CP_S_MESSAGE ||
00001998 #endif
00001999 #endif
00002000
00002001                            *bufptr==CP_SOCKET || *bufptr==CP_BYE) {
00002002                            if (me && me->p_flags & PFSELFDEST
00002003 #ifdef PING /* don't let it undo self destruct */
00002004                                && *bufptr != CP_PING_RESPONSE
00002005 #endif
00002006                                ) {
00002007                                me->p_flags &= ~PFSELFDEST;
00002008                                new_warning(85,"Self Destruct has been
canceled");
00002009                                }
00002010                                (*(handlers[*bufptr].handler))(bufptr);
00002011                                }
Server\ntserv\socket.c at lines 1976-2011

00002046 handleTorpReq(packet)
00002047 struct torp_cpacket *packet;

```

```

00002048 {
00002049     ntorp(packet->dir, TMOVE);
00002050
00002051
00002052 }
Server\ntserv\socket.c at lines 2046-50

00000041 ntorp(course, type)
00000042 u_char course;
[...]
```

```

00000048     if (me->p_flags & PFWEPE) {
00000049         new_warning(25, "Torpedo launch tubes have exceeded
maximum safe temperature!");
00000050         return;
00000051     }
[...]
```

```

00000073     if (me->p_ntorp == MAXTORP) {
00000074         new_warning(26, "Our computers limit us to having 8
live torpedos at a time captain!");
00000075         return;
00000076     }
00000077     if (me->p_fuel < myship->s_torpcost) {
00000078         new_warning(27, "We don't have enough fuel to fire
photon torpedos!");
00000079         return;
00000080     }
00000081     if (me->p_flags & PFREPAIR) {
00000082         new_warning(28, "We cannot fire while our vessel is in
repair mode.");
00000083         return;
00000084     }
00000085     if ((me->p_cloakphase) && (me->p_ship.s_type != ATT)) {
00000086         new_warning(29, "We are unable to fire while in cloak,
```

```

captain!");
00000087     return;
00000088     }
00000089
00000090 /* change TC 12/9/90 -- my attempt at torp angle stuff */

00000091
00000092
00000093     if (topgun && ((me->p_ship).s_type != STARBASE)) {

00000094         int delta;
00000095         if ((delta = ((int) me->p_dir - (int) course)) < 0)

00000096             delta = -delta;
00000097         if ((delta > topgun) && (delta < (256 - topgun))) {

00000098             /* note: 128 = 180 degrees left/right */
00000099             new_warning(30,"We only have forward mounted
cannons.");
00000100             return;
00000101         }
00000102     } /* end if topgun */
[...]
00000112
00000113     /* Setup data in new torp */
00000114     if (type>TSTRAIGHT || type<TFREE) type=TMOVE;
00000115     k->t_no = i;
00000116     k->t_status = type;
00000117     k->t_owner = me->p_no;
00000118     k->t_team = me->p_team;
00000119     k->t_x = me->p_x;
00000120     k->t_y = me->p_y;
00000121     k->t_dir = course;
00000122     k->t_damage = myship->s_torpdamage;

```

```

00000123     if (vectortorps)
00000124         k->t_speed = vector_torp_speed(me->p_dir, me->p_speed,
course,
00000125             myship->s_torpspeed);
00000126     else
00000127         k->t_speed = myship->s_torpspeed;
00000128     k->t_war = me->p_hostile | me->p_swar;
00000129     k->t_fuse = myship->s_torpfuse + (random() % 20);

00000130     k->t_turns = myship->s_torpturns;
00000131     k->t_whodet = NODET;
00000132 }

```

Server\ntserv\torp.c at lines 41-132

```

00001161 udtorps()
00001162 {
00001163     register int i, turn=0, heading=0;
00001164     register struct torp *j;
00001165
00001166     for (i = 0, j = &torps[i]; i < MAXPLAYER * MAXTORP; i++,
j++) {
00001167         switch (j->t_status) {
00001168             case TFREE:
00001169                 continue;
00001170             case TMOVE:
00001171             case TSTRAIGHT:
00001172                 if (j->t_turns > 0) {
[...]
```

Server\ntserv\daemonII.c at lines 1161-1246

```

00001125 updateTorps()

```

	<pre> 00001126 { [...]</pre> <pre> 00001132 for (i=0, torp=torps, tpi=clientTorpsInfo, tp=clientTorps; 00001133 i<MAXPLAYER*MAXTORP; 00001134 i++, torp++, tpi++, tp++) { [...]</pre> <pre> 00001142 sendClientPacket(tpi); [...]</pre> <pre> 00001151 sendClientPacket(tp); [...]</pre> <pre> 00001191 } 00001192 } Server\ntserv\socket.c at lines 1125-92 00001125 updateTorps() 00001126 { [...]</pre> <pre> 00001132 for (i=0, torp=torps, tpi=clientTorpsInfo, tp=clientTorps; 00001133 i<MAXPLAYER*MAXTORP; 00001134 i++, torp++, tpi++, tp++) { [...]</pre> <pre> 00001142 sendClientPacket(tpi); [...]</pre> <pre> 00001151 sendClientPacket(tp); [...]</pre> <pre> 00001191 } 00001192 } Server\ntserv\socket.c at lines 1125-92 </pre>
<p>aggregating, by said server in a time interval determined in accordance with a</p>	<pre> 00000076 int timerDelay=200000; /* delay between sending stuff to client */ Server\ntserv\data.c at line 76 </pre>

predefined criterion, said payload portions of said messages to create an aggregated payload;	<pre> 00000195 readFromClient(); Server\ntserv\input.c at line 195 00000152 input() 00000153 { 00000154 struct itimerval udt; 00000155 fd_set readfds; 00000156 static struct timeval poll = {2, 0}; 00000157 00000158 #ifdef DS 00000159 if (!me->p_process) 00000160 #endif 00000161 { 00000162 udt.it_interval.tv_sec = 0; 00000163 udt.it_interval.tv_usec = timerDelay; 00000164 udt.it_value.tv_sec = 0; 00000165 udt.it_value.tv_usec = timerDelay; 00000166 setitimer(ITIMER_REAL, &udt, 0); 00000167 } 00000168 SIGNAL(SIGALRM, setflag); 00000169 00000170 /* Idea: read from client often, send to client not so often */ 00000171 while (1) { [...]</pre> <pre> 00000195 readFromClient(); [...]</pre> <pre> 00000203 } 00000204 } Server\ntserv\input.c at lines 152-203 00000076 int timerDelay=200000; /* delay between sending stuff to client */ Server\ntserv\data.c at line 76 </pre>
--	--

```

00000603 updateClient()
00000604 {
[...]
00000608     static int skip = 0; /* If skip is set we skip next update
*/
00000609 /* This can halve your updates */
00000610     if (send_short && skip) {
00000611         skip = 0; /* back to default */
00000612     if (bufptr==buf && (commMode!=COMM_UDP || udpbufptr==buf))
{
00000613         /* We sent nothing! We better send something to wake
him */
00000614         if (me->p_fuel < 61000)
00000615             sendClientPacket((CVOID) &clientSelfShort);
00000616         else
00000617             sendClientPacket((CVOID) &clientSelf);
00000618     }
00000619     flushSockBuf();
00000620     repCount++;
00000621     return;
00000622 }
[...]
00000630     if(send_short){
00000631     updatePlasmas();
00000632     updateStatus();
00000633     updateSelf();
00000634     updatePhasers();
00000635     updateShips();
00000636     updateTorps();
00000637     updatePlanets();
00000638     updateMessages();
00000639     }
[...]
```



```

00000657         if(send_short && (me->p_fuel < 61000))
00000658             sendClientPacket((CVOID) &clientSelfShort);
00000659         else
00000660 #endif
00000661 sendClientPacket((CVOID) &clientSelf);
00000662     }
[...
00000685         sendClientPing();                               /* ping.c */
00000686 #endif
00000687
00000688     flushSockBuf();
00000689     repCount++;
00000690 }

```

Server\ntserv\socket.c at lines 603-90

```

00000052     intrupt();

```

Server\ntserv\input.c at lines 52

```

00000197     intrupt();

```

Server\ntserv\input.c at lines 197

```

00001390 updateMessages()
00001391 {
[...
00001590 }

```

Server\ntserv\socket.c at lines 1390-590

```

00001825 readFromClient()
00001826 {
[...
00001838     if (select(32,&readfds,0,0,&timeout) != 0) {
00001839 /* Read info from the xtrem client */
00001840 if (FD_ISSET(sock, &readfds)) {
00001841     retval += doRead(sock);

```

```

00001842 }
00001843 if (udpSock >= 0 && FD_ISSET(udpSock, &readfds)) {
00001844     V_UDPDIAG(("Activity on UDP socket\n"));
00001845     retval += doRead(udpSock);
00001846 }
00001847 }
00001848 return (retval != 0);           /* convert to 1/0 */
00001849 }
00001850
00001851 [...]
00001855 /* ripped out of above routine */
00001856 doRead(asock)
00001857 int asock;
00001858 {
00001859     struct timeval timeout;
00001860     [...]
00001877     /* Read info from the xtrek server */
00001878     count=read(asock,buf,BUFSIZ*2);
00001879     [...]
00001916     bufptr=buf;
00001917     while (bufptr < buf+count) {
00001918         [...]
00001939         while (size>count+(buf-bufptr)) {
00001940             /* We wait for up to twenty seconds for rest of packet.
00001941              * If we don't get it, we assume the client
00001942              * died.
00001943              */
00001944             timeout.tv_sec=20;
00001945             timeout.tv_usec=0;
00001946             /*readfds=1<<asock;*/
00001947             FD_ZERO(&readfds);
00001948             FD_SET(asock, &readfds);
00001949             [...]
00001956             temp=read(asock,buf+count,size-(count+(buf-bufptr)));

```

```

[...]  

00001966     }  

[...]  

00002010         (*(handlers[*bufptr].handler)) (bufptr);  

00002011     }  

00002012     /* Otherwise we ignore the request */  

00002013 } else {  

00002014     ERROR(1,("Handler for packet %d not installed...\n",  

*bufptr));  

00002015 }  

00002016     bufptr+=size;  

00002017     if (bufptr>buf+BUFSIZ) {  

00002018         bcopy(buf+BUFSIZ, buf, BUFSIZ);  

00002019         if (count==BUFSIZ*2) {  

00002020             /*readfds = 1<<asock;*/  

00002021             FD_ZERO(&readfds);  

00002022             FD_SET(asock, &readfds);  

00002023             if (select(32,&readfds,0,0,&timeout)) {  

00002024                 temp=read(asock,buf+BUFSIZ,BUFSIZ);  

00002025                 count=BUFSIZ+temp;  

[...]  

00002034         } else {  

00002035             count=BUFSIZ;  

00002036         }  

00002037         } else {  

00002038             count -=BUFSIZ;  

00002039         }  

00002040             bufptr-=BUFSIZ;  

00002041         }  

00002042     }  

00002043     return(1);  

00002044 }

```

Server\ntserv\socket.c at lines 1825-2044

```

00001825 readFromClient()
00001826 {
[...]
```

```

00001838     if (select(32,&readfds,0,0,&timeout) != 0) {
00001839 /* Read info from the xtrek client */
00001840 if (FD_ISSET(sock, &readfds)) {
00001841     retval += doRead(sock);
00001842 }
00001843 if (udpSock >= 0 && FD_ISSET(udpSock, &readfds)) {
00001844     V_UDPDIAG("Activity on UDP socket\n");
00001845     retval += doRead(udpSock);
00001846 }
00001847 }
00001848 return (retval != 0);           /* convert to 1/0 */
00001849 }
00001850
[...]
```

```

00001855 /* ripped out of above routine */
00001856 doRead(asock)
00001857 int asock;
00001858 {
00001859     struct timeval timeout;
[...]
```

```

00001877     /* Read info from the xtrek server */
00001878     count=read(asock,buf,BUFSIZ*2);
[...]
```

```

00001916     bufptr=buf;
00001917     while (bufptr < buf+count) {
[...]
```

```

00001939     while (size>count+(buf-bufptr)) {
00001940         /* We wait for up to twenty seconds for rest of packet.
00001941            * If we don't get it, we assume the client
00001942            died.
            */

```

```

00001943     timeout.tv_sec=20;
00001944             timeout.tv_usec=0;
00001945             /*readfds=1<<asock;*/
00001946     FD_ZERO(&readfds);
00001947     FD_SET(asock, &readfds);
00001948     [...]
00001956     temp=read(asock,buf+count,size-(count+(buf-bufptr)));
00001957     [...]
00001966     }
00001967     [...]
00002010             (*handlers[*bufptr].handler)(bufptr);
00002011     }
00002012     /* Otherwise we ignore the request */
00002013     } else {
00002014     ERROR(1,("Handler for packet %d not installed...\n",
*bufptr));
00002015     }
00002016     bufptr+=size;
00002017     if (bufptr>buf+BUFSIZ) {
00002018         bcopy(buf+BUFSIZ, buf, BUFSIZ);
00002019         if (count==BUFSIZ*2) {
00002020             /*readfds = 1<<asock;*/
00002021             FD_ZERO(&readfds);
00002022             FD_SET(asock, &readfds);
00002023             if (select(32,&readfds,0,0,&timeout)) {
00002024                 temp=read(asock,buf+BUFSIZ,BUFSIZ);
00002025                 count=BUFSIZ+temp;
00002026             [...]
00002034             } else {
00002035                 count=BUFSIZ;
00002036             }
00002037             } else {
00002038                 count -=BUFSIZ;
00002039             }

```

```
00002040             bufptr-=BUFSIZ;
00002041             }
00002042         }
00002043         return(1);
2044
```

Server\ntserv\socket.c at lines 1825-2044

```
00001390 updateMessages()
00001391 {
[...]
```

Server\ntserv\socket.c at lines 1390-590

```
00001603 sendClientPacket(packet)
00001604 /* Pick a random type for the packet */
00001605 struct player_spacket *packet;
00001606 {
[...]
```

```
00001618     /*
00001619      * If we're dead, dying, or just born, we definitely want
the transmission
00001620      * to get through (otherwise we can get stuck). I don't
think this will
00001621      * be a problem for anybody, though it might hang for a
bit if the TCP
00001622      * connection is bad.
00001623      */
00001624     /* Okay, now I'm not so sure. Whatever. */
00001625     if (oldstatus != PALIVE || (me != NULL && me->p_status !=
PALIVE))
00001626     orig_type = packet->type | 0x80; /* pretend it's critical
*/
00001627 #endif
00001628     if (packet->type<1 || packet->type>NUM_SIZES ||
```

```

00001629     sizes[(int)packet->type]==0) {
00001630     ERROR(1,("Attempt to send strange packet %d %d\n", packet-
>type,NUM_SIZES));
00001631     return;
00001632     }
00001633     packetsSent[(int)packet->type]++;
00001634     if (commMode == COMM_TCP || (commMode == COMM_UDP &&
udpMode == MODE_TCP)) {
00001635     /*
00001636     * business as usual
00001637     */
    [...]
00001647     bcopy(packet, bufptr, size);
00001648     bufptr+=size;
00001649
00001650     } else {
00001651     /*
00001652     * do UDP stuff unless it's a "critical" packet
00001653     * (note that both kinds get a sequence number appended)
(FIX)
00001654     */
    [...]
00001728     default:
00001729         /* these are critical packets; send them via TCP */
00001730         size=sizes[packet->type];
00001731         if (bufptr-buf+size >= BUFSIZE) {
00001732             t=bufptr-buf;
00001733             if (gwrite(sock, buf, t) != t) {
00001734                 perror("TCP gwrite failed, client marked dead");
00001735                 clientDead=1;
00001736             }
00001737             bufptr=buf /*+ addSequence(buf)*/;
00001738         }
00001739         bcopy(packet, bufptr, size);

```

	<pre> 00001740 bufptr+=size; 00001741 break; 00001742 } 00001743 } 00001744 } Server\ntserv\socket.c at lines 1603-744 00001125 updateTorps() 00001126 { [...]</pre> <pre> 00001132 for (i=0, torp=torps, tpi=clientTorpsInfo, tp=clientTorps; 00001133 i<MAXPLAYER*MAXTORP; 00001134 i++, torp++, tpi++, tp++){ [...]</pre> <pre> 00001142 sendClientPacket(tpi); [...]</pre> <pre> 00001151 sendClientPacket(tp); [...]</pre> <pre> 00001191 } 00001192 } Server\ntserv\socket.c at lines 1125-92 </pre>
forming an aggregated message using said aggregated payload; and	<pre> 00001747 flushSockBuf() 00001748 { [...]</pre> <pre> 00001755 if (gwrite(sock, buf, t) != t) { 00001756 perror("std flush gwrite failed, client marked dead"); 00001757 clientDead=1; 00001758 } [...]</pre> <pre> 00001782 if (gwrite(udpSock, udpbuf, t) != t){ 00001783 perror("UDP flush gwrite failed, client marked dead once more"); 00001784 #ifdef EXTRA_GB </pre>


```

00001785     clientDead=1;
00001786 #endif
00001787     UDPDIAG("*** UDP disconnected for %s\n", me->p_name));
00001788     printUdpInfo();
00001789     closeUdpConn();
00001790     commMode = COMM_TCP;
00001791 }
[...
00001802 }
Server\ntserv.c at lines 1747-802

00002607 gwrite(fd, wbuf, size)
00002608 int fd;
00002609 char *wbuf;
00002610 size_t size;
00002611 {
00002625     while (bytes>0) {
00002626     n = write(fd, wbuf, bytes);
[...
00002671     }
00002672     return(orig);
00002673 }
Server\ntserv.c at lines 2607-73

00000603 updateClient()
00000604 {
[...
00000608     static int skip = 0; /* If skip is set we skip next update
*/
00000609 /* This can halve your updates */
00000610     if (send_short && skip) {
00000611         skip = 0; /* back to default */
00000612     if (bufptr==buf && (commMode!=COMM_UDP || udpbufptr==buf))
{

```

```

00000613      /* We sent nothing!  We better send something to wake
him */
00000614      if (me->p_fuel < 61000)
00000615          sendClientPacket((CVOID) &clientSelfShort);
00000616      else
00000617          sendClientPacket((CVOID) &clientSelf);
00000618  }
00000619      flushSockBuf();
00000620      repCount++;
00000621      return;
00000622  }
[...]
00000630      if(send_short){
00000631          updatePlasmas();
00000632          updateStatus();
00000633          updateSelf();
00000634          updatePhasers();
00000635          updateShips();
00000636          updateTorps();
00000637          updatePlanets();
00000638          updateMessages();
00000639      }
[...]
00000657      if(send_short && (me->p_fuel < 61000))
00000658          sendClientPacket((CVOID) &clientSelfShort);
00000659      else
00000660 #endif
00000661  sendClientPacket((CVOID) &clientSelf);
00000662  }
[...]
00000685      sendClientPing();          /* ping.c */
00000686 #endif
00000687
00000688      flushSockBuf();

```

```

00000689     repCount++;
00000690 }
Server\ntserv\socket.c at lines 603-90

00001603 sendClientPacket(packet)
00001604 /* Pick a random type for the packet */
00001605 struct player_spacket *packet;
00001606 {
    [...]
00001618     /*
00001619      * If we're dead, dying, or just born, we definitely want
the transmission
00001620      * to get through (otherwise we can get stuck). I don't
think this will
00001621      * be a problem for anybody, though it might hang for a
bit if the TCP
00001622      * connection is bad.
00001623      */
00001624     /* Okay, now I'm not so sure. Whatever. */
00001625     if (oldstatus != PALIVE || (me != NULL && me->p_status !=
PALIVE))
00001626     orig_type = packet->type | 0x80; /* pretend it's critical
*/
00001627 #endif
00001628     if (packet->type<1 || packet->type>NUM_SIZES ||
00001629         sizes[(int)packet->type]==0) {
00001630     ERROR(1,("Attempt to send strange packet %d %d\n", packet-
>type,NUM_SIZES));
00001631     return;
00001632     }
00001633     packetsSent[(int)packet->type]++;
00001634     if (commMode == COMM_TCP || (commMode == COMM_UDP &&
udpMode == MODE_TCP)) {
00001635     /*

```

	<pre> 00001636 * business as usual 00001637 */ [...] 00001647 bcopy(packet, bufptr, size); 00001648 bufptr+=size; 00001649 00001650 } else { 00001651 /* 00001652 * do UDP stuff unless it's a "critical" packet 00001653 * (note that both kinds get a sequence number appended) (FIX) 00001654 */ [...] 00001728 default: 00001729 /* these are critical packets; send them via TCP */ 00001730 size=sizes[packet->type]; 00001731 if (bufptr-buf+size >= BUFSIZE) { 00001732 t=bufptr-buf; 00001733 if (gwrite(sock, buf, t) != t) { 00001734 perror("TCP gwrite failed, client marked dead"); 00001735 clientDead=1; 00001736 } 00001737 bufptr=buf /*+ addSequence(buf)*/; 00001738 } 00001739 bcopy(packet, bufptr, size); 00001740 bufptr+=size; 00001741 break; 00001742 } 00001743 } 00001744 } Server\ntserv\socket.c at lines 1603-744 </pre>
transmitting, by said server via said unicast	<pre> 00001603 sendClientPacket(packet) 00001604 /* Pick a random type for the packet */ </pre>

<p>network, said aggregated message to a recipient host computer belonging to said first message group.</p>	<pre> 00001605 struct player_spacket *packet; 00001606 { [...] 00001639 if (bufptr-buf+size >= BUFSIZE) { 00001640 t=bufptr-buf; 00001641 if (gwrite(sock, buf, t) != t) { 00001642 perror("std gwrite failed, client marked dead "); 00001643 clientDead=1; 00001644 } 00001645 bufptr=buf; 00001646 } 00001647 bcopy(packet, bufptr, size); 00001648 bufptr+=size; 00001649 00001650 } else { [...] 00001731 if (bufptr-buf+size >= BUFSIZE) { 00001732 t=bufptr-buf; 00001733 if (gwrite(sock, buf, t) != t) { 00001734 perror("TCP gwrite failed, client marked dead"); 00001735 clientDead=1; 00001736 } 00001737 bufptr=buf /*+ addSequence(buf)*/; 00001738 } 00001739 bcopy(packet, bufptr, size); 00001740 bufptr+=size; 00001741 break; 00001742 } 00001743 } 00001744 } Server\ntserv\socket.c at lines 1603-744 00000603 updateClient() 00000604 { </pre>
---	---

```

[...]
```

```

00000688     flushSockBuf();
00000689     repCount++;
00000690 }
Server\ntserv\socket.c at lines 603-90

00001747 flushSockBuf()
00001748 {
[...]
```

```

00001755     if (gwrite(sock, buf, t) != t) {
00001756         perror("std flush gwrite failed, client marked dead");
00001757         clientDead=1;
00001758     }
[...]
```

```

00001782     if (gwrite(udpSock, udpbuf, t) != t){
00001783         perror("UDP flush gwrite failed, client marked dead once
more");
[...]
```

```

00001791 }
[...]
```

```

00001802 }
Server\ntserv\socket.c at lines 1747-802

00002607 gwrite(fd, wbuf, size)
00002608 int fd;
00002609 char *wbuf;
00002610 size_t size;
00002611 {
[...]
```

```

00002625     while (bytes>0) {
00002626     n = write(fd, wbuf, bytes);
00002627     if (count++ > 100) {
00002628         ERROR(1,("Gwrite hosed: too many writes
(%d)\n",getpid()));

```

	<pre> 00002629 clientDead = 1; 00002630 return (-1); 00002631 } [...] 00002671 } 00002672 return(orig); 00002673 } Server\ntserv\socket.c at lines 2607-73 00001125 updateTorps() 00001126 { [...] 00001132 for (i=0, torp=torps, tpi=clientTorpsInfo, tp=clientTorps; 00001133 i<MAXPLAYER*MAXTORP; 00001134 i++, torp++, tpi++, tp++) { [...] 00001142 sendClientPacket (tpi); [...] 00001151 sendClientPacket (tp); [...] 00001191 } 00001192 } Server\ntserv\socket.c at lines 1125-92 </pre>
<p>2. The method of claim 1 wherein said time interval is a fixed period of time.</p>	<pre> 00000152 input () 00000153 { 00000154 struct itimerval udt; 00000155 fd_set readfds; 00000156 static struct timeval poll = {2, 0}; 00000157 00000158 #ifdef DS 00000159 if (!me->p_process) 00000160 #endif 00000161 { </pre>

	<pre> 00000162 udt.it_interval.tv_sec = 0; 00000163 udt.it_interval.tv_usec = timerDelay; 00000164 udt.it_value.tv_sec = 0; 00000165 udt.it_value.tv_usec = timerDelay; 00000166 setitimer(ITIMER_REAL, &udt, 0); 00000167 } 00000168 SIGNAL(SIGALRM, setflag); Server\ntserv\input.c at lines 152-168 00000076 int timerDelay=200000; /* delay between sending stuff to client */ Server\ntserv\data.c at line 76 </pre>
<p>3. The method of claim 1 wherein said time interval corresponds to a time for said server to receive at least one message from each host computer belonging to said first message group.</p>	<pre> 00000195 readFromClient(); Server\ntserv\input.c at line 195 00000152 input() 00000153 { 00000154 struct itimerval udt; 00000155 fd_set readfds; 00000156 static struct timeval poll = {2, 0}; 00000157 00000158 #ifdef DS 00000159 if (!me->p_process) 00000160 #endif 00000161 { 00000162 udt.it_interval.tv_sec = 0; 00000163 udt.it_interval.tv_usec = timerDelay; 00000164 udt.it_value.tv_sec = 0; 00000165 udt.it_value.tv_usec = timerDelay; 00000166 setitimer(ITIMER_REAL, &udt, 0); 00000167 } 00000168 SIGNAL(SIGALRM, setflag); 00000169 </pre>


```

00000170      /* Idea: read from client often, send to client
not so often */
00000171      while (1) {
[...]
00000195          readFromClient();
[...]
00000203      }
00000204  }
Server\ntserv\input.c at lines 152-203

00000076 int  timerDelay=200000; /* delay between sending stuff to
client */
Server\ntserv\data.c at line 76

00000603 updateClient()
00000604 {
[...]
00000608     static int skip = 0; /* If skip is set we skip next update
*/
00000609 /* This can halve your updates */
00000610     if (send_short && skip) {
00000611         skip = 0; /* back to default */
00000612     if (bufptr==buf && (commMode!=COMM_UDP || udpbufptr==buf))
{
00000613         /* We sent nothing! We better send something to wake
him */
00000614         if (me->p_fuel < 61000)
00000615             sendClientPacket((CVOID) &clientSelfShort);
00000616         else
00000617             sendClientPacket((CVOID) &clientSelf);
00000618     }
00000619     flushSockBuf();
00000620     repCount++;
00000621     return;

```

```

00000622     }
[...
00000630         if(send_short){
00000631     updatePlasmas();
00000632     updateStatus();
00000633     updateSelf();
00000634     updatePhasers();
00000635     updateShips();
00000636     updateTorps();
00000637     updatePlanets();
00000638     updateMessages();
00000639     }
[...
00000657         if(send_short && (me->p_fuel < 61000))
00000658             sendClientPacket((CVOID) &clientSelfShort);
00000659     else
00000660 #endif
00000661     sendClientPacket((CVOID) &clientSelf);
00000662     }
[...
00000685         sendClientPing();           /* ping.c */
00000686 #endif
00000687
00000688     flushSockBuf();
00000689     repCount++;
00000690 }

```

Server\ntserv\socket.c at lines 603-90

```

00000052     intrupt();

```

Server\ntserv\input.c at lines 52

```

00000197     intrupt();

```

Server\ntserv\input.c at lines 197

```
00001390 updateMessages ()
00001391 {
[...]
```

Server\ntserv\socket.c at lines 1390-590

```
00001390 updateMessages ()
00001391 {
[...]
```

```
00001563     if (cur->m_from==DOOSHMSG) msg.m_from=255; /* god */
00001564     if ((cur->m_from < 0) || (cur->m_from > MAXPLAYER))
00001565         sendClientPacket((CVOID) &msg);
00001566     else if (cur->m_flags & MALL && !(ignored[cur->m_from]
& MALL))
00001567         sendClientPacket((CVOID) &msg);
00001568     else if (cur->m_flags & MTEAM && !(ignored[cur->m_from]
& MTEAM)){
00001569         sendClientPacket((CVOID) &msg);
00001570     }
```

```
[...]
00001590 }
```

Server\ntserv\socket.c at lines 1390-590

```
00001825 readFromClient ()
00001826 {
[...]
```

```
00001838     if (select(32,&readfds,0,0,&timeout) != 0) {
00001839     /* Read info from the xtrek client */
00001840     if (FD_ISSET(sock, &readfds)) {
00001841         retval += doRead(sock);
00001842     }
00001843     if (udpSock >= 0 && FD_ISSET(udpSock, &readfds)) {
00001844         V_UDPDIAG(("Activity on UDP socket\n"));
00001845         retval += doRead(udpSock);
```

```

00001846 }
00001847 }
00001848     return (retval != 0);           /* convert to 1/0 */
00001849 }
00001850
[...]
```

```

00001855 /* ripped out of above routine */
00001856 doRead(asock)
00001857 int asock;
00001858 {
00001859     struct timeval timeout;
[...]
```

```

00001877         /* Read info from the xtrem server */
00001878         count=read(asock,buf,BUFSIZ*2);
[...]
```

```

00001916         bufptr=buf;
00001917         while (bufptr < buf+count) {
[...]
```

```

00001939         while (size>count+(buf-bufptr)) {
00001940             /* We wait for up to twenty seconds for rest of packet.
00001941              * If we don't get it, we assume the client
00001942              * died.
00001943              */
00001943             timeout.tv_sec=20;
00001944             timeout.tv_usec=0;
00001945             /*readfds=1<<asock;*/
00001946             FD_ZERO(&readfds);
00001947             FD_SET(asock, &readfds);
[...]
```

```

00001956             temp=read(asock,buf+count,size-(count+(buf-bufptr)));
[...]
```

```

00001966         }
[...]
```

```

00002010             (*(handlers[*bufptr].handler))(bufptr);

```

```

00002011     }
00002012     /* Otherwise we ignore the request */
00002013     } else {
00002014         ERROR(1,("Handler for packet %d not installed...\n",
*bufptr));
00002015     }
00002016         bufptr+=size;
00002017         if (bufptr>buf+BUFSIZ) {
00002018             bcopy(buf+BUFSIZ, buf, BUFSIZ);
00002019             if (count==BUFSIZ*2) {
00002020                 /*readfds = 1<<asock;*/
00002021                 FD_ZERO(&readfds);
00002022                 FD_SET(asock, &readfds);
00002023                 if (select(32,&readfds,0,0,&timeout)) {
00002024                     temp=read(asock,buf+BUFSIZ,BUFSIZ);
00002025                     count=BUFSIZ+temp;
[...]
```

Server\ntserv\socket.c at lines 1825-2044

```

00000021 intrupt()
00000022 {
[...]
```

```

00000114     updateClient();
```

	<pre> 00000115 } Server\ntserv\redraw.c at lines 21-115 </pre>
<p>4. The method of claim 1 further comprising the step of creating, by one of said plurality of host computers, said first message group by sending a first control message to said server via said unicast network.</p>	<pre> 00000057 entrywindow(team, s_type) 00000058 int *team, *s_type; [...] 00000074 /* The following allows quick choosing of teams */ 00000075 00000076 if(fastQuit){ 00000077 *team = -1; 00000078 return; 00000079 } 00000080 [...] 00000182 switch ((int) event.type) { [...] 00000260 case W_EV_BUTTON: 00000261 if (typeok == 0) 00000262 break; 00000263 for (i = 0; i < 4; i++) 00000264 if (event.Window == teamWin[i]) { 00000265 *team = i; 00000266 break; 00000267 } 00000268 if (event.Window == qwin /* new */ && 00000269 event.type == W_EV_BUTTON) { 00000270 *team = 4; 00000271 break; 00000272 } 00000273 if (*team != -1 && !teamRequest(*team, *s_type)) { 00000274 *team = -1; 00000275 } 00000276 break; 00000319 /* Attempt to pick specified team & ship */ </pre>

```

00000320 teamRequest(team, ship)
00000321     int             team, ship;
00000322 {
00000323     int             lastTime;
00000324
00000325     pickOk = -1;
00000326     sendTeamReq(team, ship);
00000327     lastTime = time(NULL);
00000328     while (pickOk == -1) {
00000329         if (lastTime + 3 < time(NULL)) {
00000330             sendTeamReq(team, ship);
00000331         }
00000332         socketPause();
00000333         readFromServer(NULL);
00000334         if (isServerDead()) {
00000335             [...]
00000340         if (udpSock >= 0)
00000341             closeUdpConn();
00000342         if (udpWin) {
00000343             udprefresh(UDP_CURRENT);
00000344             udprefresh(UDP_STATUS);
00000345         }
00000346         connectToServer(nextSocket);
00000347         printf("Yea! We've been resurrected!\n");
00000348         pickOk = 0;
00000349         break;
00000350     }
00000351 }
00000352     return (pickOk);
00000353 }
brmh-1.7/entrywin.c at lines 57-353

00001800 sendTeamReq(team, ship)
00001801     int             team, ship;

```

	<pre> 00001802 { 00001803 struct outfit_cpacket outfitReq; 00001804 00001805 outfitReq.type = CP_OUTFIT; 00001806 outfitReq.team = team; 00001807 outfitReq.ship = ship; 00001808 sendServerPacket((struct player_spacket *) & outfitReq); 00001809 } </pre> <p>brmh-1.7entrywin.c at lines 1800-09</p>
<p>5. The method of claim 4 further comprising the step of joining, by some of said plurality of host computers, said first message group by sending control messages via said unicast network to said server specifying said first message group.</p>	<pre> 00000057 entrywindow(team, s_type) 00000058 int *team, *s_type; [...] 00000074 /* The following allows quick choosing of teams */ 00000075 00000076 if(fastQuit){ 00000077 *team = -1; 00000078 return; 00000079 } 00000080 [...] 00000182 switch ((int) event.type) { [...] 00000260 case W_EV_BUTTON: 00000261 if (typeok == 0) 00000262 break; 00000263 for (i = 0; i < 4; i++) 00000264 if (event.Window == teamWin[i]) { 00000265 *team = i; 00000266 break; 00000267 } 00000268 if (event.Window == qwin /* new */ && 00000269 event.type == W_EV_BUTTON) { 00000270 *team = 4; </pre>


```

00000271     break;
00000272     }
00000273     if (*team != -1 && !teamRequest(*team, *s_type)) {
00000274         *team = -1;
00000275     }
00000276     break;
00000319 /* Attempt to pick specified team & ship */
00000320 teamRequest(team, ship)
00000321     int         team, ship;
00000322 {
00000323     int         lastTime;
00000324
00000325     pickOk = -1;
00000326     sendTeamReq(team, ship);
00000327     lastTime = time(NULL);
00000328     while (pickOk == -1) {
00000329         if (lastTime + 3 < time(NULL)) {
00000330             sendTeamReq(team, ship);
00000331         }
00000332         socketPause();
00000333         readFromServer(NULL);
00000334         if (isServerDead()) {
00000335             [...]
00000340             if (udpSock >= 0)
00000341                 closeUdpConn();
00000342             if (udpWin) {
00000343                 udprefresh(UDP_CURRENT);
00000344                 udprefresh(UDP_STATUS);
00000345             }
00000346             connectToServer(nextSocket);
00000347             printf("Yea! We've been resurrected!\n");
00000348             pickOk = 0;
00000349             break;
00000350         }

```

	<pre> 00000351 } 00000352 return (pickOk); 00000353 } brmh-1.7/entrywin.c at lines 57-353 00001800 sendTeamReq(team, ship) 00001801 int team, ship; 00001802 { 00001803 struct outfit_cpacket outfitReq; 00001804 00001805 outfitReq.type = CP_OUTFIT; 00001806 outfitReq.team = team; 00001807 outfitReq.ship = ship; 00001808 sendServerPacket((struct player_spacket *) & outfitReq); 00001809 } brmh-1.7/socket.c at lines 1800-09 </pre>
<p>6. The method of claim 1 wherein said network is Internet and</p>	<pre> 00000179 int connectionAttemptDetected(num_progs) 00000180 int num_progs; 00000181 { [...]</pre> <pre> 00000192 /* check all ports */ 00000193 for (i = 0; i < num_progs; i++) { 00000194 sock = prog[i].sock; 00000195 if(sock < 0){ 00000196 if((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0){ 00000197 fprintf(stderr, "Hey! I can't make a socket!\n"); 00000198 fprintf(stderr, "I'll try again \n"); 00000199 return(-1); 00000200 } [...]</pre> <pre> 00000214 if(bind(sock, &addr, sizeof(addr)) < 0){ [...]</pre> <pre> 00000227 if (bind(sock, &addr, sizeof(addr)) < 0) { </pre>

```

[...]
```

00000237	}
00000238	}
00000239	}
00000240	if(listen(sock, 1)<0) {
00000241	fprintf(stderr,"Listen failed: ");
00000242	reporterror();
00000243	sock = -1;
00000244	}
00000245	prog[i].sock=sock;
00000246	/* close(0); */
00000247	fprintf (stderr,"listening on %d, connection will
00000248	start %s \"%s\" %s %s %s %s\n",
00000249	prog[i].port, prog[i].prog, prog[i].progname,
00000250	prog[i].arg[0], prog[i].arg[1], prog[i].arg[2],
00000251	prog[i].arg[3]);
00000252	fflush (stderr);
00000253	}
00000254	}
00000255	while (1) /* Wait for a connection */
00000256	{
00000257	[...]
00000271	for (i = 0; i < num_progs; i++)
00000272	{
00000273	sock = prog[i].sock;
00000274	if (sock < 0)
00000275	continue;
00000276	if (FD_ISSET (sock, &accept_fds))
00000277	/* found a connection, procede */
00000278	goto found;
00000279	}
00000280	}
00000281	}

	<pre> [...] 00000311 } Server\newstartd\newstartd.c at lines 179-311 00000442 connectToClient(machine, port) 00000443 char *machine; 00000444 int port; 00000445 { [...] 00000456 ERROR(3,("Connecting to %s through %d\n", machine, port)); [...] 00000478 00000479 if (connect(ns, &addr, sizeof(addr)) < 0) { 00000480 ERROR(3,("I cannot connect through port %d\n", port)); 00000481 close(ns); 00000482 return(0); 00000483 } 00000484 sock=ns; 00000485 initClientData(); 00000486 testtime = -1; 00000487 return(1); 00000488 } Server\ntserv\socket.c at lines 442-88 </pre>
<p>said server communicates with said plurality of host computers using a session layer protocol.</p>	<pre> 00000179 int connectionAttemptDetected(num_progs) 00000180 int num_progs; 00000181 { [...] 00000192 /* check all ports */ 00000193 for (i = 0; i < num_progs; i++) { 00000194 sock = prog[i].sock; 00000195 if(sock < 0){ 00000196 if((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0){ 00000197 fprintf(stderr,"Hey! I can't make a socket!\n"); </pre>

```

00000198     fprintf(stderr,"I'll try again \n");
00000199     return(-1);
00000200 }
[...]
00000214 if(bind(sock, &addr, sizeof(addr)) < 0){
[...]
00000227     if (bind(sock, &addr, sizeof(addr)) < 0) {
[...]
00000237     }
00000238 }
00000239 }
00000240 if(listen(sock, 1)<0) {
00000241     fprintf(stderr,"Listen failed: ");
00000242     reporterror();
00000243     sock = -1;
00000244 }
00000245 prog[i].sock=sock;
00000246 /* close(0); */
00000247     fprintf (stderr,"listening on %d, connection will
start %s \"%s\" %s %s %s %s\n",
00000248             prog[i].port, prog[i].prog, prog[i].progname,
00000249             prog[i].arg[0], prog[i].arg[1], prog[i].arg[2],
prog[i].arg[3]);
00000250     fflush (stderr);
00000251 }
00000252 }
00000253
00000254 while (1)                                /* Wait for a connection */
00000255     {
[...]
00000271         for (i = 0; i < num_progs; i++)
00000272             {
00000273                 sock = prog[i].sock;
00000274                 if (sock < 0)

```

```
00000275         continue;
00000276         if (FD_ISSET (sock, &accept_fds))
00000277             /* found a connection, procede */
00000278     goto found;
00000279     }
00000280     }
00000281     }
[...]
```

Server\newstartd\newstartd.c at lines 179-31

```
00000493 struct mesg_cpacket {
00000494     char type;          /* CP_MESSAGE */
00000495     u_char group;
00000496     char indiv;
00000497     char pad1;
00000498     char mesg[MSG_LEN];
00000499 };
```

Server\packets.h at lines 493-99

CC-C

Claim Chart comparing Claims 1, 2, and 4-6 of U.S. Patent No. 5,822,523 to the disclosure in Van Hook in view DIS

Prior art cited in this chart:

- Daniel J. Van Hook, James O. Calvin, Michael K. Newton, and David A. Fusco, “An Approach to DIS Scalability,” 11th DIS Workshop, 26-30 Sept. 1994 (“Van Hook”).
- IEEE 1278-1993 IEEE Standard for Information Technology- Protocols for Distributed Interactive Simulation Applications, approved March 18, 1993, and published in 1993 (“DIS”).
- Michael R. Macedonia, “Exploiting Reality with Multicast Groups”, published September 1995 (“Macedonia”)

Reasons to Combine:

Van Hook discloses techniques that have been developed and deployed for ARPA’s Synthetic Theater of War (STOW) program and Distributed Interactive Simulation (DIS), wherein a virtual world simulates battlefield conditions, and “[e]xplicit representations of command, control, and communication are required to permit command forces to transmit orders to and receive reports from a new generation of more intelligent semi-automated forces. Van Hook at p. 1. Likewise, DIS is part of a proposed set of standards for the Distributed Interactive Simulation (DIS) used in conjunction with the STOW program in Van Hook. DIS at pp. 1-3. Van Hook provides for bundling of the PDUs from host computers by the AG server into larger transmission packets to be distributed to other packets. *Id.* at pp. 2 and 7. DIS goes one step further to discuss the anatomy of a packet, as the PDU packets disclosed in DIS include a PDU header, an ID denoting a host computer (Entity ID) as well as a message group (Force ID), and the PDU message relating to positional information of the entity. DIS at Table 18. Pp. 40-41. It would have been obvious to a person of ordinary skill in that art to combine the teachings of bundling packets, or PDUs, in a Distributed Interactive Simulation disclosed in Van Hook with the teachings of the contents of a PDU in a Distributed Interactive Simulation as disclosed in DIS.

Claims of the '523 Patent	Disclosure of Van Hook, DIS and Macedonia
<p>1. A method for providing group messages to a plurality of host computers connected over a unicast wide area communication network, comprising the steps of:</p>	<p>“With the advent of ARPA’s Synthetic Theater of War (STOW) program and the continued development of Distributed Interactive Simulation (DIS), the scope of the problem has changed substantially. Under STOW and related programs, the virtual world must expand substantially — in area, total population, and the types of entities represented.” Van Hook at p. 1.</p> <p>“For these reasons, it has become necessary to reexamine the approach used to distribute information across the network. Several innovative algorithms have been suggested for reducing the amount of traffic transmitted, transported, received, and processed. Under ARPA’s Scaleability project, several Bandwidth Reduction Techniques (BRTs) are being explored, evaluated analytically and in simulation, and implemented in an Application Gateway (AG) residing at the LAN/WAN interface for each participating site in this Fall’s STOW-Europe (STOW-E) demonstration.” Van Hook at p. 1.</p> <p>“Major characteristics of STOW-E are:</p> <ul style="list-style-type: none"> • Live, virtual and constructive simulations • Wide area connectivity provided by the Defense Simulation Internet (DSI) Wide Area Network (WAN) • 24 network sites in the continental US and Europe: 18 directly on the DSI, the remaining bridged • Approximately 2,000 interacting DIS entities • Two security levels: Secret No-Foreign and US1 • DIS 2.0.3 protocols <p>Numerous legacy systems and simulators” Van Hook at p. 1.</p>

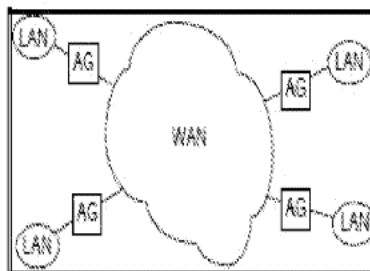


Figure 1: Application Gateway connections within the network

Figure 1 of Van Hook at p. 4.

“On the WAN side, the AG transmits / receives to / from a subnet broadcast address on a well known UDP port. The WAN essentially bridges traffic between sites that are members of a multicast group. ... On the LAN side, the AG runs in promiscuous mode, receiving all packets transmitted on the site LAN.”

Van Hook at pp. 4-5.

“**distributed interactive simulation (DIS).** A time and space coherent synthetic representation of world environments designed for linking the interactive, free-play activities of people in operational exercises. The synthetic environment is created through the real-time exchange of data units between distributed, computationally autonomous simulation applications in the form of simulations, simulators, and instrumented equipment interconnected through standard computer services. The computational simulation entities may be present in one location or may be distributed geographically.”

DIS at p. 3.

“**3.17 host computer:** A computer that supports one or more simulation applications. All host computers participating in a simulation exercise are connected by a common network.”

DIS at p. 4.

“Data messages, called protocol data units (PDUs) that are exchanged between simulation applications are defined. These PDUs provide information concerning simulated entity states and the types of entity interactions that take place in a distributed interactive simulation (DIS).”

	<p>DIS at Abstract.</p> <p>“3.33 unicast: A transmission mode in which a single message is sent to a single network destination, that is, one-to-one.”</p> <p>DIS at p. 5.</p> <p>“The communication services require by each DIS PDU are described in detail in IST-CR-92-6. A brief summary of the basic communication services necessary for DIS is as follows:</p> <ul style="list-style-type: none"> a) <i>Data transfer.</i> Each simulation application must be able to transfer data to another simulation application on the network in a single operation, with or without first establishing a logical connection with the destination computer. The unit of data passed in a single operation is called a packet. b) <i>Delivery.</i> The communication architecture must support either, multicast, broadcast, or unicast packets. Multicast packets are delivered to a subset of all computers on a network. Broadcast packets are delivered to all computers on a network. (Broadcasting is actually a special kind of multicast.) Unicast packets are delivered to a single computer on a network. c) <i>Best effort service.</i> The communication architecture should support best effort delivery. Although DIS simulation applications will tolerate occasional failures of the network to deliver packets, these should be allowed to occur only rarely. d) <i>Packet integrity.</i> The communication protocols should be capable of detecting transmission errors associated with the network. Corrupted packets should not be delivered to the simulation application. e) <i>Performance requirements.</i> The communication architecture should provide a certain level of performance characterized in terms of throughput and delay. Both network delay and network delay variance are to be minimized.” <p>DIS at p. 10.</p>
<p>providing a group messaging server coupled to said network, said server communicating with said plurality of host computers</p>	<p>“Exercise scale. The large number of entities involved in STOW-E will produce offered loads of as much as four megabits and perhaps up to 2,000 packets per second. Such traffic levels will severely tax all simulation computers even if unlimited communications resources were available.”</p> <p>Van Hook at p. 1.</p>

using said unicast network and maintaining a list of message groups, each message group containing at least one host computer;

“Explicit representations of command, control, and communication are required to permit command forces to transmit orders to and receive reports from a new generation of more intelligent semi-automated forces. These new elements and phenomena require new protocols and generate new classes of traffic that must be carried on the connecting networks.”

Van Hook at p. 1.

“A component of ARPA’s approach to scalability for STOW-E is to implement cooperating and complementary instances of a number of the information flow management techniques in an Application Gateway (AG) situated at the LAN/WAN boundary of each participating network site (figure 1). The AG may be thought of as a collection of information flow management agents [4] that perform services on behalf of their clients, the simulation applications. The purpose of these agents is to compensate for and efficiently use the available communication and processing resources. Each AG processes PDUs received from its attached LAN and sends representation of local exercise state and events to other AGs over the WAN. Similarly, each AG receives representations of remote state and events from other AGs over the WAN and sends PDUs onto its attached LAN. Communication between AGs is via an Application Gateway to Gateway Protocol (AGGP). AGGP supports communication of control information related to the information flow management techniques as well as representations of exercise state and events.”

Van Hook at p. 4.

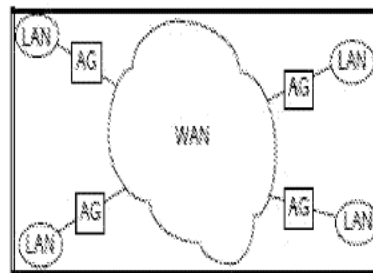


Figure 1: Application Gateway connections within the network

Figure 1 of Van Hook at p. 4.

“The algorithm operates as follows. The terrain is divided into a grid of square cells by each AG. A

square grid is used because it makes calculations simple and permits regions of the terrain to be specified as a list of cells. Each AG determines the set of cells from which it needs to receive full accuracy data. This set consists of those cells that overlay the circular regions of interest of the entities at the AG's site LAN. Figure 5 illustrates this idea by showing three entities and their circular regions of interest. For determining the full accuracy region, the AGs use regions of interest that are based upon the viewing ranges of the entities on the site LAN. The set of cells for which full accuracy data is needed is outlined in the figure. All AGs transmit their cell sets to each other. The full accuracy region for any AG consists of the union of the sets of cells received from all other AGs."

Van Hook at p. 6.

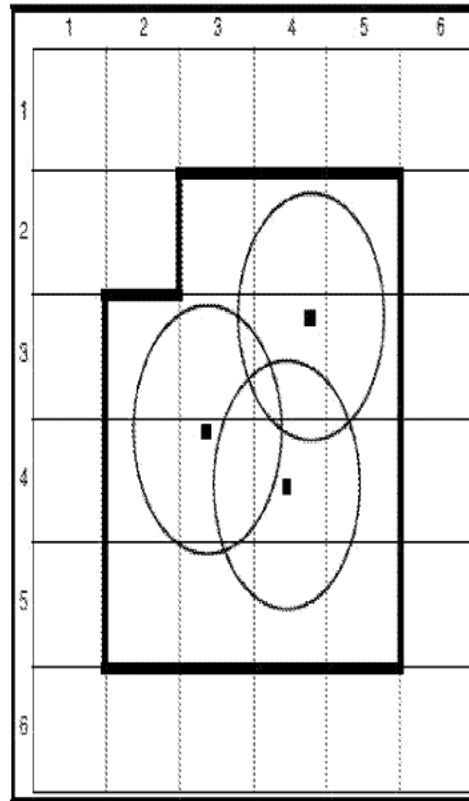


Figure 5: Cells for which full accuracy is required

Figure 5 of Van Hook at p. 6.

	<p>“6.2.14 Force ID This field shall distinguish the different teams or sides in a DIS exercise.” DIS at p. 36.</p>
<p>sending, by a plurality of host computers belonging to a first message group, messages to said server via said unicast network, said messages containing a payload portion and a portion for identifying said first message group;</p>	<p>“The DIS protocols support the exchange of information about the state of the entities participating in an exercise and events related to their activities and interactions.” Van Hook at p. 2.</p> <p>“A grid-based relevance filtering algorithm is incorporated into the AG. It operates on Entity State PDUs originating on an AG’s site LAN as well as those arriving from remote AGs via the WAN, as shown in figure 4.” Van Hook at p. 5.</p> <p>“The algorithm operates as follows. The terrain is divided into a grid of square cells by each AG. A square grid is used because it makes calculations simple and permits regions of the terrain to be specified as a list of cells. Each AG determines the set of cells from which it needs to receive full accuracy data. This set consists of those cells that overlay the circular regions of interest of the entities at the AG’s site LAN. Figure 5 illustrates this idea by showing three entities and their circular regions of interest. For determining the full accuracy region, the AGs use regions of interest that are based upon the viewing ranges of the entities on the site LAN. The set of cells for which full accuracy data is needed is outlined in the figure. All AGs transmit their cell sets to each other. The full accuracy region for any AG consists of the union of the sets of cells received from all other AGs.” Van Hook at p. 6.</p> <p>“The Entity State PDU shall communicate information about an entity’s state. This include state information that is necessary for the receiving simulation applications to represent the issuing entity in the simulation applications’ own simulation.” DIS at p. 14.</p> <p>“A PDU header record shall be the first part of each PDU. This record is represented in table 14. The fields of the PDU header record are described in the following four items (see also 5.5.1).</p>

- a) *Protocol version*. This field shall specify the version of protocol used in this PDU. Protocol data units found in this standard shall be specified as version 2. This field shall be specified by an 8-bit enumeration.
- b) *Exercise identification*. This field shall specify the exercise to which the PDU pertains. The value contained in this field shall not be equal to zero. This field shall be represented by an Exercise Identifier (see 6.2.13).
- c) *Protocol data unit type*. This field shall indicate the type of PDU that follows. This field shall be represented by an 8-bit enumeration. The values in this field are defined in Section 4 in IST-CR-92-16.
- d) *Length*. This field shall specify the length of the PDU in 32-bit words. This field shall be represented by an 8-bit unsigned integer.”

DIS at p. 36.

“Information about a particular entity shall be communicated by issuing an Entity State PDU. The Entity State PDU shall contain the following fields:

- a) *PDU header*. This field shall contain data common to all DIS PDUs. The PDU header shall be represented by the PDU Header Record (see 6.2.15).
- b) *Entity Identification*. This field shall identify the entity issuing the PDU. This field shall be represented by an Entity Identifier Record (see 6.2.8).
- c) *Force identification*. This field shall identify the force to which the issuing entity belongs. This field shall be represented by an 8-bit enumeration (see Section 4 in IST-CR-92-16).
- d) *Entity type*. This field shall identify the entity type to be displayed by members of the same force as the issuing entity. This field shall be represented by an Entity Type Record (see 6.2.10 and Section 6 in IST-CR-92-16).
- e) *Alternate entity type*. This field shall identify the entity type to be displayed by members of forces other than that of the issuing entity. This field shall be represented by an Entity Type Record (see 6.2.10 and Section 4 in IST-CR-92-16).
- f) *Timestamp*. This field shall specify that time at which the data in the PDU is valid. This field shall be represented by a timestamp (see 6.2.19).
- g) *Entity location*. This field shall specify an entity’s physical location in the simulated world. This field shall be represented by a World Coordinates Record (see 6.2.21).

- h) *Entity linear velocity*. This field shall specify an entity's linear velocity. This field shall be represented by a Linear Velocity Vector Record (see 6.2.20.3).
- i) *Entity orientation*. This field shall specify an entity's orientation. This field shall be represented by a Euler Angles Record (see 6.2.11.)..."

DIS at p. 39.

Table 18--Entity State PDU

Field Size (bits)	Entity State PDU Fields	
32	PDU header	Protocol Version--8-bit enumeration
		Exercise ID--4-bit unsigned integer
		PDU Type--8-bit enumeration
		Length--8-bit unsigned integer
48	Entity ID	Site--16-bit unsigned integer
		Host--18-bit unsigned integer
		Entity--16-bit unsigned integer
8	Padding	8 bits unused
8	Force ID	8-bit enumeration
64	Entity type	Entity Kind--8-bit enumeration
		Domain--8-bit enumeration
		Country--16-bit enumeration
		Category--8-bit enumeration
		Subcategory--8-bit enumeration
		Specific--8-bit enumeration
		Extra--8-bit enumeration

Field Size (bits)	Entity State PDU Fields	
64	Alternative entity type	Entity Kind -- 8-bit enumeration
		Domain -- 8-bit enumeration
		Country -- 16-bit enumeration
		Category -- 8-bit enumeration
		Subcategory -- 8-bit enumeration
		Specific -- 8-bit enumeration
72	Timestamp	32-bit unsigned integer
192	Entity location	X Component -- 64-bit floating point
		Y Component -- 64-bit floating point
		Z Component -- 64-bit floating point
96	Entity linear velocity	X Component -- 32-bit floating point
		Y Component -- 32-bit floating point
		Z Component -- 32-bit floating point
96	Entity orientation	q -- 32-bit floating point
		h -- 32-bit floating point
		o -- 32-bit floating point
304	Dead reckoning parameters	Dead reckoning algorithm -- 8-bit enumeration
		Other parameters -- 120 bits reserved
		Entity Linear Accel -- 3 x 32-bit floating points
		Entity Angular Velocity -- 3 x 32-bit integers
32	Entity appearance	32-bit vector of enumerations
96	Entity marking	Character set -- 8-bit enumeration
		N -- 8-bit unsigned integer
32	Capabilities	32 Boolean fields
24	Flags	Unsigned
8	# of articulation parameters	8-bit unsigned integer
n x 128	Articulation parameters	Change -- 16-bit unsigned integer
		Qb -- attached as 36-bit unsigned integer
		Parameter type -- 32-bit Parameter Type Reserved
		Parameter value -- 64 bits
Total Entity State PDU size = 1152 + 128n bits where n = number of articulation parameters		

For each articulation parameter

Table 18 of DIS at pp. 40-41.

“3.33 unicast: A transmission mode in which a single message is sent to a single network destination,

	<p>that is, one-to-one.” DIS at p. 5.</p>
<p>aggregating, by said server in a time interval determined in accordance with a predefined criterion, said payload portions of said messages to create an aggregated payload;</p>	<p>“Bundling. Network components such as switches, routers, and encryption devices as well as simulation host computers have limitations in the rate at which they may process packets. Rather than transmitting each DIS PDU as an individual packet, multiple PDUs may be bundled together into larger packets before transmission. Bundled packets are transmitted when either of two conditions are satisfied: when a maximum size has been reached (the packet under construction is full of PDUs); or when a maximum time has passed without another PDU arriving. The dominant effect of bundling is to reduce packet rates. Additionally, bundling reduces bit rates because fewer packet headers are sent.” Van Hook at p. 2.</p> <p>“4.6 Bundling</p> <p>The AG collects AGGP PDUs and bundles them into larger packets for transmission over the WAN. The purpose of the bundling algorithm is to reduce the number of packets that are transmitted. The bundling algorithm has two parameters, a maximum bundle size and a maximum delay time. PDUs are added to a bundle until either the maximum size is reached or the first PDU in the bundle has been delayed by the maximum delay time. At this point, the bundle is transmitted.” Van Hook at p. 7.</p>
<p>forming an aggregated message using said aggregated payload; and</p>	<p>“Bundling. Network components such as switches, routers, and encryption devices as well as simulation host computers have limitations in the rate at which they may process packets. Rather than transmitting each DIS PDU as an individual packet, multiple PDUs may be bundled together into larger packets before transmission. Bundled packets are transmitted when either of two conditions are satisfied: when a maximum size has been reached (the packet under construction is full of PDUs); or when a maximum time has passed without another PDU arriving. The dominant effect of bundling is to reduce packet rates. Additionally, bundling reduces bit rates because fewer packet headers are sent.” Van Hook at p. 2.</p> <p>“4.6 Bundling</p> <p>The AG collects AGGP PDUs and bundles them into larger packets for transmission over the WAN.</p>

	<p>The purpose of the bundling algorithm is to reduce the number of packets that are transmitted. The bundling algorithm has two parameters, a maximum bundle size and a maximum delay time. PDUs are added to a bundle until either the maximum size is reached or the first PDU in the bundle has been delayed by the maximum delay time. At this point, the bundle is transmitted.”</p> <p>Van Hook at p. 7.</p>
<p>transmitting, by said server via said unicast network, said aggregated message to a recipient host computer belonging to said first message group.</p>	<p>“Bundling. Network components such as switches, routers, and encryption devices as well as simulation host computers have limitations in the rate at which they may process packets. Rather than transmitting each DIS PDU as an individual packet, multiple PDUs may be bundled together into larger packets before transmission. Bundled packets are transmitted when either of two conditions are satisfied: when a maximum size has been reached (the packet under construction is full of PDUs); or when a maximum time has passed without another PDU arriving. The dominant effect of bundling is to reduce packet rates. Additionally, bundling reduces bit rates because fewer packet headers are sent.”</p> <p>Van Hook at p. 2.</p> <p>“4.6 Bundling</p> <p>The AG collects AGGP PDUs and bundles them into larger packets for transmission over the WAN. The purpose of the bundling algorithm is to reduce the number of packets that are transmitted. The bundling algorithm has two parameters, a maximum bundle size and a maximum delay time. PDUs are added to a bundle until either the maximum size is reached or the first PDU in the bundle has been delayed by the maximum delay time. At this point, the bundle is transmitted.”</p> <p>Van Hook at p. 7.</p> <p>“Exercise scale. The large number of entities involved in STOW-E will produce offered loads of as much as four megabits and perhaps up to 2,000 packets per second. Such traffic levels will severely tax all simulation computers even if unlimited communications resources were available.”</p> <p>Van Hook at p. 1.</p> <p>“Explicit representations of command, control, and communication are required to permit command forces to transmit orders to and receive reports from a new generation of more intelligent semi-automated forces. These new elements and phenomena require new protocols and generate new classes</p>

of traffic that must be carried on the connecting networks.”

Van Hook at p. 1.

“A component of ARPA’s approach to scalability for STOW-E is to implement cooperating and complementary instances of a number of the information flow management techniques in an Application Gateway (AG) situated at the LAN/WAN boundary of each participating network site (figure 1). The AG may be thought of as a collection of information flow management agents [4] that perform services on behalf of their clients, the simulation applications. The purpose of these agents is to compensate for and efficiently use the available communication and processing resources. Each AG processes PDUs received from its attached LAN and sends representation of local exercise state and events to other AGs over the WAN. Similarly, each AG receives representations of remote state and events from other AGs over the WAN and sends PDUs onto its attached LAN. Communication between AGs is via an Application Gateway to Gateway Protocol (AGGP). AGGP supports communication of control information related to the information flow management techniques as well as representations of exercise state and events.”

Van Hook at p. 4.

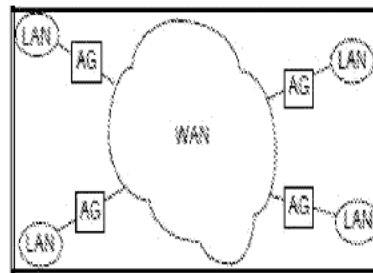


Figure 1: Application Gateway connections within the network

Figure 1 of Van Hook at p. 4.

“The algorithm operates as follows. The terrain is divided into a grid of square cells by each AG. A square grid is used because it makes calculations simple and permits regions of the terrain to be specified as a list of cells. Each AG determines the set of cells from which it needs to receive full accuracy data. This set consists of those cells that overlay the circular regions of interest of the entities

at the AG's site LAN. Figure 5 illustrates this idea by showing three entities and their circular regions of interest. For determining the full accuracy region, the AGs use regions of interest that are based upon the viewing ranges of the entities on the site LAN. The set of cells for which full accuracy data is needed is outlined in the figure. All AGs transmit their cell sets to each other. The full accuracy region for any AG consists of the union of the sets of cells received from all other AGs."

Van Hook at p. 6.

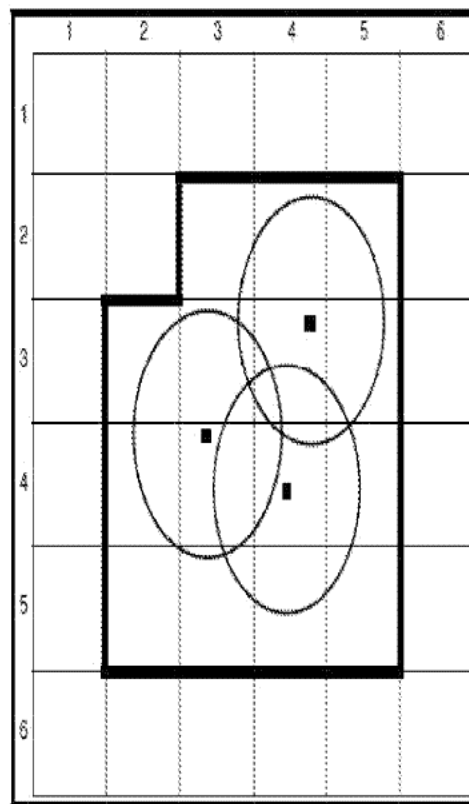


Figure 5: Cells for which full accuracy is required

Figure 5 of Van Hook at p. 6.

“3.33 unicast: A transmission mode in which a single message is sent to a single network destination, that is, one-to-one.”

<p>2. The method of claim 1 wherein said time interval is a fixed period of time.</p>	<p>DIS at p. 5.</p> <p>“Bundling. Network components such as switches, routers, and encryption devices as well as simulation host computers have limitations in the rate at which they may process packets. Rather than transmitting each DIS PDU as an individual packet, multiple PDUs may be bundled together into larger packets before transmission. Bundled packets are transmitted when either of two conditions are satisfied: when a maximum size has been reached (the packet under construction is full of PDUs); or when a maximum time has passed without another PDU arriving. The dominant effect of bundling is to reduce packet rates. Additionally, bundling reduces bit rates because fewer packet headers are sent.”</p> <p>Van Hook at p. 2.</p> <p>“4.6 Bundling</p> <p>The AG collects AGGP PDUs and bundles them into larger packets for transmission over the WAN. The purpose of the bundling algorithm is to reduce the number of packets that are transmitted. The bundling algorithm has two parameters, a maximum bundle size and a maximum delay time. PDUs are added to a bundle until either the maximum size is reached or the first PDU in the bundle has been delayed by the maximum delay time. At this point, the bundle is transmitted.”</p> <p>Van Hook at p. 7.</p>
<p>4. The method of claim 1 further comprising the step of creating, by one of said plurality of host computers, said first message group by sending a first control message to said server via said unicast network.</p>	<p>“The algorithm operates as follows. The terrain is divided into a grid of square cells by each AG. A square grid is used because it makes calculations simple and permits regions of the terrain to be specified as a list of cells. Each AG determines the set of cells from which it needs to receive full accuracy data. This set consists of those cells that overlay the circular regions of interest of the entities at the AG’s site LAN. Figure 5 illustrates this idea by showing three entities and their circular regions of interest. For determining the full accuracy region, the AGs use regions of interest that are based upon the viewing ranges of the entities on the site LAN. The set of cells for which full accuracy data is needed is outlined in the figure. All AGs transmit their cell sets to each other. The full accuracy region for any AG consists of the union of the sets of cells received from all other AGs.”</p> <p>Van Hook at p. 6.</p> <p>“6.2.14 Force ID</p> <p>This field shall distinguish the different teams or sides in a DIS exercise.”</p>

<p>5. The method of claim 4 further comprising the step of joining, by some of said plurality of host computers, said first message group by sending control messages via said unicast network to said server specifying said first message group.</p>	<p>DIS at p. 36.</p> <p>“The algorithm operates as follows. The terrain is divided into a grid of square cells by each AG. A square grid is used because it makes calculations simple and permits regions of the terrain to be specified as a list of cells. Each AG determines the set of cells from which it needs to receive full accuracy data. This set consists of those cells that overlay the circular regions of interest of the entities at the AG’s site LAN. Figure 5 illustrates this idea by showing three entities and their circular regions of interest. For determining the full accuracy region, the AGs use regions of interest that are based upon the viewing ranges of the entities on the site LAN. The set of cells for which full accuracy data is needed is outlined in the figure. All AGs transmit their cell sets to each other. The full accuracy region for any AG consists of the union of the sets of cells received from all other AGs.”</p> <p>Van Hook at p. 6.</p> <p>“6.2.14 Force ID This field shall distinguish the different teams or sides in a DIS exercise.”</p> <p>DIS at p. 36.</p> <p>“An entity joins a group as a passive or active member. Active members send as well as receive PDUs within the group, are located in the cell associated with the group (that is, the center of seven cells), and can become the group leader. Passive members normally do not send PDUs to the group except when they join or leave. They are associated with the group because the cell lies within their area of interest, yet they do not occupy the central cell.</p> <p>When an entity joins a new group, it notes the time it entered and issues a Join Request PDU to the cell group. The PDU has a flag indicating whether the cell is active or passive. The group leader replies with a Pointer PDU that references the request and in turn multicasts a PDU containing a pointer to itself or another active entity. The new member sends a Data Request PDU to the referenced source, which issues a Data PDU containing the aggregate set of active entity PDUs. A passive entity becomes an active member of a group by reissuing the Join Request PDU with a flag set to active when entering a cell. Departures from the group are announced with a Leave Request PDU.”</p> <p>Macedonia at p. 42.</p>
<p>6. The method of claim 1</p>	<p>“Major characteristics of STOW-E are:</p>

<p>wherein said network is Internet and said server communicates with said plurality of host computers using a session layer protocol.</p>	<ul style="list-style-type: none"> • Live, virtual and constructive simulations • Wide area connectivity provided by the Defense Simulation Internet (DSI) Wide Area Network (WAN) • 24 network sites in the continental US and Europe: 18 directly on the DSI, the remaining bridged • Approximately 2,000 interacting DIS entities • Two security levels: Secret No-Foreign and US1 • DIS 2.0.3 protocols <p>Numerous legacy systems and simulators” Van Hook at p. 1.</p>
--	--

CC-D

Claim Chart comparing Claims 1-2 and 4-6 of U.S. Patent No. 5,822,523 to the disclosure of IRC RFC in view of Friedman

Prior art cited in this chart:

- J. Oikarinen et al., RFC 1459- Internet Relay Chat Protocol, published May 1993
- R. Friedman et al., Packing Messages as a Tool for Boosting the Performance of Total Ordering Protocols, Dept. of Science of Cornell University, published July 7, 1995

Reason to Combine:

IRC RFC does not disclose aggregating payload portions, but Friedman discloses that messages are buffered and then the payloads are aggregated, *i.e.*, packed, before sending. Friedman at 5. In addition, RFC IRC states “The main goal of IRC is to provide a forum which allows easy and efficient conferencing (one to many conversations).” IRC RFC at § 3.2. Friedman discloses aggregation of message packets improves both latency and throughput compared to non-aggregating communication protocols. Friedman at 1. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to aggregate the group messages of IRC RFC, *i.e.*, channel messages, in order to increase the efficiency of the network which was a main goal of stated by IRC RFC.

Claims of the '523 Patent	Disclosure of IRC RFC and Friedman
<p>1. A method for providing group messages to a plurality of host computers connected over a unicast wide area communication network, comprising the steps of:</p>	<p>“IRC itself is a teleconferencing system, which (through the use of the client-server model) is well-suited to running on many machines in a distributed fashion. A typical setup involves a single process (the server) forming a central point for clients (or other servers) to connect to, performing the required message delivery/multiplexing and other functions.” IRC RFC at § 1</p> <p>“The IRC protocol was developed over the last 4 years since it was first implemented as a means for users on a BBS to chat amongst themselves. Now it supports a world-wide network of servers and clients, and is stringing [sic] to cope with growth.” IRC RFC at abstract</p> <p>“If there are multiple users on a server in the same channel, the message text is sent only once to that server and then sent to each client on the channel. This action is then repeated for each client-server combination until the original message has fanned out and reached each member of the channel.” IRC RFC at § 3.2.2.</p>
<p>providing a group messaging server coupled to said network, said server communicating with said plurality of host computers using said unicast network and maintaining a list of message groups, each message group containing at least one host computer;</p>	<p>“IRC itself is a teleconferencing system, which (through the use of the client-server model) is well-suited to running on many machines in a distributed fashion. A typical setup involves a single process (the server) forming a central point for clients (or other servers) to connect to, performing the required message delivery/multiplexing and other functions.” IRC RFC at § 1</p> <p>“A channel is a named group of one or more clients which will all receive messages addressed to that channel. The channel is created implicitly when the first client joins it and the channel ceases to exist when the last client leaves it. While channel exists, any client can reference the channel using the name of the channel.” IRC RFC at § 1.3</p>

	<p>“In IRC the channel has a role equivalent to that of the multicast group; their existence is dynamic (coming and going as people join and leave channels) and the actual conversation carried out on a channel is only sent to servers which are supporting users on a given channel. If there are multiple users on a server in the same channel, the message text is sent only once to that server and then sent to each client on the channel. This action is then repeated for each client-server combination until the original message has fanned out and reached each member of the channel.”</p> <p>IRC RFC at § 3.2.2</p> <p>“The current channel layout requires that all servers know about all channels, their inhabitants and properties.”</p> <p>IRC RFC at § 9.2.2</p>
<p>sending, by a plurality of host computers belonging to a first message group, messages to said server via said unicast network, said messages containing a payload portion and a portion for identifying said first message group;</p>	<p>“Command: PRIVMSG Parameters: <receiver>{,<receiver>} <text to be sent></p> <p>PRIVMSG is used to send private messages between users. <receiver> can also be a list of names or channels separated with commas.”</p> <p>IRC RFC at § 4.4.1</p>
<p>aggregating, by said server in a time interval determined in accordance with a predefined criterion, said payload portions of said messages to create an aggregated payload;</p>	<p>“The least efficient style of one-to-many conversation is through clients talking to a ‘list’ of users. How this is done is almost self explanatory: the client gives a list of destinations to which the message is to be delivered and the server breaks it up and dispatches a separate copy of the message to each given destination. This isn’t as efficient as using a group since the destination list is broken up and the dispatch sent without checking to make sure duplicates aren’t sent down each path.”</p> <p>IRC RFC at § 3.2.1</p> <p>“This protocol is essentially the same as Dynseq except that here processes are not allowed to send their messages all the time. Instead messages are buffered and every / millisecond they are packed and</p>

	<p>sent as one packed message. In this case we have chosen l to be one millisecond since it is less than the minimal expected one way user to user latency”</p> <p>Friedman at 5.</p> <p>“It turned out that packing messages improves both the latency and throughput of the protocols by two order of magnitudes and is therefore overwhelmingly more important for the performance than any other optimization that we used.”</p> <p>Friedman at 1.</p>
forming an aggregated message using said aggregated payload; and	<p>“This protocol is essentially the same as Dynseq except that here processes are not allowed to send their messages all the time. Instead messages are buffered and every 1 millisecond they are packed and sent as one packed message.”</p> <p>Friedman at 5.</p> <p>“If we denote this byte overhead by h then by packing m application messages as one message the headers overhead for these messages becomes only h instead of $h \times m$ which is required without packing.”</p> <p>Friedman at 12.</p>
transmitting, by said server via said unicast network, said aggregated message to a recipient host computer belonging to said first message group.	<p>“In IRC the channel has a role equivalent to that of the multicast group; their existence is dynamic (coming and going as people join and leave channels) and the actual conversation carried out on a channel is only sent to servers which are supporting users on a given channel. If there are multiple users on a server in the same channel, the message text is sent only once to that server and then sent to each client on the channel. This action is then repeated for each client-server combination until the original message has fanned out and reached each member of the channel.”</p> <p>IRC RFC at § 3.2.2</p>
2. The method of claim 1 wherein said time interval is	<p>“This protocol is essentially the same as Dynseq except that here processes are not allowed to send their messages all the time. Instead messages are buffered and every 1 millisecond they are packed and</p>

<p>a fixed period of time.</p>	<p>sent as one packed message.” Friedman at 5.</p>
<p>4. The method of claim 1 further comprising the step of creating, by one of said plurality of host computers, said first message group by sending a first control message to said server via said unicast network.</p>	<p>“To create a new channel or become part of an existing channel, a user is required to JOIN the channel. If the channel doesn't exist prior to joining, the channel is created and the creating user becomes a channel operator.” IRC RFC at § 1.3</p> <p>“Command: JOIN Parameters: <channel>{,<channel>} [<key>{,<key>}] The JOIN command is used by client to start listening a specific channel.” IRC RFC at § 4.2.1</p>
<p>5. The method of claim 4 further comprising the step of joining, by some of said plurality of host computers, said first message group by sending control messages via said unicast network to said server specifying said first message group.</p>	<p>“Command: JOIN Parameters: <channel>{,<channel>} [<key>{,<key>}] The JOIN command is used by client to start listening a specific channel.” IRC RFC at § 4.2.1</p>
<p>6. The method of claim 1 wherein said network is Internet and said server communicates with said plurality of host computers using a session layer protocol.</p>	<p>“The IRC protocol has been developed on systems using the TCP/IP network protocol, although there is no requirement that this remain the only sphere in which it operates.” IRC RFC at § 1.</p>

CC-E

Claim Chart comparing Claims 1-6 of U.S. Patent No. 5,822,523 to the disclosure in RING in view of Netrek

Prior art cited in this chart:

- Thomas A. Funkhouser, "RING: A Client-Server System for Multi-User Virtual Environments," Association of Computing Machinery, 1995 Symposium on Interactive 3D Graphics, Monterey CA. ("RING")
- *Server2.5pl4.tar.gz* ("Server Code") and *BRMH-1.7.tar.gz* ("Client Code") (source code dated no later than August 1994).

Reasons to Combine:

RING discloses communicating messages over a network. RING at Figs. 5 and 7, pp. 88, 87 and 91. Similarly, Netrek discloses clients and servers communicating over a network using messages. See *Server Code*, *Server\ntserv\newstartd.c* at lines 129-73, lines 179-311, lines 146-70; *Server\ntserv\main.c* at lines 135-43; *Server\ntserv\socket.c* at lines 442-88. Netrek further discloses aggregating packets to reduce the number of packets sent from the server. (e.g., "Idea: read from client often, send to client not so often"). *Server\ntserv\input.c* at lines 152-203; *Server\ntserv\redraw.c* at lines 21-115; *Server\ntserv\socket.c* at lines 603-90. A person of ordinary skill in the art, looking to increase network efficiency, would have looked to related methods of communicating messages over a network. Accordingly, a person of ordinary skill in the art would have looked to the aggregation teachings of Netrek to aggregate messages in RING to increase network efficiency.

Claims of the '523 Patent	Disclosure of RING and Netrek
<p>1. A method for providing group messages to a plurality of host computers connected over a unicast wide area communication network, comprising the steps of:</p>	<p>“This paper describes the client-server design, implementation and experimental results for a system that supports real-time visual interaction between a large number of users in a shared 3D virtual environment. The key feature of the system is that server-based visibility algorithms compute potential visual interactions between entities representing users in order to reduce the number of messages required to maintain consistent state among many workstations distributed across a wide-area network. When an entity changes state, update messages are sent only to workstations with entities that can potentially perceive the change- i.e., ones to which the update is visible.” RING at Abstract.</p> <p>“In a multi-user visual simulation system, users run an interactive interface program on (usually distinct) workstations connected to each other via a network.” RING at p. 85.</p> <p>“A difficult challenge in multi-user visual simulation is maintaining consistent state among a large number of workstations distributed over a wide-area network.” RING at p. 85.</p> <p>“In order to support very large numbers of users (> 1000) interacting simultaneously in a distributed virtual environment it is necessary to develop a system design and communication protocol that does not require sending update messages to all participating hosts for every entity state change.” RING at p. 86.</p> <p>“This paper describes a system (called RING) that supports interaction between large numbers of users in virtual environments with dense occlusion (e.g., buildings, cities, etc.). RING takes advantage of the fact that state changes must be propagated only to hosts containing entities that can possibly perceive the change- i.e., the one that can see it. Object-space visibility algorithms are used to compute the region of influence for each state change, and then update messages are sent only to the small subset of workstations to which the update is relevant.” RING at p. 86.</p>

“We have experimented with a variety of topologies for connecting RING clients and servers. For practical reasons, we have focused mainly on arrangements in which clients communicate with a single server. However, depending on the capabilities of available workstations and networks, clients can send messages to server(s) via unicast or multicast.”

RING at p. 91.

“In our first experiments with multi-user virtual environments, we used IP multicast to send update messages directly between clients. The general idea is to map entity properties into multicast groups, and send update messages only to relevant groups. For instance, Macedonia partitions a virtual world into a 2D grid of hexagonal shaped cells each of which is represented by a separate multicast group. Entities localize their visual interactions by sending updates only to the multicast group representing the cell in which they reside, and they listen only to multicast groups representing cells within some radius.

The multicast approach is similar to the RING client-server approach for wide-area networks. In both cases, intermediate machines may cull messages rather than propagating them to all participating workstations. However, using multicast, message culling is done by routers at the network layer, whereas, in RING, message culling is done by server machines at the application layer (see Figure 11). The advantages of the multicast approach are that: 1) fewer messages must be passed if clients are connected directly to a multicast-capable LAN (e.g., ethernet), and 2) latency is reduced due to faster message routing. The disadvantages are that: 1) delays associated with joining and leaving multicast groups make it impractical to use highly dynamic entity properties for multicast group mappings, 2) the number of unique multicast groups accessible to any one application may not be sufficient for complex virtual environments, and 3) multicast is not generally available across wide-area networks to many types of network computers (e.g., PCs with modems).

The advantage of the RING client-server approach is that very dynamic and complex *message processing* may be performed by servers. In contrast to multicast routers, which can only cull messages based on a relatively small, static set of multicast groups, RING servers can cull messages using high-level geometric algorithms and knowledge regarding a multiplicity of highly dynamic entity

attributes (e.g., location, orientation, velocity, etc.) and interaction types (e.g., visibility, sound, collision, etc.). Since RING servers can take advantage of knowledge regarding message semantics and the 3D geometry of the virtual environment directly, they can execute more effective and flexible culling algorithms than would be possible using only IP address and port mappings. Furthermore, unlike multicast routers, RING servers may process, augment, and alter messages in addition to culling them. For instance, RING servers already augment update messages with “Add” and “Remove” messages to inform clients that entities are entering or leaving their potentially visible areas.”
RING at p. 90-91.

providing a group messaging server coupled to said network, said server communicating with said plurality of host computers using said unicast network and maintaining a list of message groups, each message group containing at least one host computer;

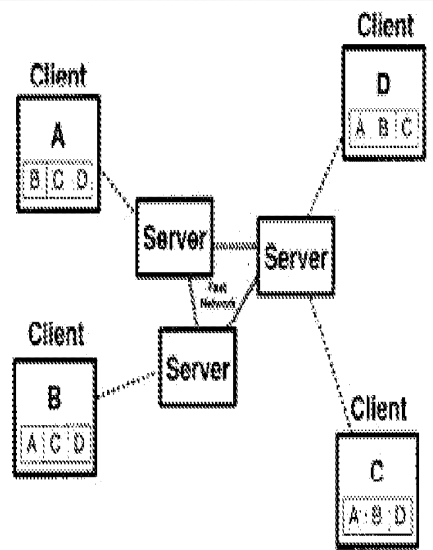


Figure 5: RING servers manage communication between clients, possibly culling, augmenting, or altering messages.

Figure 5 of RING at p. 87.

“We have experimented with a variety of topologies for connecting RING clients and servers. For practical reasons, we have focused mainly on arrangements in which clients communicate with a single server. However, depending on the capabilities of available workstations and networks, clients can send messages to server(s) via unicast or multicast.”

	<p>RING at p. 91.</p> <p>“Server-based message culling is implemented using precomputed line-of-sight visibility information. Prior to the multi-user simulation, the shared virtual environment is partitioned into a spatial subdivision of <i>cells</i> whose boundaries are comprised of the static, axis-aligned polygons of the virtual environment [1, 15]. A visibility precomputation is performed in which the set of cells potentially visible to each cell is determined by tracing beams of possible sight-lines through transparent cell boundaries [15, 16] (see Figure 6). During the multi-user simulation, servers keep track of which cells contain which entities by exchanging “periodic” update messages when entities cross cell boundaries. Real-time update messages are propagated only to servers and clients containing entities inside some cell visible to the one containing the updated entity. Since an entity’s visibility is conservatively over-estimated by the precomputed visibility of its containing cell, this algorithm allows servers to prove update messages quickly using cell visibility “look-ups” rather than more exact real-time entity visibility computations which would be too expensive on currently available workstations.”</p> <p>RING at p. 87.</p>
--	--

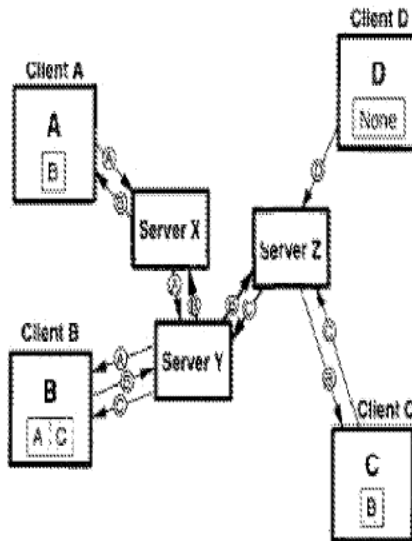


Figure 7: Flow of update messages (labeled arrows) for updates to entities A, B, C, and D arranged in a virtual environment as shown in Figure 6.

Figure 7 of RING at p. 88.

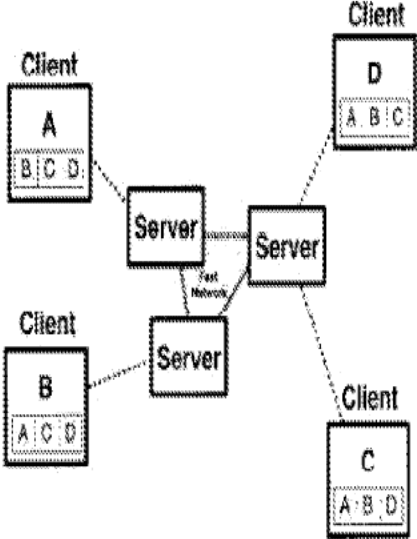
“Communication between clients is managed by *servers*. Clients do not send messages directly to other clients, but instead send them to servers which forward them to other client and server workstations participating in the same distributed simulation (see Figure 5). A key feature of this client-server design is that servers can process messages before propagating them to other workstations, culling, augmenting, or altering them. For instance, a server may determine that a particular update message is relevant only to a small subset of clients and the propagate the message only to those clients or their servers.”

RING at p. 87.

sending, by a plurality of host computers belonging to a first message group, messages to said server via

“RING represents a virtual environment as a set of independent entities each of which has a geometric description and a behavior. Some entities are static (e.g., terrain, buildings, etc.), whereas others have dynamic behavior that can be either autonomous (e.g., robots) or controlled by a user via input devices (e.g., vehicles). Distributed simulation occurs when multiple entities interact in a shared virtual

<p>said unicast network, said messages containing a payload portion and a portion for identifying said first message group;</p>	<p>environment by sending messages to one another to announce updates to their own geometry or behavior modifications to the shared environment, or impact on other entities.” RING at p. 87.</p> <p>“Communication between clients is managed by <i>servers</i>. Clients do not send messages directly to other clients, but instead send them to servers which forward them to other client and server workstations participating in the same distributed simulation (see Figure 5).” RING at p. 87.</p> <p>“We have experimented with a variety of topologies for connecting RING clients and servers. For practical reasons, we have focused mainly on arrangements in which <u>clients communicate with a single server</u>. However, depending on the capabilities of available workstations and networks, clients can send messages to server(s) via unicast or multicast.” RING at p. 91.</p> <p>“Update messages containing 40 bytes (message-type[4], entity-ID[4], target-position[12], target-orientation[12], positional velocity[4], and rotational-velocity[4] were generated for each entity once every 2.3 seconds on average with this ‘random’ navigational behavior.” RING at p. 89.</p>
---	--

	 <p data-bbox="738 955 1193 1029">Figure 5: RING servers manage communication between clients, possibly calling, augmenting, or altering messages.</p> <p data-bbox="828 1039 1088 1081">Figure 5 of RING at p. 87.</p>
<p data-bbox="211 1092 470 1386">aggregating, by said server in a time interval determined in accordance with a predefined criterion, said payload portions of said messages to create an aggregated payload;</p>	<pre data-bbox="487 1092 1412 1680"> 00000076 int timerDelay=200000; /* delay between sending stuff to client */ Server\ntserv\data.c at line 76 00000195 readFromClient(); Server\ntserv\input.c at line 195 00000152 input() 00000153 { 00000154 struct itimerval udt; 00000155 fd_set readfds; 00000156 static struct timeval poll = {2, 0}; 00000157 00000158 #ifdef DS </pre>

```

00000159     if (!me->p_process)
00000160 #endif
00000161     {
00000162         udt.it_interval.tv_sec = 0;
00000163         udt.it_interval.tv_usec = timerDelay;
00000164         udt.it_value.tv_sec = 0;
00000165         udt.it_value.tv_usec = timerDelay;
00000166         setitimer(ITIMER_REAL, &udt, 0);
00000167     }
00000168     SIGNAL(SIGALRM, setflag);
00000169
00000170         /* Idea: read from client often, send to client
not so often */
00000171         while (1) {
[...]
00000195             readFromClient();
[...]
00000203         }
00000204     }

```

Server\ntserv\input.c at lines 152-203

```

00000076 int timerDelay=200000; /* delay between sending stuff to
client */

```

Server\ntserv\data.c at line 76

```

00000603 updateClient()
00000604 {
[...]
00000608     static int skip = 0; /* If skip is set we skip next
update */
00000609 /* This can halve your updates */

```



```

00000610     if (send_short && skip) {
00000611         skip = 0; /* back to default */
00000612     if (bufptr==buf && (commMode!=COMM_UDP ||
udpbufptr==buf)) {
00000613         /* We sent nothing! We better send something to
wake him */
00000614         if (me->p_fuel < 61000)
00000615             sendClientPacket((CVOID) &clientSelfShort);
00000616         else
00000617             sendClientPacket((CVOID) &clientSelf);
00000618     }
00000619     flushSockBuf();
00000620     repCount++;
00000621     return;
00000622 }
[...]
00000630     if(send_short){
00000631     updatePlasmas();
00000632     updateStatus();
00000633     updateSelf();
00000634     updatePhasers();
00000635     updateShips();
00000636     updateTorps();
00000637     updatePlanets();
00000638     updateMessages();
00000639     }
[...]
00000657     if(send_short && (me->p_fuel < 61000))
00000658         sendClientPacket((CVOID) &clientSelfShort);
00000659     else
00000660 #endif

```

```
00000661 sendClientPacket((CVOID) &clientSelf);
00000662     }
[...]
00000685         sendClientPing();                               /* ping.c */
00000686 #endif
00000687
00000688     flushSockBuf();
00000689     repCount++;
00000690 }
Server\ntserv\socket.c at lines 603-90

00000052     intrupt();
Server\ntserv\input.c at lines 52

00000197         intrupt();
Server\ntserv\input.c at lines 197

00001390 updateMessages()
00001391 {
[... ]
00001590 }
Server\ntserv\socket.c at lines 1390-590

00001825 readFromClient()
00001826 {
[... ]
00001838     if (select(32,&readfds,0,0,&timeout) != 0) {
00001839 /* Read info from the xtrek client */
00001840 if (FD_ISSET(sock, &readfds)) {
00001841     retval += doRead(sock);
00001842 }
```

```

00001843 if (udpSock >= 0 && FD_ISSET(udpSock, &readfds)) {
00001844     V_UDPDIAG("Activity on UDP socket\n");
00001845     retval += doRead(udpSock);
00001846 }
00001847 }
00001848 return (retval != 0);           /* convert to 1/0 */
00001849 }
00001850
00001851 [...]
00001852 /* ripped out of above routine */
00001853 doRead(asock)
00001854 int asock;
00001855 {
00001856     struct timeval timeout;
00001857     [...]
00001858     /* Read info from the xtrek server */
00001859     count=read(asock,buf,BUFSIZ*2);
00001860     [...]
00001861     bufptr=buf;
00001862     while (bufptr < buf+count) {
00001863         [...]
00001864     while (size>count+(buf-bufptr)) {
00001865         /* We wait for up to twenty seconds for rest of
00001866         packet.
00001867         * If we don't get it, we assume the client
00001868         died.
00001869         */
00001870         timeout.tv_sec=20;
00001871         timeout.tv_usec=0;
00001872         /*readfds=1<<asock;*/
00001873         FD_ZERO(&readfds);

```

```

00001947     FD_SET(asock, &readfds);
[...]
00001956     temp=read(asock,buf+count,size-(count+(buf-bufptr)));
[...]
00001966     }
[...]
00002010         (*handlers[*bufptr].handler)(bufptr);
00002011     }
00002012     /* Otherwise we ignore the request */
00002013 } else {
00002014     ERROR(1,("Handler for packet %d not installed...\n",
*bufptr));
00002015 }
00002016     bufptr+=size;
00002017     if (bufptr>buf+BUFSIZ) {
00002018         bcopy(buf+BUFSIZ, buf, BUFSIZ);
00002019         if (count==BUFSIZ*2) {
00002020             /*readfds = 1<<asock;*/
00002021             FD_ZERO(&readfds);
00002022             FD_SET(asock, &readfds);
00002023             if (select(32,&readfds,0,0,&timeout)) {
00002024                 temp=read(asock,buf+BUFSIZ,BUFSIZ);
00002025                 count=BUFSIZ+temp;
[...]
00002034         } else {
00002035             count=BUFSIZ;
00002036         }
00002037     } else {
00002038         count -=BUFSIZ;
00002039     }
00002040         bufptr-=BUFSIZ;

```

```

00002041         }
00002042     }
00002043     return(1);
00002044 }
Server\ntserv\socket.c at lines 1825-2044

00001825 readFromClient()
00001826 {
00001827     [...]
00001838     if (select(32,&readfds,0,0,&timeout) != 0) {
00001839     /* Read info from the xtrek client */
00001840     if (FD_ISSET(sock, &readfds)) {
00001841         retval += doRead(sock);
00001842     }
00001843     if (udpSock >= 0 && FD_ISSET(udpSock, &readfds)) {
00001844         V_UDPDIAG(("Activity on UDP socket\n"));
00001845         retval += doRead(udpSock);
00001846     }
00001847     }
00001848     return (retval != 0);           /* convert to 1/0 */
00001849 }
00001850
00001851     [...]
00001852     /* ripped out of above routine */
00001853     doRead(asock)
00001854     int asock;
00001855     {
00001856         struct timeval timeout;
00001857     [...]
00001877         /* Read info from the xtrek server */
00001878         count=read(asock,buf,BUFSIZ*2);

```

```

[...]  

00001916     bufptr=buf;  

00001917     while (bufptr < buf+count) {  

[...]  

00001939     while (size>count+(buf-bufptr)) {  

00001940         /* We wait for up to twenty seconds for rest of  

packet.  

00001941             * If we don't get it, we assume the client  

died.  

00001942             */  

00001943         timeout.tv_sec=20;  

00001944             timeout.tv_usec=0;  

00001945             /*readfds=1<<asock;*/  

00001946         FD_ZERO(&readfds);  

00001947         FD_SET(asock, &readfds);  

[...]  

00001956         temp=read(asock,buf+count,size-(count+(buf-bufptr)));  

[...]  

00001966     }  

[...]  

00002010         (*(handlers[*bufptr].handler))(bufptr);  

00002011     }  

00002012     /* Otherwise we ignore the request */  

00002013 } else {  

00002014     ERROR(1,("Handler for packet %d not installed...\n",  

*bufptr));  

00002015 }  

00002016     bufptr+=size;  

00002017     if (bufptr>buf+BUFSIZ) {  

00002018         bcopy(buf+BUFSIZ, buf, BUFSIZ);  

00002019         if (count==BUFSIZ*2) {

```

```
00002020      /*readfds = 1<<asock;*/
00002021      FD_ZERO(&readfds);
00002022      FD_SET(asock, &readfds);
00002023      if (select(32,&readfds,0,0,&timeout)) {
00002024          temp=read(asock,buf+BUFSIZ,BUFSIZ);
00002025          count=BUFSIZ+temp;
00002026      [...]
00002034      } else {
00002035          count=BUFSIZ;
00002036      }
00002037      } else {
00002038          count -=BUFSIZ;
00002039      }
00002040          bufptr-=BUFSIZ;
00002041      }
00002042      }
00002043      return(1);
00002044      2044
Server\ntserv\socket.c at lines 1825-2044

00001390 updateMessages()
00001391 {
00001392 [...]
00001393 }
00001394 }
Server\ntserv\socket.c at lines 1390-590

00001603 sendClientPacket(packet)
00001604 /* Pick a random type for the packet */
00001605 struct player_spacket *packet;
00001606 {
00001607 [...]
00001608 }
```

```

00001618  /*
00001619      * If we're dead, dying, or just born, we definitely
want the transmission
00001620      * to get through (otherwise we can get stuck). I don't
think this will
00001621      * be a problem for anybody, though it might hang for a
bit if the TCP
00001622      * connection is bad.
00001623      */
00001624  /* Okay, now I'm not so sure. Whatever. */
00001625  if (oldstatus != PALIVE || (me != NULL && me->p_status
!= PALIVE))
00001626  orig_type = packet->type | 0x80; /* pretend it's critical
*/
00001627 #endif
00001628      if (packet->type<1 || packet->type>NUM_SIZES ||
00001629          sizes[(int)packet->type]==0) {
00001630  ERROR(1,("Attempt to send strange packet %d %d\n", packet-
>type,NUM_SIZES));
00001631  return;
00001632      }
00001633      packetsSent[(int)packet->type]++;
00001634      if (commMode == COMM_TCP || (commMode == COMM_UDP &&
udpMode == MODE_TCP)) {
00001635  /*
00001636      * business as usual
00001637      */
[...]
```

```

00001647  bcopy(packet, bufptr, size);
00001648  bufptr+=size;
00001649
```



```

00001650     } else {
00001651     /*
00001652     * do UDP stuff unless it's a "critical" packet
00001653     * (note that both kinds get a sequence number appended)
(FIX)
00001654     */
[...]
00001728 default:
00001729     /* these are critical packets; send them via TCP */
00001730     size=sizes[packet->type];
00001731     if (bufptr-buf+size >= BUFSIZE) {
00001732         t=bufptr-buf;
00001733         if (gwrite(sock, buf, t) != t) {
00001734             perror("TCP gwrite failed, client marked dead");
00001735             clientDead=1;
00001736         }
00001737         bufptr=buf /*+ addSequence(buf)*/;
00001738     }
00001739     bcopy(packet, bufptr, size);
00001740     bufptr+=size;
00001741     break;
00001742 }
00001743 }
00001744 }

```

Server\ntserv\socket.c at lines 1603-744

```

00001125 updateTorps()
00001126 {
[...]
00001132     for (i=0, torp=torps, tpi=clientTorpsInfo,
tp=clientTorps;

```

<pre> 00001133 i<MAXPLAYER*MAXTORP; 00001134 i++, torp++, tpi++, tp++) { [...] 00001142 sendClientPacket (tpi); [...] 00001151 sendClientPacket (tp); [...] 00001191 } 00001192 } </pre> <p>Server\ntserv\socket.c at lines 1125-92</p> <p>“Furthermore unlike multicast routers, RING servers may process, augment, and alter messages in addition to culling them. For instance, RING servers already augment update messages with “Add” and “Remove” messages to inform clients that entities are entering or leaving their potentially visible sets.”</p> <p>RING at p. 91.</p> <p>“RING servers allow each client workstation to maintain surrogates for only the subset of remote entities visible to at least one entity local to the client. . . . To support this feature, servers send their client an “Add” message each time a remote entity enters a cell potentially visible to one of the client’s local entities for the first time. A “Remove” message is sent when the server determines that an entity has left the client’s visible region. As entities move through the environment, servers augment update messages with “Add” and “Remove” messages notifying clients that remote entities have become relevant or irrelevant to the client’s local entities.”</p> <p>RING at p. 88.</p> <p>“Finally, time critical computing algorithms can be used to determine an ‘optimal’ set of messages to send to each client based on network connection bandwidths, workstation processing capabilities, and many other real-time performance factors (i.e., in a manner similar to that used in [8]).”</p> <p>RING at p. 91.</p>
--

	<p>“During the multi-user simulation, servers keep track of which cells contain which entities by exchanging ‘periodic’ update messages when entities cross cell boundaries. Real-time update messages are propagated only to servers and clients containing entities inside some cell visible to the one containing the updated entity. Since an entity’s visibility is conservatively over-estimated by the precomputed visibility of its containing cell, this algorithm allows servers to process update messages quickly using cell visibility ‘look-ups’ rather than more exact real-time entity visibility computations which would be too expensive on currently available workstations.”</p> <p>RING at p. 87.</p> <p>“Rather than sending messages directly between clients, RING routes each on through at least one server, and possibly two. Computations are performed in the servers before messages are propagated further adding to latency.”</p> <p>RING at p. 88.</p>
forming an aggregated message using said aggregated payload; and	<pre> 00001747 flushSockBuf() 00001748 { [...] 00001755 if (gwrite(sock, buf, t) != t) { 00001756 perror("std flush gwrite failed, client marked dead"); 00001757 clientDead=1; 00001758 } [...] 00001782 if (gwrite(udpSock, udpbuf, t) != t){ 00001783 perror("UDP flush gwrite failed, client marked dead once more"); 00001784 #ifdef EXTRA_GB 00001785 clientDead=1; 00001786 #endif 00001787 UDPDIAG("*** UDP disconnected for %s\n", me->p_name)); 00001788 printUdpInfo(); 00001789 closeUdpConn(); 00001790 commMode = COMM_TCP; </pre>

```
00001791 }
[...]
```

Server\ntserv.c at lines 1747-802

```
00002607 gwrite(fd, wbuf, size)
00002608 int fd;
00002609 char *wbuf;
00002610 size_t size;
00002611 {
00002625     while (bytes>0) {
00002626     n = write(fd, wbuf, bytes);
[...]
```

Server\ntserv.c at lines 2607-73

```
00000603 updateClient()
00000604 {
[...]
```

```
00000608     static int skip = 0; /* If skip is set we skip next
update */
00000609 /* This can halve your updates */
00000610     if (send_short && skip) {
00000611         skip = 0; /* back to default */
00000612     if (bufptr==buf && (commMode!=COMM_UDP ||
udpbuflptr==buf)) {
00000613         /* We sent nothing! We better send something to
wake him */
00000614         if (me->p_fuel < 61000)
```

```

00000615             sendClientPacket((CVOID) &clientSelfShort);
00000616         else
00000617             sendClientPacket((CVOID) &clientSelf);
00000618     }
00000619     flushSockBuf();
00000620     repCount++;
00000621     return;
00000622 }
[...]
00000630     if(send_short){
00000631 updatePlasmas();
00000632 updateStatus();
00000633 updateSelf();
00000634 updatePhasers();
00000635 updateShips();
00000636 updateTorps();
00000637 updatePlanets();
00000638 updateMessages();
00000639     }
[...]
00000657     if(send_short && (me->p_fuel < 61000))
00000658         sendClientPacket((CVOID) &clientSelfShort);
00000659     else
00000660 #endif
00000661 sendClientPacket((CVOID) &clientSelf);
00000662     }
[...]
00000685     sendClientPing();                               /* ping.c */
00000686 #endif
00000687
00000688     flushSockBuf();

```

```
00000689     repCount++;
00000690 }
Server\ntserv\socket.c at lines 603-90

00001603 sendClientPacket(packet)
00001604 /* Pick a random type for the packet */
00001605 struct player_spacket *packet;
00001606 {
00001607     [...]
00001618     /*
00001619      * If we're dead, dying, or just born, we definitely
00001620      * want the transmission
00001621      * to get through (otherwise we can get stuck). I don't
00001622      * think this will
00001623      * be a problem for anybody, though it might hang for a
00001624      * bit if the TCP
00001625      * connection is bad.
00001626      */
00001627     /* Okay, now I'm not so sure. Whatever. */
00001628     if (oldstatus != PALIVE || (me != NULL && me->p_status
00001629     != PALIVE))
00001630     orig_type = packet->type | 0x80; /* pretend it's critical
00001631     */
00001632     #endif
00001633     if (packet->type<1 || packet->type>NUM_SIZES ||
00001634     sizes[(int)packet->type]==0) {
00001635     ERROR(1,("Attempt to send strange packet %d %d\n", packet-
00001636     >type,NUM_SIZES));
00001637     return;
00001638     }
00001639     packetsSent[(int)packet->type]++;
```

```

00001634     if (commMode == COMM_TCP || (commMode == COMM_UDP &&
udpMode == MODE_TCP)) {
00001635     /*
00001636     * business as usual
00001637     */
00001638     [...]
00001647     bcopy(packet, bufptr, size);
00001648     bufptr+=size;
00001649
00001650     } else {
00001651     /*
00001652     * do UDP stuff unless it's a "critical" packet
00001653     * (note that both kinds get a sequence number appended)
(FIX)
00001654     */
00001655     [...]
00001728     default:
00001729         /* these are critical packets; send them via TCP */
00001730         size=sizes[packet->type];
00001731         if (bufptr-buf+size >= BUFSIZE) {
00001732             t=bufptr-buf;
00001733             if (gwrite(sock, buf, t) != t) {
00001734                 perror("TCP gwrite failed, client marked dead");
00001735                 clientDead=1;
00001736             }
00001737             bufptr=buf /*+ addSequence(buf)*/;
00001738         }
00001739         bcopy(packet, bufptr, size);
00001740         bufptr+=size;
00001741         break;
00001742     }

```

	<pre>00001743 } 00001744 }</pre> <p>Server\ntserv\socket.c at lines 1603-744</p> <p>“Furthermore unlike multicast routers, RING servers may process, augment, and alter messages in addition to culling them. For instance, RING servers already augment update messages with “Add” and “Remove” messages to inform clients that entities are entering or leaving their potentially visible sets.”</p> <p>RING at p. 91.</p> <p>“RING servers allow each client workstation to maintain surrogates for only the subset of remote entities visible to at least one entity local to the client. . . . To support this feature, servers send their client an “Add” message each time a remote entity enters a cell potentially visible to one of the client’s local entities for the first time. A “Remove” message is sent when the server determines that an entity has left the client’s visible region. As entities move through the environment, server augment update messages with “Add” and “Remove” messages notifying clients that remote entities have become relevant or irrelevant to the client’s local entities.”</p> <p>RING at p. 88.</p>
<p>transmitting, by said server via said unicast network, said aggregated message to a recipient host computer belonging to said first message group.</p>	<pre>00001603 sendClientPacket(packet) 00001604 /* Pick a random type for the packet */ 00001605 struct player_spacket *packet; 00001606 { 00001607 [...] 00001639 if (bufptr-buf+size >= BUFSIZE) { 00001640 t=bufptr-buf; 00001641 if (gwrite(sock, buf, t) != t) { 00001642 perror("std gwrite failed, client marked dead "); 00001643 clientDead=1; 00001644 } 00001645 bufptr=buf; 00001646 }</pre>


```
00001647 bcopy(packet, bufptr, size);
00001648 bufptr+=size;
00001649
00001650     } else {
[...]
```

```
00001731     if (bufptr-buf+size >= BUFSIZE) {
00001732         t=bufptr-buf;
00001733         if (gwrite(sock, buf, t) != t) {
00001734             perror("TCP gwrite failed, client marked dead");
00001735             clientDead=1;
00001736         }
00001737         bufptr=buf /*+ addSequence(buf)*/;
00001738     }
00001739     bcopy(packet, bufptr, size);
00001740     bufptr+=size;
00001741     break;
00001742 }
00001743 }
00001744 }
```

Server\ntserv\socket.c at lines 1603-744

```
00000603 updateClient()
00000604 {
[...]
```

```
00000688     flushSockBuf();
00000689     repCount++;
00000690 }
```

Server\ntserv\socket.c at lines 603-90

```
00001747 flushSockBuf()
00001748 {
```

```

[...]
00001755 if (gwrite(sock, buf, t) != t) {
00001756     perror("std flush gwrite failed, client marked dead");
00001757     clientDead=1;
00001758 }
[...]
00001782 if (gwrite(udpSock, udpbuf, t) != t){
00001783     perror("UDP flush gwrite failed, client marked dead
once more");
[...]
00001791 }
[...]
00001802 }
Server\ntserv\socket.c at lines 1747-802

00002607 gwrite(fd, wbuf, size)
00002608 int fd;
00002609 char *wbuf;
00002610 size_t size;
00002611 {
[...]
00002625     while (bytes>0) {
00002626 n = write(fd, wbuf, bytes);
00002627 if (count++ > 100) {
00002628     ERROR(1,("Gwrite hosed: too many writes
(%d)\n",getpid()));
00002629     clientDead = 1;
00002630     return (-1);
00002631 }
[...]
00002671     }

```

	<pre> 00002672 return(orig); 00002673 } Server\ntserv\socket.c at lines 2607-73 00001125 updateTorps() 00001126 { [...]</pre> <pre> 00001132 for (i=0, torp=torps, tpi=clientTorpsInfo, 00001133 tp=clientTorps; 00001134 i<MAXPLAYER*MAXTORP; 00001135 i++, torp++, tpi++, tp++) { [...]</pre> <pre> 00001142 sendClientPacket(tpi); [...]</pre> <pre> 00001151 sendClientPacket(tp); [...]</pre> <pre> 00001191 } 00001192 } Server\ntserv\socket.c at lines 1125-92 “Communication between clients is managed by <i>servers</i>. Clients do not send messages directly to other clients, but instead send them to servers which forward them to other client and server workstations participating in the same distributed simulation (see Figure 5).” RING at p. 87. “ We have experimented with a variety of topologies for connecting RING clients and servers. For practical reasons, we have focused mainly on arrangements in which clients communicate with a single server. However, depending on the capabilities of available workstations and networks, clients can send messages to server(s) via unicast or multicast.” RING at p. 91. </pre>
2. The method of claim 1	“During the multi-user simulation, servers keep track of which cells contain which entities by

<p>wherein said time interval is a fixed period of time.</p>	<p>exchanging “periodic” update messages when entities cross cell boundaries.” RING at p. 87.</p> <p>“This paper describes the client-server design, implementation and experimental results for a system that supports real-time visual interaction between a large number of users in a shared 3D virtual environment. The key feature of the system is that server-based visibility algorithms compute potential visual interactions between entities representing users in order to reduce the number of messages required to maintain consistent state among many workstations distributed across a wide-area network. When an entity changes state, update messages are sent only to workstations with entities that can potentially perceive the change- i.e., ones to which the update is visible.” RING at Abstract.</p> <p>“In a multi-user visual simulation system, users run an interactive interface program on (usually distinct) workstations connected to each other via a network.” RING at p. 85.</p> <p>“A difficult challenge in multi-user visual simulation is maintaining consistent state among a large number of workstations distributed over a wide-area network.” RING at p. 85.</p> <p>“In order to support very large numbers of users (> 1000) interacting simultaneously in a distributed virtual environment it is necessary to develop a system design and communication protocol that does not require sending update messages to all participating hosts for every entity state change.” RING at p. 86.</p> <p>“This paper describes a system (called RING) that supports interaction between large numbers of users in virtual environments with dense occlusion (e.g., buildings, cities, etc.). RING takes advantage of the fact that state changes must be propagated only to hosts containing entities that can possibly perceive the change- i.e., the one that can see it. Object-space visibility algorithms are used to compute the region of influence for each state change, and then update messages are sent only to the small subset of workstations to which the update is relevant.”</p>
--	--

RING at p. 86.

“We have experimented with a variety of topologies for connecting RING clients and servers. For practical reasons, we have focused mainly on arrangements in which clients communicate with a single server. However, depending on the capabilities of available workstations and networks, clients can send messages to server(s) via unicast or multicast.”

RING at p. 91.

“In our first experiments with multi-user virtual environments, we used IP multicast to send update messages directly between clients. The general idea is to map entity properties into multicast groups, and send update messages only to relevant groups. For instance, Macedonia partitions a virtual world into a 2D grid of hexagonal shaped cells each of which is represented by a separate multicast group. Entities localize their visual interactions by sending updates only to the multicast group representing the cell in which they reside, and they listen only to multicast groups representing cells within some radius.

The multicast approach is similar to the RING client-server approach for wide-area networks. In both cases, intermediate machines may cull messages rather than propagating them to all participating workstations. However, using multicast, message culling is done by routers at the network layer, whereas, in RING, message culling is done by server machines at the application layer (see Figure 11). The advantages of the multicast approach are that: 1) fewer messages must be passed if clients are connected directly to a multicast-capable LAN (e.g., ethernet), and 2) latency is reduced due to faster message routing. The disadvantages are that: 1) delays associated with joining and leaving multicast groups make it impractical to use highly dynamic entity properties for multicast group mappings, 2) the number of unique multicast groups accessible to any one application may not be sufficient for complex virtual environments, and 3) multicast is not generally available across wide-area networks to many types of network computers (e.g., PCs with modems).

The advantage of the RING client-server approach is that very dynamic and complex *message processing* may be performed by servers. In contrast to multicast routers, which can only cull messages based on a relatively small, static set of multicast groups, RING servers can cull messages

	<p>using high-level geometric algorithms and knowledge regarding a multiplicity of highly dynamic entity attributes (e.g., location, orientation, velocity, etc.) and interaction types (e.g., visibility, sound, collision, etc.). Since RING servers can take advantage of knowledge regarding message semantics and the 3D geometry of the virtual environment directly, they can execute more effective and flexible culling algorithms than would be possible using only IP address and port mappings. Furthermore, unlike multicast routers, RING servers may process, augment, and alter messages in addition to culling them. For instance, RING servers already augment update messages with “Add” and “Remove” messages to inform clients that entities are entering or leaving their potentially visible areas.”</p> <p>RING at p. 90-91.</p>
<p>3. The method of claim 1 wherein said time interval corresponds to a time for said server to receive at least one message from each host computer belonging to said first message group.</p>	<pre> 00000195 readFromClient(); Server\ntserv\input.c at line 195 00000152 input() 00000153 { 00000154 struct itimerval udt; 00000155 fd_set readfds; 00000156 static struct timeval poll = {2, 0}; 00000157 00000158 #ifdef DS 00000159 if (!me->p_process) 00000160 #endif 00000161 { 00000162 udt.it_interval.tv_sec = 0; 00000163 udt.it_interval.tv_usec = timerDelay; 00000164 udt.it_value.tv_sec = 0; 00000165 udt.it_value.tv_usec = timerDelay; 00000166 setitimer(ITIMER_REAL, &udt, 0); 00000167 } 00000168 SIGNAL(SIGALRM, setflag); 00000169 00000170 /* Idea: read from client often, send to client </pre>

```
not so often */
```

```
00000171         while (1) {  
[...]  
00000195             readFromClient();  
[...]  
00000203         }  
00000204     }
```

Server\ntserv\input.c at lines 152-203

```
00000076 int     timerDelay=200000; /* delay between sending stuff to  
client */
```

Server\ntserv\data.c at line 76

```
00000603 updateClient()  
00000604 {  
[...]  
00000608     static int skip = 0; /* If skip is set we skip next  
update */  
00000609 /* This can halve your updates */  
00000610     if (send_short && skip) {  
00000611         skip = 0; /* back to default */  
00000612     if (bufptr==buf && (commMode!=COMM_UDP ||  
udpbufptr==buf)) {  
00000613         /* We sent nothing! We better send something to  
wake him */  
00000614         if (me->p_fuel < 61000)  
00000615             sendClientPacket((CVOID) &clientSelfShort);  
00000616         else  
00000617             sendClientPacket((CVOID) &clientSelf);  
00000618     }  
00000619     flushSockBuf();
```

```
00000620     repCount++;
00000621     return;
00000622 }
[...]
00000630     if(send_short){
00000631     updatePlasmas();
00000632     updateStatus();
00000633     updateSelf();
00000634     updatePhasers();
00000635     updateShips();
00000636     updateTorps();
00000637     updatePlanets();
00000638     updateMessages();
00000639     }
[...]
00000657     if(send_short && (me->p_fuel < 61000))
00000658         sendClientPacket((CVOID) &clientSelfShort);
00000659     else
00000660 #endif
00000661     sendClientPacket((CVOID) &clientSelf);
00000662     }
[...]
00000685     sendClientPing();           /* ping.c */
00000686 #endif
00000687
00000688     flushSockBuf();
00000689     repCount++;
00000690 }
Server\ntserv\socket.c at lines 603-90
00000052     intrupt();
```



```
Server\ntserv\input.c at lines 52

00000197      intrupt ();
Server\ntserv\input.c at lines 197

00001390 updateMessages ()
00001391 {
[...]
```

```
Server\ntserv\socket.c at lines 1390-590

00001390 updateMessages ()
00001391 {
[...]
```

```
00001563      if (cur->m_from==DOOSHMSG) msg.m_from=255; /* god */
00001564      if ((cur->m_from < 0) || (cur->m_from > MAXPLAYER))
00001565          sendClientPacket((CVOID) &msg);
00001566      else if (cur->m_flags & MALL && !(ignored[cur-
>m_from] & MALL))
00001567          sendClientPacket((CVOID) &msg);
00001568      else if (cur->m_flags & MTEAM && !(ignored[cur-
>m_from] & MTEAM)){
00001569          sendClientPacket((CVOID) &msg);
00001570      }
[...]
```

```
00001590 }
Server\ntserv\socket.c at lines 1390-590

00001825 readFromClient ()
00001826 {
[...]
```

```

00001838     if (select(32,&readfds,0,0,&timeout) != 0) {
00001839 /* Read info from the xtrem client */
00001840 if (FD_ISSET(sock, &readfds)) {
00001841     retval += doRead(sock);
00001842 }
00001843 if (udpSock >= 0 && FD_ISSET(udpSock, &readfds)) {
00001844     V_UDPDIAG("Activity on UDP socket\n");
00001845     retval += doRead(udpSock);
00001846 }
00001847 }
00001848     return (retval != 0);           /* convert to 1/0 */
00001849 }
00001850
00001851 [...]
00001852 /* ripped out of above routine */
00001853 doRead(asock)
00001854 int asock;
00001855 {
00001856     struct timeval timeout;
00001857     [...]
00001858     /* Read info from the xtrem server */
00001859     count=read(asock,buf,BUFSIZ*2);
00001860     [...]
00001861     bufptr=buf;
00001862     while (bufptr < buf+count) {
00001863     [...]
00001864     while (size>count+(buf-bufptr)) {
00001865     /* We wait for up to twenty seconds for rest of
00001866     packet.
00001867     * If we don't get it, we assume the client
00001868     died.

```

```

00001942         */
00001943     timeout.tv_sec=20;
00001944         timeout.tv_usec=0;
00001945         /*readfds=1<<asock;*/
00001946     FD_ZERO(&readfds);
00001947     FD_SET(asock, &readfds);
00001948     [...]
00001956     temp=read(asock,buf+count,size-(count+(buf-bufptr)));
00001957     [...]
00001966     }
00001967     [...]
00002010         (*(handlers[*bufptr].handler))(bufptr);
00002011     }
00002012     /* Otherwise we ignore the request */
00002013     } else {
00002014         ERROR(1,("Handler for packet %d not installed...\n",
00002015 *bufptr));
00002016     }
00002017         bufptr+=size;
00002018         if (bufptr>buf+BUFSIZ) {
00002019             bcopy(buf+BUFSIZ, buf, BUFSIZ);
00002020             if (count==BUFSIZ*2) {
00002021                 /*readfds = 1<<asock;*/
00002022                 FD_ZERO(&readfds);
00002023                 FD_SET(asock, &readfds);
00002024                 if (select(32,&readfds,0,0,&timeout)) {
00002025                     temp=read(asock,buf+BUFSIZ,BUFSIZ);
00002026                     count=BUFSIZ+temp;
00002027                 }
00002028                 [...]
00002034             } else {
00002035                 count=BUFSIZ;

```

	<pre> 00002036 } 00002037 } else { 00002038 count -=BUFSIZ; 00002039 } 00002040 bufptr-=BUFSIZ; 00002041 } 00002042 } 00002043 return(1); 00002044 } </pre> <p>Server\ntserv\socket.c at lines 1825-2044</p>
<p>4. The method of claim 1 further comprising the step of creating, by one of said plurality of host computers, said first message group by sending a first control message to said server via said unicast network.</p>	<p>“In our first experiments with multi-user virtual environments, we used IP multicast to send update messages directly between clients. The general idea is to map entity properties into multicast groups, and send update messages only to relevant groups. For instance, Macedonia partitions a virtual world into a 2D grid of hexagonal shaped cells each of which is represented by a separate multicast group. Entities localize their visual interactions by sending updates only to the multicast group representing the cell in which the reside, and they listen only to multicast groups representing cells within some radius.</p> <p>The multicast approach is similar to the RING client-server approach for wide-area networks. In both cases, intermediate machines may cull messages rather than propagating them to all participating workstations. However, using multicast, message culling is done by routers at the network layer, whereas, in RING, message culling is done by server machines at the application layer (see Figure 11). The advantages of the multicast approach are that: 1) fewer messages must be passed if clients are connected directly to a multicast-capable LAN (e.g., ethernet), and 2) latency is reduced due to faster message routing. The disadvantages are that: 1) delays associated with joining and leaving multicast groups make it impractical to use highly dynamic entity properties for multicast group mappings, 2) the number of unique multicast groups accessible to any one application may not be sufficient for complex virtual environments, and 3) multicast is not generally available across wide-area networks to many types of networks computers (e.g., PCs with modems).</p>

	<p>The advantage of the RING client-server approach is that very dynamic and complex <i>message processing</i> may be performed by servers. In contrast to multicast routers, which can only cull messages based on a relatively small, static set of multicast groups, RING servers can cull messages using high-level geometric algorithms and knowledge regarding a multiplicity of highly dynamic entity attributes (e.g., location, orientation, velocity, etc.) and interaction types (e.g., visibility, sound, collision, etc.). Since RING servers can take advantage of knowledge regarding message semantics and the 3D geometry of the virtual environment directly, they can execute more effective and flexible culling algorithms than would be possible using only IP address and port mappings. Furthermore, unlike multicast routers, RING servers may process, augment, and alter messages in addition to culling them. For instance, RING servers already augment update messages with “Add” and “Remove” messages to inform clients that entities are entering or leaving their potentially visible areas.”</p> <p>RING at p. 90-91.</p>
<p>5. The method of claim 4 further comprising the step of joining, by some of said plurality of host computers, said first message group by sending control messages via said unicast network to said server specifying said first message group.</p>	<p>“Update messages containing 40 bytes (message-type[4], entity-ID[4], target-position[12], target-orientation[12], positional velocity[4], and rotational-velocity[4]) were generated for each entity once every 2.3 seconds on average with this ‘random’ navigational behavior.”</p> <p>RING at p. 89.</p> <p>“In our first experiments with multi-user virtual environments, we used IP multicast to send update messages directly between clients. The general idea is to map entity properties into multicast groups, and send update messages only to relevant groups. For instance, Macedonia partitions a virtual world into a 2D grid of hexagonal shaped cells each of which is represented by a separate multicast group. Entities localize their visual interactions by sending updates only to the multicast group representing the cell in which they reside, and they listen only to multicast groups representing cells within some radius.</p> <p>The multicast approach is similar to the RING client-server approach for wide-area networks. In both cases, intermediate machines may cull messages rather than propagating them to all participating workstations. However, using multicast, message culling is done by routers at the network layer, whereas, in RING, message culling is done by server machines at the application layer (see Figure 11). The advantages of the multicast approach are that: 1) fewer messages must be passed if clients are</p>

	<p>connected directly to a multicast-capable LAN (e.g., ethernet), and 2) latency is reduced due to faster message routing. The disadvantages are that: 1) delays associated with joining and leaving multicast groups make it impractical to use highly dynamic entity properties for multicast group mappings, 2) the number of unique multicast groups accessible to any one application may not be sufficient for complex virtual environments, and 3) multicast is not generally available across wide-area networks to many types of networks computers (e.g., PCs with modems).</p> <p>The advantage of the RING client-server approach is that very dynamic and complex <i>message processing</i> may be performed by servers. In contrast to multicast routers, which can only cull messages based on a relatively small, static set of multicast groups, RING servers can cull messages using high-level geometric algorithms and knowledge regarding a multiplicity of highly dynamic entity attributes (e.g., location, orientation, velocity, etc.) and interaction types (e.g., visibility, sound, collision, etc.). Since RING servers can take advantage of knowledge regarding message semantics and the 3D geometry of the virtual environment directly, they can execute more effective and flexible culling algorithms than would be possible using only IP address and port mappings. Furthermore, unlike multicast routers, RING servers may process, augment, and alter messages in addition to culling them. For instance, RING servers already augment update messages with “Add” and “Remove” messages to inform clients that entities are entering or leaving their potentially visible areas.”</p> <p>RING at p. 90-91.</p>
<p>6. The method of claim 1 wherein said network is Internet and said server communicates with said plurality of host computers using a session layer protocol.</p>	<p>“The system runs on Silicon Graphics workstations and uses UDP/IP datagrams for message passing.”</p> <p>RING at p. 89.</p> <p>“A difficult challenge in multi-user visual simulation is maintaining consisted state among a large number of workstations distributed over a wide-area network.”</p> <p>RING at p. 85.</p>

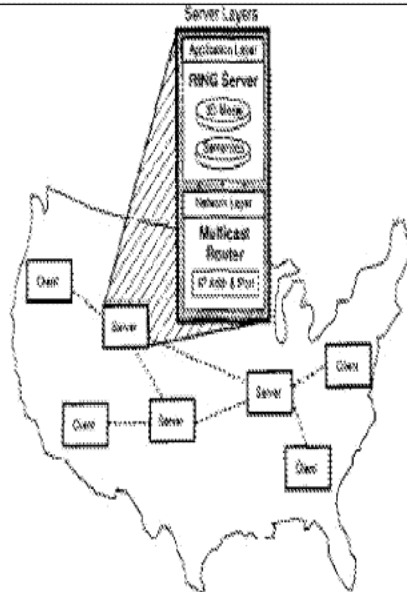


Figure 11: RING servers process messages in the application layer using 3D model and semantic information. Multicast routers use only IP addressing in the network layer.

Figure 11 of RING at p. 91.

“However, using multicast, message culling is done by routers at the network layer, whereas, in RING, message culling is done by server machines at the application layer (see Figure 11).”

RING at p. 90.

“This paper describes the client-server design, implementation and experimental results for a system that supports real-time visual interaction between a large number of users in a shared 3D virtual environment. The key feature of the system is that server-based visibility algorithms compute potential visual interactions between entities representing users in order to reduce the number of messages required to maintain consistent state among many workstations distributed across a wide-area network. When an entity changes state, update messages are sent only to workstations with entities that can potentially perceive the change- i.e., ones to which the update is visible.”

RING at Abstract.

	<p>“In a multi-user visual simulation system, users run an interactive interface program on (usually distinct) workstations connected to each other via a network.” RING at p. 85.</p> <p>“A difficult challenge in multi-user visual simulation is maintaining consistent state among a large number of workstations distributed over a wide-area network.” RING at p. 85.</p> <p>“In order to support very large numbers of users (> 1000) interacting simultaneously in a distributed virtual environment it is necessary to develop a system design and communication protocol that does not require sending update messages to all participating hosts for every entity state change.” RING at p. 86.</p> <p>“This paper describes a system (called RING) that supports interaction between large numbers of users in virtual environments with dense occlusion (e.g., buildings, cities, etc.). RING takes advantage of the fact that state changes must be propagated only to hosts containing entities that can possibly perceive the change- i.e., the one that can see it. Object-space visibility algorithms are used to compute the region of influence for each state change, and then update messages are sent only to the small subset of workstations to which the update is relevant.” RING at p. 86.</p> <p>“We have experimented with a variety of topologies for connecting RING clients and servers. For practical reasons, we have focused mainly on arrangements in which <u>clients communicate with a single server</u>. However, depending on the capabilities of available workstations and networks, clients can send messages to server(s) via unicast or multicast.” RING at p. 91.</p> <p>“In our first experiments with multi-user virtual environments, we used IP multicast to send update messages directly between clients. The general idea is to map entity properties into multicast groups, and send update messages only to relevant groups. For instance, Macedonia partitions a virtual world</p>
--	--

	<p>into a 2D grid of hexagonal shaped cells each of which is represented by a separate multicast group. Entities localize their visual interactions by sending updates only to the multicast group representing the cell in which they reside, and they listen only to multicast groups representing cells within some radius.</p> <p>The multicast approach is similar to the RING client-server approach for wide-area networks. In both cases, intermediate machines may cull messages rather than propagating them to all participating workstations. However, using multicast, message culling is done by routers at the network layer, whereas, in RING, message culling is done by server machines at the application layer (see Figure 11). The advantages of the multicast approach are that: 1) fewer messages must be passed if clients are connected directly to a multicast-capable LAN (e.g., ethernet), and 2) latency is reduced due to faster message routing. The disadvantages are that: 1) delays associated with joining and leaving multicast groups make it impractical to use highly dynamic entity properties for multicast group mappings, 2) the number of unique multicast groups accessible to any one application may not be sufficient for complex virtual environments, and 3) multicast is not generally available across wide-area networks to many types of network computers (e.g., PCs with modems).</p> <p>The advantage of the RING client-server approach is that very dynamic and complex <i>message processing</i> may be performed by servers. In contrast to multicast routers, which can only cull messages based on a relatively small, static set of multicast groups, RING servers can cull messages using high-level geometric algorithms and knowledge regarding a multiplicity of highly dynamic entity attributes (e.g., location, orientation, velocity, etc.) and interaction types (e.g., visibility, sound, collision, etc.). Since RING servers can take advantage of knowledge regarding message semantics and the 3D geometry of the virtual environment directly, they can execute more effective and flexible culling algorithms than would be possible using only IP address and port mappings. Furthermore, unlike multicast routers, RING servers may process, augment, and alter messages in addition to culling them. For instance, RING servers already augment update messages with “Add” and “Remove” messages to inform clients that entities are entering or leaving their potentially visible areas.”</p> <p>RING at p. 90-91.</p>
--	---

CC-F

Claim Chart comparing Claims 1, 2, and 4-6 of U.S. Patent No. 5,822,523 to the disclosure in RING in view of Van Hook

Prior art cited in this chart:

- Thomas A. Funkhouser, “RING: A Client-Server System for Multi-User Virtual Environments,” Association of Computing Machinery, 1995 Symposium on Interactive 3D Graphics, Monterey CA. (“RING”)
- Daniel J. Van Hook, James O. Calvin, Michael K. Newton, and David A. Fusco, “An Approach to DIS Scalability,” 11th DIS Workshop, 26-30 Sept. 1994 (“Van Hook”).

Reasons to Combine:

RING discloses communicating messages over a network. RING at Figs. 5 and 7, pp. 88, 87 and 91. RING does not disclose aggregating payloads into a single aggregated message, but Van Hook discloses aggregating group messages into a single packet by bundling the packets. Van Hook at 2. Van Hook states that “[t]he dominant effect of bundling is to reduce packet rates. Additionally, bundling reduces bit rates because fewer packet headers are sent.” *Id.* Therefore, one of ordinary skill in the art would have looked to Van Hook to aggregate group messages in order to reduce bit rates and increase the network efficiency of RING.

Claims of the '523 Patent	Disclosure of RING and Van Hook
1. A method for providing group messages to a plurality of host computers connected over a unicast wide area communication network, comprising the steps of:	<p>“This paper describes the client-server design, implementation and experimental results for a system that supports real-time visual interaction between a large number of users in a shared 3D virtual environment. The key feature of the system is that server-based visibility algorithms compute potential visual interactions between entities representing users in order to reduce the number of messages required to maintain consistent state among many workstations distributed across a wide-area network. When an entity changes state, update messages are sent only to workstations with entities that can potentially perceive the change- i.e., ones to which the update is visible.”</p> <p>RING at Abstract.</p>

“In a multi-user visual simulation system, users run an interactive interface program on (usually distinct) workstations connected to each other via a network.”

RING at p. 85.

“A difficult challenge in multi-user visual simulation is maintaining consistent state among a large number of workstations distributed over a wide-area network.”

RING at p. 85.

“In order to support very large numbers of users (> 1000) interacting simultaneously in a distributed virtual environment it is necessary to develop a system design and communication protocol that does not require sending update messages to all participating hosts for every entity state change.”

RING at p. 86.

“This paper describes a system (called RING) that supports interaction between large numbers of users in virtual environments with dense occlusion (e.g., buildings, cities, etc.). RING takes advantage of the fact that state changes must be propagated only to hosts containing entities that can possibly perceive the change- i.e., the one that can see it. Object-space visibility algorithms are used to compute the region of influence for each state change, and then update messages are sent only to the small subset of workstations to which the update is relevant.”

RING at p. 86.

“We have experimented with a variety of topologies for connecting RING clients and servers. For practical reasons, we have focused mainly on arrangements in which clients communicate with a single server. However, depending on the capabilities of available workstations and networks, clients can send messages to server(s) via unicast or multicast.”

RING at p. 91.

“In our first experiments with multi-user virtual environments, we used IP multicast to send update messages directly between clients. The general idea is to map entity properties into multicast groups, and send update messages only to relevant groups. For instance, Macedonia partitions a virtual world into a 2D grid of hexagonal shaped cells each of which is represented by a separate multicast group. Entities

localize their visual interactions by sending updates only to the multicast group representing the cell in which they reside, and they listen only to multicast groups representing cells within some radius.

The multicast approach is similar to the RING client-server approach for wide-area networks. In both cases, intermediate machines may cull messages rather than propagating them to all participating workstations. However, using multicast, message culling is done by routers at the network layer, whereas, in RING, message culling is done by server machines at the application layer (see Figure 11). The advantages of the multicast approach are that: 1) fewer messages must be passed if clients are connected directly to a multicast-capable LAN (e.g., ethernet), and 2) latency is reduced due to faster message routing. The disadvantages are that: 1) delays associated with joining and leaving multicast groups make it impractical to use highly dynamic entity properties for multicast group mappings, 2) the number of unique multicast groups accessible to any one application may not be sufficient for complex virtual environments, and 3) multicast is not generally available across wide-area networks to many types of network computers (e.g., PCs with modems).

The advantage of the RING client-server approach is that very dynamic and complex *message processing* may be performed by servers. In contrast to multicast routers, which can only cull messages based on a relatively small, static set of multicast groups, RING servers can cull messages using high-level geometric algorithms and knowledge regarding a multiplicity of highly dynamic entity attributes (e.g., location, orientation, velocity, etc.) and interaction types (e.g., visibility, sound, collision, etc.). Since RING servers can take advantage of knowledge regarding message semantics and the 3D geometry of the virtual environment directly, they can execute more effective and flexible culling algorithms than would be possible using only IP address and port mappings. Furthermore, unlike multicast routers, RING servers may process, augment, and alter messages in addition to culling them. For instance, RING servers already augment update messages with “Add” and “Remove” messages to inform clients that entities are entering or leaving their potentially visible areas.”

RING at p. 90-91.

providing a group messaging server coupled to said network, said server communicating with said plurality of host computers using said unicast network and maintaining a list of message groups, each message group containing at least one host computer;

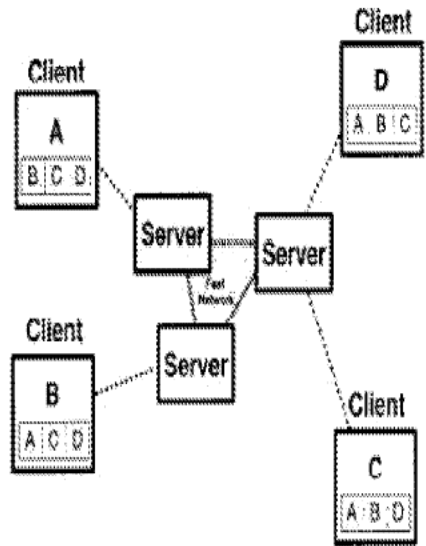


Figure 5: RING servers manage communication between clients, possibly culling, augmenting, or altering messages.

Figure 5 of RING at p. 87.

“We have experimented with a variety of topologies for connecting RING clients and servers. For practical reasons, we have focused mainly on arrangements in which clients communicate with a single server. However, depending on the capabilities of available workstations and networks, clients can send messages to server(s) via unicast or multicast.”

RING at p. 91.

“Server-based message culling is implemented using precomputed line-of-sight visibility information. Prior to the multi-user simulation, the shared virtual environment is partitioned into a spatial subdivision of *cells* whose boundaries are comprised of the static, axis-aligned polygons of the virtual environment [1, 15]. A visibility precomputation is performed in which the set of cells potentially visible to each cell is determined by tracing beams of possible sight-lines through transparent cell boundaries [15, 16] (see Figure 6). During the multi-user simulation, servers keep track of which cells contain which entities by

exchanging “periodic” update messages when entities cross cell boundaries. Real-time update messages are propagated only to servers and clients containing entities inside some cell visible to the one containing the updated entity. Since an entity’s visibility is conservatively over-estimated by the precomputed visibility of its containing cell, this algorithm allows servers to prove update messages quickly using cell visibility “look-ups” rather than more exact real-time entity visibility computations which would be too expensive on currently available workstations.”

RING at p. 87.

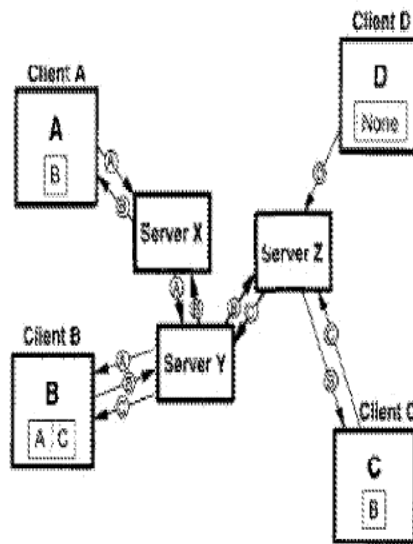
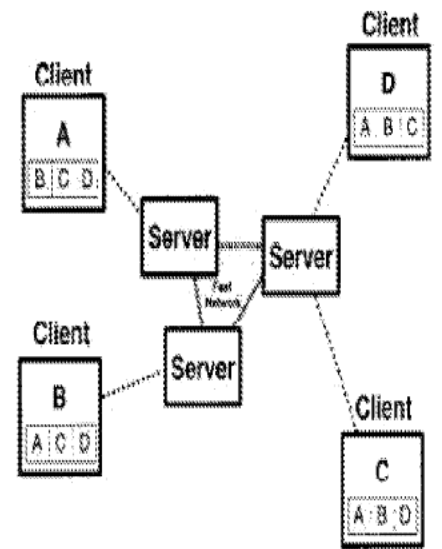


Figure 7: Flow of update messages (labeled arrows) for updates to entities A, B, C, and D arranged in a virtual environment as shown in Figure 4.

Figure 7 of RING at p. 88.

“Communication between clients is managed by *servers*. Clients do not send messages directly to other clients, but instead send them to servers which forward them to other client and server workstations participating in the same distributed simulation (see Figure 5). A key feature of this client-server design is that servers can process messages before propagating them to other workstations, culling, augmenting, or altering them. For instance, a server may determine that a particular update message is relevant only

	<p>to a small subset of clients and the propagate the message only to those clients or their servers.” RING at p. 87.</p>
<p>sending, by a plurality of host computers belonging to a first message group, messages to said server via said unicast network, said messages containing a payload portion and a portion for identifying said first message group;</p>	<p>“RING represents a virtual environment as a set of independent entities each of which has a geometric description and a behavior. Some entities are static (e.g., terrain, buildings, etc.), whereas others have dynamic behavior that can be either autonomous (e.g., robots) or controlled by a user via input devices (e.g., vehicles). Distributed simulation occurs when multiple entities interact in a shared virtual environment by sending messages to one another to announce updates to their own geometry or behavior modifications to the shared environment, or impact on other entities.” RING at p. 87.</p> <p>“Communication between clients is managed by <i>servers</i>. Clients do not send messages directly to other clients, but instead send them to servers which forward them to other client and server workstations participating in the same distributed simulation (see Figure 5).” RING at p. 87.</p> <p>“We have experimented with a variety of topologies for connecting RING clients and servers. For practical reasons, we have focused mainly on arrangements in which <u>clients communicate with a single server</u>. However, depending on the capabilities of available workstations and networks, clients can send messages to server(s) via unicast or multicast.” RING at p. 91.</p> <p>“Update messages containing 40 bytes (message-type[4], entity-ID[4], target-position[12], target-orientation[12], positional velocity[4], and rotational-velocity[4] were generated for each entity once every 2.3 seconds on average with this ‘random’ navigational behavior.” RING at p. 89.</p>

	 <p data-bbox="714 945 1185 1029">Figure 5: RING servers manage communication between clients, possibly calling, augmenting, or altering messages.</p> <p data-bbox="812 1039 1088 1081">Figure 5 of RING at p. 87.</p>
<p data-bbox="211 1092 454 1428">aggregating, by said server in a time interval determined in accordance with a predefined criterion, said payload portions of said messages to create an aggregated payload;</p>	<p data-bbox="470 1092 1421 1386">“Bundling. Network components such as switches, routers, and encryption devices as well as simulation host computers have limitations in the rate at which they may process packets. Rather than transmitting each DIS PDU as an individual packet, multiple PDUs may be bundled together into larger packets before transmission. Bundled packets are transmitted when either of two conditions are satisfied: when a maximum size has been reached (the packet under construction is full of PDUs); or when a maximum time has passed without another PDU arriving. The dominant effect of bundling is to reduce packet rates. Additionally, bundling reduces bit rates because fewer packet headers are sent.”</p> <p data-bbox="470 1396 649 1438">Van Hook at p. 2.</p> <p data-bbox="470 1470 617 1522">“4.6 Bundling</p> <p data-bbox="470 1564 1421 1690">The AG collects AGGP PDUs and bundles them into larger packets for transmission over the WAN. The purpose of the bundling algorithm is to reduce the number of packets that are transmitted. The bundling algorithm has two parameters, a maximum bundle size and a maximum delay time. PDUs are added to a</p>

bundle until either the maximum size is reached or the first PDU in the bundle has been delayed by the maximum delay time. At this point, the bundle is transmitted.”

Van Hook at p. 7.

“Furthermore unlike multicast routers, RING servers may process, augment, and alter messages in addition to culling them. For instance, RING servers already augment update messages with “Add” and “Remove” messages to inform clients that entities are entering or leaving their potentially visible sets.”

RING at p. 91.

“RING servers allow each client workstation to maintain surrogates for only the subset of remote entities visible to at least one entity local to the client. . . . To support this feature, servers send their client an “Add” message each time a remote entity enters a cell potentially visible to one of the client’s local entities for the first time. A “Remove” message is sent when the server determines that an entity has left the client’s visible region. As entities move through the environment, servers augment update messages with “Add” and “Remove” messages notifying clients that remote entities have become relevant or irrelevant to the client’s local entities.”

RING at p. 88.

“Finally, time critical computing algorithms can be used to determine an ‘optimal’ set of messages to send to each client based on network connection bandwidths, workstation processing capabilities, and many other real-time performance factors (i.e., in a manner similar to that used in [8]).”

RING at p. 91.

“During the multi-user simulation, servers keep track of which cells contain which entities by exchanging ‘periodic’ update messages when entities cross cell boundaries. Real-time update messages are propagated only to servers and clients containing entities inside some cell visible to the one containing the updated entity. Since an entity’s visibility is conservatively over-estimated by the precomputed visibility of its containing cell, this algorithm allows servers to process update messages quickly using cell visibility ‘look-ups’ rather than more exact real-time entity visibility computations which would be too expensive on currently available workstations.”

RING at p. 87.

	<p>“Rather than sending messages directly between clients, RING routes each on through at least one server, and possibly two. Computations are performed in the servers before messages are propagated further adding to latency.”</p> <p>RING at p. 88.</p>
<p>forming an aggregated message using said aggregated payload; and</p>	<p>“Bundling. Network components such as switches, routers, and encryption devices as well as simulation host computers have limitations in the rate at which they may process packets. Rather than transmitting each DIS PDU as an individual packet, multiple PDUs may be bundled together into larger packets before transmission. Bundled packets are transmitted when either of two conditions are satisfied: when a maximum size has been reached (the packet under construction is full of PDUs); or when a maximum time has passed without another PDU arriving. The dominant effect of bundling is to reduce packet rates. Additionally, bundling reduces bit rates because fewer packet headers are sent.”</p> <p>Van Hook at p. 2.</p> <p>“Furthermore unlike multicast routers, RING servers may process, augment, and alter messages in addition to culling them. For instance, RING servers already augment update messages with “Add” and “Remove” messages to inform clients that entities are entering or leaving their potentially visible sets.”</p> <p>RING at p. 91.</p> <p>“RING servers allow each client workstation to maintain surrogates for only the subset of remote entities visible to at least one entity local to the client. . . . To support this feature, servers send their client an “Add” message each time a remote entity enters a cell potentially visible to one of the client’s local entities for the first time. A “Remove” message is sent when the server determines that an entity has left the client’s visible region. As entities move through the environment, server augment update messages with “Add” and “Remove” messages notifying clients that remote entities have become relevant or irrelevant to the client’s local entities.”</p> <p>RING at p. 88.</p>
<p>transmitting, by said server via said unicast network, said aggregated message to a recipient</p>	<p>“Communication between clients is managed by <i>servers</i>. Clients do not send messages directly to other clients, but instead send them to servers which forward them to other client and server workstations participating in the same distributed simulation (see Figure 5).”</p> <p>RING at p. 87.</p>

<p>host computer belonging to said first message group.</p>	<p>“We have experimented with a variety of topologies for connecting RING clients and servers. For practical reasons, we have focused mainly on arrangements in which clients communicate with a single server. However, depending on the capabilities of available workstations and networks, clients can send messages to server(s) via unicast or multicast.” RING at p. 91.</p> <p>“RING servers allow each client workstation to maintain surrogates for only the subset of remote entities visible to at least one entity local to the client. . . . To support this feature, servers send their client an “Add” message each time a remote entity enters a cell potentially visible to one of the client’s local entities for the first time. A “Remove” message is sent when the server determines that an entity has left the client’s visible region. As entities move through the environment, server augment update messages with “Add” and “Remove” messages notifying clients that remote entities have become relevant or irrelevant to the client’s local entities.” RING at p. 88.</p> <p>“Bundling. Network components such as switches, routers, and encryption devices as well as simulation host computers have limitations in the rate at which they may process packets. Rather than transmitting each DIS PDU as an individual packet, multiple PDUs may be bundled together into larger packets before transmission. Bundled packets are transmitted when either of two conditions are satisfied: when a maximum size has been reached (the packet under construction is full of PDUs); or when a maximum time has passed without another PDU arriving. The dominant effect of bundling is to reduce packet rates. Additionally, bundling reduces bit rates because fewer packet headers are sent.” Van Hook at p. 2.</p> <p>“4.6 Bundling</p> <p>The AG collects AGGP PDUs and bundles them into larger packets for transmission over the WAN. The purpose of the bundling algorithm is to reduce the number of packets that are transmitted. The bundling algorithm has two parameters, a maximum bundle size and a maximum delay time. PDUs are added to a bundle until either the maximum size is reached or the first PDU in the bundle has been delayed by the</p>
---	--

maximum delay time. At this point, the bundle is transmitted.”

Van Hook at p. 7.

“**Exercise scale.** The large number of entities involved in STOW-E will produce offered loads of as much as four megabits and perhaps up to 2,000 packets per second. Such traffic levels will severely tax all simulation computers even if unlimited communications resources were available.”

Van Hook at p. 1.

“Explicit representations of command, control, and communication are required to permit command forces to transmit orders to and receive reports from a new generation of more intelligent semi-automated forces. These new elements and phenomena require new protocols and generate new classes of traffic that must be carried on the connecting networks.”

Van Hook at p. 1.

“A component of ARPA’s approach to scalability for STOW-E is to implement cooperating and complementary instances of a number of the information flow management techniques in an Application Gateway (AG) situated at the LAN/WAN boundary of each participating network site (figure 1). The AG may be thought of as a collection of information flow management agents [4] that perform services on behalf of their clients, the simulation applications. The purpose of these agents is to compensate for and efficiently use the available communication and processing resources. Each AG processes PDUs received from its attached LAN and sends representation of local exercise state and events to other AGs over the WAN. Similarly, each AG receives representations of remote state and events from other AGs over the WAN and sends PDUs onto its attached LAN. Communication between AGs is via an Application Gateway to Gateway Protocol (AGGP). AGGP supports communication of control information related to the information flow management techniques as well as representations of exercise state and events.”

Van Hook at p. 4.

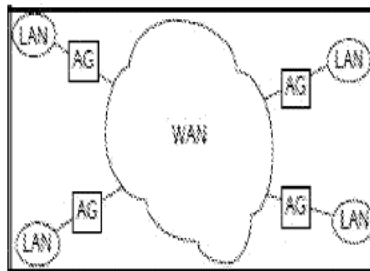


Figure 1: Application Gateway connections within the network

Figure 1 of Van Hook at p. 4.

“The algorithm operates as follows. The terrain is divided into a grid of square cells by each AG. A square grid is used because it makes calculations simple and permits regions of the terrain to be specified as a list of cells. Each AG determines the set of cells from which it needs to receive full accuracy data. This set consists of those cells that overlay the circular regions of interest of the entities at the AG’s site LAN. Figure 5 illustrates this idea by showing three entities and their circular regions of interest. For determining the full accuracy region, the AGs use regions of interest that are based upon the viewing ranges of the entities on the site LAN. The set of cells for which full accuracy data is needed is outlined in the figure. All AGs transmit their cell sets to each other. The full accuracy region for any AG consists of the union of the sets of cells received from all other AGs.”

Van Hook at p. 6.

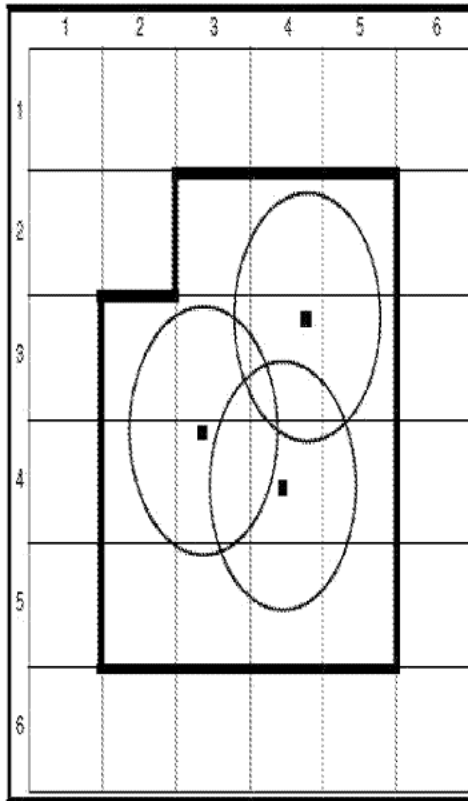


Figure 5: Cells for which full accuracy is required

Figure 5 of Van Hook at p. 6.

2. The method of claim 1 wherein said time interval is a fixed period of time.

“During the multi-user simulation, servers keep track of which cells contain which entities by exchanging “periodic” update messages when entities cross cell boundaries.”

RING at p. 87.

“This paper describes the client-server design, implementation and experimental results for a system that supports real-time visual interaction between a large number of users in a shared 3D virtual environment. The key feature of the system is that server-based visibility algorithms compute potential visual interactions between entities representing users in order to reduce the number of messages required to maintain consistent state among many workstations distributed across a wide-area network. When an

entity changes state, update messages are sent only to workstations with entities that can potentially perceive the change- i.e., ones to which the update is visible.”

RING at Abstract.

“In a multi-user visual simulation system, users run an interactive interface program on (usually distinct) workstations connected to each other via a network.”

RING at p. 85.

“A difficult challenge in multi-user visual simulation is maintaining consistent state among a large number of workstations distributed over a wide-area network.”

RING at p. 85.

“In order to support very large numbers of users (> 1000) interacting simultaneously in a distributed virtual environment it is necessary to develop a system design and communication protocol that does not require sending update messages to all participating hosts for every entity state change.”

RING at p. 86.

“This paper describes a system (called RING) that supports interaction between large numbers of users in virtual environments with dense occlusion (e.g., buildings, cities, etc.). RING takes advantage of the fact that state changes must be propagated only to hosts containing entities that can possibly perceive the change- i.e., the one that can see it. Object-space visibility algorithms are used to compute the region of influence for each state change, and then update messages are sent only to the small subset of workstations to which the update is relevant.”

RING at p. 86.

“We have experimented with a variety of topologies for connecting RING clients and servers. For practical reasons, we have focused mainly on arrangements in which clients communicate with a single server. However, depending on the capabilities of available workstations and networks, clients can send messages to server(s) via unicast or multicast.”

RING at p. 91.

“In our first experiments with multi-user virtual environments, we used IP multicast to send update messages directly between clients. The general idea is to map entity properties into multicast groups, and send update messages only to relevant groups. For instance, Macedonia partitions a virtual world into a 2D grid of hexagonal shaped cells each of which is represented by a separate multicast group. Entities localize their visual interactions by sending updates only to the multicast group representing the cell in which they reside, and they listen only to multicast groups representing cells within some radius.

The multicast approach is similar to the RING client-server approach for wide-area networks. In both cases, intermediate machines may cull messages rather than propagating them to all participating workstations. However, using multicast, message culling is done by routers at the network layer, whereas, in RING, message culling is done by server machines at the application layer (see Figure 11). The advantages of the multicast approach are that: 1) fewer messages must be passed if clients are connected directly to a multicast-capable LAN (e.g., ethernet), and 2) latency is reduced due to faster message routing. The disadvantages are that: 1) delays associated with joining and leaving multicast groups make it impractical to use highly dynamic entity properties for multicast group mappings, 2) the number of unique multicast groups accessible to any one application may not be sufficient for complex virtual environments, and 3) multicast is not generally available across wide-area networks to many types of networks computers (e.g., PCs with modems).

The advantage of the RING client-server approach is that very dynamic and complex *message processing* may be performed by servers. In contrast to multicast routers, which can only cull messages based on a relatively small, static set of multicast groups, RING servers can cull messages using high-level geometric algorithms and knowledge regarding a multiplicity of highly dynamic entity attributes (e.g., location, orientation, velocity, etc.) and interaction types (e.g., visibility, sound, collision, etc.). Since RING servers can take advantage of knowledge regarding message semantics and the 3D geometry of the virtual environment directly, they can execute more effective and flexible culling algorithms than would be possible using only IP address and port mappings. Furthermore, unlike multicast routers, RING servers may process, augment, and alter messages in addition to culling them. For instance, RING servers already augment update messages with “Add” and “Remove” messages to inform clients that entities are entering or leaving their potentially visible areas.”

RING at p. 90-91.

<p>4. The method of claim 1 further comprising the step of creating, by one of said plurality of host computers, said first message group by sending a first control message to said server via said unicast network.</p>	<p>“In our first experiments with multi-user virtual environments, we used IP multicast to send update messages directly between clients. The general idea is to map entity properties into multicast groups, and send update messages only to relevant groups. For instance, Macedonia partitions a virtual world into a 2D grid of hexagonal shaped cells each of which is represented by a separate multicast group. Entities localize their visual interactions by sending updates only to the multicast group representing the cell in which they reside, and they listen only to multicast groups representing cells within some radius.</p> <p>The multicast approach is similar to the RING client-server approach for wide-area networks. In both cases, intermediate machines may cull messages rather than propagating them to all participating workstations. However, using multicast, message culling is done by routers at the network layer, whereas, in RING, message culling is done by server machines at the application layer (see Figure 11). The advantages of the multicast approach are that: 1) fewer messages must be passed if clients are connected directly to a multicast-capable LAN (e.g., ethernet), and 2) latency is reduced due to faster message routing. The disadvantages are that: 1) delays associated with joining and leaving multicast groups make it impractical to use highly dynamic entity properties for multicast group mappings, 2) the number of unique multicast groups accessible to any one application may not be sufficient for complex virtual environments, and 3) multicast is not generally available across wide-area networks to many types of networks computers (e.g., PCs with modems).</p> <p>The advantage of the RING client-server approach is that very dynamic and complex <i>message processing</i> may be performed by servers. In contrast to multicast routers, which can only cull messages based on a relatively small, static set of multicast groups, RING servers can cull messages using high-level geometric algorithms and knowledge regarding a multiplicity of highly dynamic entity attributes (e.g., location, orientation, velocity, etc.) and interaction types (e.g., visibility, sound, collision, etc.). Since RING servers can take advantage of knowledge regarding message semantics and the 3D geometry of the virtual environment directly, they can execute more effective and flexible culling algorithms than would be possible using only IP address and port mappings. Furthermore, unlike multicast routers, RING servers may process, augment, and alter messages in addition to culling them. For instance, RING servers already augment update messages with “Add” and “Remove” messages to inform clients that entities are entering or leaving their potentially visible areas.”</p> <p>RING at p. 90-91.</p>
---	--

<p>5. The method of claim 4 further comprising the step of joining, by some of said plurality of host computers, said first message group by sending control messages via said unicast network to said server specifying said first message group.</p>	<p>“Update messages containing 40 bytes (message-type[4], entity-ID[4], target-position[12], target-orientation[12], positional velocity[4], and rotational-velocity[4] were generated for each entity once every 2.3 seconds on average with this ‘random’ navigational behavior.”</p> <p>RING at p. 89.</p> <p>“In our first experiments with multi-user virtual environments, we used IP multicast to send update messages directly between clients. The general idea is to map entity properties into multicast groups, and send update messages only to relevant groups. For instance, Macedonia partitions a virtual world into a 2D grid of hexagonal shaped cells each of which is represented by a separate multicast group. Entities localize their visual interactions by sending updates only to the multicast group representing the cell in which they reside, and they listen only to multicast groups representing cells within some radius.</p> <p>The multicast approach is similar to the RING client-server approach for wide-area networks. In both cases, intermediate machines may cull messages rather than propagating them to all participating workstations. However, using multicast, message culling is done by routers at the network layer, whereas, in RING, message culling is done by server machines at the application layer (see Figure 11). The advantages of the multicast approach are that: 1) fewer messages must be passed if clients are connected directly to a multicast-capable LAN (e.g., ethernet), and 2) latency is reduced due to faster message routing. The disadvantages are that: 1) delays associated with joining and leaving multicast groups make it impractical to use highly dynamic entity properties for multicast group mappings, 2) the number of unique multicast groups accessible to any one application may not be sufficient for complex virtual environments, and 3) multicast is not generally available across wide-area networks to many types of network computers (e.g., PCs with modems).</p> <p>The advantage of the RING client-server approach is that very dynamic and complex <i>message processing</i> may be performed by servers. In contrast to multicast routers, which can only cull messages based on a relatively small, static set of multicast groups, RING servers can cull messages using high-level geometric algorithms and knowledge regarding a multiplicity of highly dynamic entity attributes (e.g., location, orientation, velocity, etc.) and interaction types (e.g., visibility, sound, collision, etc.). Since RING servers can take advantage of knowledge regarding message semantics and the 3D geometry of the</p>
--	--

	<p>virtual environment directly, they can execute more effective and flexible culling algorithms than would be possible using only IP address and port mappings. Furthermore, unlike multicast routers, RING servers may process, augment, and alter messages in addition to culling them. For instance, RING servers already augment update messages with “Add” and “Remove” messages to inform clients that entities are entering or leaving their potentially visible areas.”</p> <p>RING at p. 90-91.</p>
<p>6. The method of claim 1 wherein said network is Internet and said server communicates with said plurality of host computers using a session layer protocol.</p>	<p>“The system runs on Silicon Graphics workstations and uses UDP/IP datagrams for message passing.”</p> <p>RING at p. 89.</p> <p>“A difficult challenge in multi-user visual simulation is maintaining consisted state among a large number of workstations distributed over a wide-area network.”</p> <p>RING at p. 85.</p> <div data-bbox="743 982 1156 1570" data-label="Diagram"> </div> <p>Figure 11: RING servers process messages in the application layer using 3D model and semantic information. Multicast routers use only IP addressing in the network layer.</p>

Figure 11 of RING at p. 91.

“However, using multicast, message culling is done by routers at the network layer, whereas, in RING, message culling is done by server machines at the application layer (see Figure 11).”

RING at p. 90.

“This paper describes the client-server design, implementation and experimental results for a system that supports real-time visual interaction between a large number of users in a shared 3D virtual environment. The key feature of the system is that server-based visibility algorithms compute potential visual interactions between entities representing users in order to reduce the number of messages required to maintain consistent state among many workstations distributed across a wide-area network. When an entity changes state, update messages are sent only to workstations with entities that can potentially perceive the change- i.e., ones to which the update is visible.”

RING at Abstract.

“In a multi-user visual simulation system, users run an interactive interface program on (usually distinct) workstations connected to each other via a network.”

RING at p. 85.

“A difficult challenge in multi-user visual simulation is maintaining consistent state among a large number of workstations distributed over a wide-area network.”

RING at p. 85.

“In order to support very large numbers of users (> 1000) interacting simultaneously in a distributed virtual environment it is necessary to develop a system design and communication protocol that does not require sending update messages to all participating hosts for every entity state change.”

RING at p. 86.

“This paper describes a system (called RING) that supports interaction between large numbers of users in virtual environments with dense occlusion (e.g., buildings, cities, etc.). RING takes advantage of the fact that state changes must be propagated only to hosts containing entities that can possibly perceive the

change- i.e., the one that can see it. Object-space visibility algorithms are used to compute the region of influence for each state change, and then update messages are sent only to the small subset of workstations to which the update is relevant.”

RING at p. 86.

“We have experimented with a variety of topologies for connecting RING clients and servers. For practical reasons, we have focused mainly on arrangements in which clients communicate with a single server. However, depending on the capabilities of available workstations and networks, clients can send messages to server(s) via unicast or multicast.”

RING at p. 91.

“In our first experiments with multi-user virtual environments, we used IP multicast to send update messages directly between clients. The general idea is to map entity properties into multicast groups, and send update messages only to relevant groups. For instance, Macedonia partitions a virtual world into a 2D grid of hexagonal shaped cells each of which is represented by a separate multicast group. Entities localize their visual interactions by sending updates only to the multicast group representing the cell in which they reside, and they listen only to multicast groups representing cells within some radius.

The multicast approach is similar to the RING client-server approach for wide-area networks. In both cases, intermediate machines may cull messages rather than propagating them to all participating workstations. However, using multicast, message culling is done by routers at the network layer, whereas, in RING, message culling is done by server machines at the application layer (see Figure 11). The advantages of the multicast approach are that: 1) fewer messages must be passed if clients are connected directly to a multicast-capable LAN (e.g., ethernet), and 2) latency is reduced due to faster message routing. The disadvantages are that: 1) delays associated with joining and leaving multicast groups make it impractical to use highly dynamic entity properties for multicast group mappings, 2) the number of unique multicast groups accessible to any one application may not be sufficient for complex virtual environments, and 3) multicast is not generally available across wide-area networks to many types of network computers (e.g., PCs with modems).

The advantage of the RING client-server approach is that very dynamic and complex *message processing*

may be performed by servers. In contrast to multicast routers, which can only cull messages based on a relatively small, static set of multicast groups, RING servers can cull messages using high-level geometric algorithms and knowledge regarding a multiplicity of highly dynamic entity attributes (e.g., location, orientation, velocity, etc.) and interaction types (e.g., visibility, sound, collision, etc.). Since RING servers can take advantage of knowledge regarding message semantics and the 3D geometry of the virtual environment directly, they can execute more effective and flexible culling algorithms than would be possible using only IP address and port mappings. Furthermore, unlike multicast routers, RING servers may process, augment, and alter messages in addition to culling them. For instance, RING servers already augment update messages with “Add” and “Remove” messages to inform clients that entities are entering or leaving their potentially visible areas.”

RING at p. 90-91.

OTH-C

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re <i>Ex Parte</i> Reexamination of:)
)
Patent No. 6,264,560) Control Number: Not Yet Assigned
)
Inventors: S. Goldberg) Group Art Unit: Not Yet Assigned
J. Van Antwerp)
) Examiner: Not Yet Assigned
Issue Date: July 24, 2001)
Application No. 90/140,979) Box: <i>Ex Parte</i> Reexam
Filed: August 27, 1998)
)
For: Method and System for Playing Games)
on a Network)

Mail Stop *Ex Parte* Reexam
 Commissioner for Patents
 P.O. Box 1450
 Alexandria, VA 22313-1450

DECLARATION OF DAVID AHN

I, David Ahn, declare:

1. I hereby certify that I am over 18 years of age and am competent to execute this declaration. If called as a witness, I could and would competently testify to the following facts, of which I have personal knowledge.

2. I understand that this declaration is being submitted in conjunction with a request for reexamination of claim 92 of U.S. Patent No. 6,264,560. I further understand that the source code for the multi-user online game Netrek is being submitted in support of that reexamination request, which code was downloaded from the Netrek Software Archive at <http://ftp.netrek.org/>. Specifically, I understand that the source code archive files named "BRMH-1.7.tar.gz" and

Declaration of David Ahn

“Server2.5pl4.tar.gz” have been submitted. This declaration addresses my personal knowledge regarding the public accessibility of those files and the source code they contain.

3. My educational and professional background are in the field of computer science. In 1995 I received a Bachelor of Science degree in Computer Science from Wake Forest University. Since then, I have worked in the area of information technology (IT) and software engineering. After graduation I became employed full-time at the Virtual Endoscopy Center at Wake Forest University School of Medicine, where my work focused on researching and developing software algorithms and techniques for medical imaging and visualization. Between 1999 and 2001 I was instrumental in forming a medical software company called PointDx, Inc. In 2001 I left Wake Forest to join PointDx full-time. My responsibilities at PointDx included overseeing all aspects of the technology side of the business, including technical direction, product engineering and development, and IT infrastructure. I was primarily responsible for the design and development of software products, and I continued working in this capacity through two acquisitions of my employer, the first by IDX Systems Corporation and the second by GE Healthcare IITS. In 2006 I left GE and subsequently joined GreatWall Systems, Inc., an early-stage startup company offering IT network security products and services. I am responsible for the technology side of the company, including overseeing all hardware, software, and systems development of the company’s products.

4. Including my time as an undergraduate student and an amateur programmer, I have over 16 years of experience in the field of computer science, programming, and software engineering.

5. I am very familiar with Netrek. I first learned about Netrek in or around 1992 or 1993 as a computer science student at Wake Forest University. At that time, Netrek was quite

Declaration of David Ahn

popular on college campuses all over the world.¹ Netrek software generally falls into two categories: Netrek client software and Netrek server software. In order to play Netrek, a player uses Netrek client software running on a local computer to connect over the network, such as the Internet, to Netrek server software running on a remote host computer. There are many versions and variations of both Netrek client and server software. For example, the archive files “BRMH-1.7.tar.gz” and “Server2.5pl4.tar.gz,” addressed below, contain the source code to specific versions of Netrek client and server software.

6. In approximately 1994, I began to play Netrek extensively and, by the end of that year, had become involved with Netrek software and with the Netrek community in general. Over the subsequent years, I expanded my involvement with Netrek through various activities such as joining and playing in various Netrek leagues, organizing Netrek leagues and tournaments, developing and maintaining Netrek software, and maintaining and supporting public Netrek resources on the Internet such as the International Netrek League (INL) statistics home page, the Netrek Home Page, the Netrek Software Archive, and others. I also participated actively in the Usenet newsgroup rec.games.netrek, so much so that a recent search on “Dave Ahn” in that newsgroup covering 1994 to the present resulted in over 1600 results.² I consider myself deeply involved and very well-known in the Netrek community, even though my involvement has lessened greatly in the last five years.

¹ Posting of Tom Holub to rec.games.netrek, Subject: Netrek Server List, <http://groups.google.com/group/rec.games.netrek/msg/8dbc01d4abee5ace> (Dec. 21, 1993), a true and correct copy of which is contained in attached Exhibit A. (Note: in citations to Google Groups postings, this declaration specifies the “Local” date included in the header information.)

² Attached hereto as Exhibit B is a true and correct copy of the first page of a search of rec.games.netrek for “Dave Ahn” performed on Dec. 10, 2007 (<http://groups.google.com/group/rec.games.netrek/search?q=dave+ahn&start=0&scoring=d&>).

Declaration of David Ahn

7. The software source code for Netrek is publicly accessible and, to the best of my knowledge, has been ever since I became aware of Netrek. I began building (compiling and installing) my own Netrek server in 1994 using source code for the Netrek “Vanilla” server which I had obtained from a publicly accessible file transfer protocol (FTP) site at <ftp.ecst.csuchico.edu>.³ Thereafter, I spent a great deal of time downloading, experimenting with, and modifying the source code to various Netrek software. Over the years, I became quite familiar with Netrek software and its source code, including how they worked and where various versions of the associated files could be found.

8. The Usenet newsgroup <rec.games.netrek>⁴ was a central communications tool for the Netrek community in the early 1990s and still exists today. That newsgroup not only offered an arena for general discussions of Netrek-related topics but also served as a medium for publishing the locations of various FTP sites from which Netrek software and source code could be freely downloaded. Such information was periodically posted in the form of messages containing lists of answers to frequently asked questions (FAQ) about Netrek, lists of known public Netrek servers, and lists of known FTP servers where Netrek software and source code were published. One instance of the Netrek FAQ List appears in a posting dated July 21, 1994,

³ Posting of Dave Ahn to <rec.games.netrek>, Subject: Help getting res-rsa working with server..., http://groups.google.com/group/rec.games.netrek/browse_frm/thread/4da4c5af59745a61/fde76eab98a25b8a (Nov. 14, 1994), a true and correct copy of which is contained in attached Exhibit C.

⁴ Google Groups provides an archive of Usenet newsgroups that dates back to 1981. How far back does Google’s Usenet archive go?, <http://groups.google.com/support/bin/answer.py?answer=46439&topic=9246>, a true and correct copy of which is contained in attached Exhibit D, at D-1. Google Groups includes an archive of <rec.games.netrek> that dates back to 1992. <rec.games.netrek>, <http://groups.google.com/group/rec.games.netrek/about>, a true and correct copy of which is contained in attached Exhibit D, at D-2/3.

Declaration of David Ahn

titled "rec.games.netrek FAQ List."⁵ The Netrek FAQ List posting points readers to a Netrek FTP List for locations where Netrek server source code could be downloaded.⁶

9. A subsequent posting, also dated July 21, 1994, is titled "Netrek FTP List." That posting lists various FTP servers from which both server source code and client source code could be downloaded. It also identifies locations from which the latest versions of the Netrek FAQ List, Netrek Server List, and Netrek FTP List could be downloaded.⁷ Among other things, it features a list of "blessed clients" that includes, for example, a client version called BRM-Hadley 1.7. The description in that posting explains that the source code for BRM-Hadley 1.7 was accessible at cad.ics.uci.edu in the directory /pub/netrek.⁸ Later in the post is a section titled "Subject: SERVER SOURCE" that contains "a listing of all known netrek server sources."⁹ The second server on the list is named "New Vanilla Server 2.2+," and the post indicates that it was

⁵ Posting of Tom Holub to rec.games.netrek, Subject: rec.games.netrek FAQ List, <http://groups.google.com/group/rec.games.netrek/msg/9bbd5514020d51fa> (Jul. 21, 1994), a true and correct copy of which is contained in attached Exhibit E; thread view of same available at http://groups.google.com/group/rec.games.netrek/browse_frm/thread/35a84ea78ce38bdb/9bbd5514020d51fa (referencing FAQ, Server List, and FTP List), a true and correct copy of which is contained in attached Exhibit F.

⁶ Exhibit E, *supra* note 5 (Holub, FAQ List), at E-4 ("Read the Netrek FTP list to find out where you can get the server source.").

⁷ Posting of Tom Holub to rec.games.netrek, Subject: Netrek FTP list, <http://groups.google.com/group/rec.games.netrek/msg/ac03262b6ac8c4c1> (Jul. 21, 1994), a true and correct copy of which is contained in attached Exhibit G. The FTP List was updated from time to time. For example, a series of postings dated September through October 1994 mention the FTP List and identify it as a source for Netrek software. Posting of Tatsuya Murase to rec.games.netrek, Subject: Re: Windows Client, http://groups.google.com/group/rec.games.netrek/browse_frm/thread/d6ecc5c095bf8a38 (Sep. 30, 1994), a true and correct copy of which is contained in attached Exhibit H; Posting of Tatsuya Murase to rec.games.netrek, Subject: Re: Windows Client, <http://groups.google.com/group/rec.games.netrek/msg/20da2a42b64333a8> (Sep. 30, 1994) ("Reading the FAQ/FTPlist, you can easily download a client for your computer set it up, and run in under 45 minutes, if even that."), a true and correct copy of which is contained in attached Exhibit I.

⁸ Exhibit G, *supra* note 7 (Holub, Netrek FTP list), at G-3.

⁹ *Id.*, at G-6.

Declaration of David Ahn

maintained by Nick Trown at ftp.ecst.csuchico.edu in the directory /pub/netrek/src.¹⁰ In a later post to rec.games.netrek dated August 9, 1994, Nick Trown announced that a new version (2.5pl4) of the Vanilla Server had been posted to ftp.ecst.csuchico.edu.¹¹ My understanding, which is consistent with these postings, is that anyone involved in the Netrek community and/or reading rec.games.netrek could easily and freely have accessed these copies of Netrek source code at the indicated locations, as I myself did on several occasions.

10. Any number of other messages on rec.games.netrek confirm that the Netrek community was widely aware of how to locate and download Netrek software and source code. For example, an October 15, 1993 posting by Tedd Hadley announced that the BRM-Hadley (BRMH) 1.7 client source code was accessible at cad.ics.uci.edu:/pub/netrek/.¹² An August 17, 1994 posting by James Ivey describes having obtained a copy of the Vanilla 2.5pl4 server from ftp.ecst.csuchico.edu under /pub/netrek/src/Server2.5pl4.tar.gz.¹³ In that posting, Mr. Ivey declared, “[i]t’s really not hard to grab some code and take a look.”¹⁴ On November 19, 1994, a posting by Vanilla Server maintainer Nick Trown responded to a message from a user who had

¹⁰ *Id.*, at G-6/7.

¹¹ Posting of Nick Trown to rec.games.netrek, Subject: New Server Release, <http://groups.google.com/group/rec.games.netrek/msg/d7fb4451975e6fb2> (Aug. 9, 1994), a true and correct copy of which is contained in attached Exhibit J.

¹² Posting of Tedd Hadley to rec.games.netrek, Subject: BRMH-1.7 available, <http://groups.google.com/group/rec.games.netrek/msg/00b0aa5dfdb1ba99> (Oct. 15, 1993), a true and correct copy of which is contained in attached Exhibit K.

¹³ The .tar.gz file extension signifies the multiple source code files that defined the Vanilla server had been combined into a single compressed archive file. (A “tar” file is an archive file that collates a collection of files into one larger file for distribution or archiving. A “tar.gz” file is a “tar” file that has been compressed to reduce storage usage. Such formats were and still are commonly used for creating software source code packages that are easily distributed and downloaded over the Internet.)

¹⁴ Posting of James Ivey to rec.games.netrek, Subject: Re: AGRI poppage (was Re: Bombing a planet -- is it an art?), <http://groups.google.com/group/rec.games.netrek/msg/df66eac4e839bc59> (Aug. 17, 1994), a true and correct copy of which is contained in attached Exhibit L.

Declaration of David Ahn

downloaded version 2.5pl4 of the Vanilla Server source code and was requesting assistance with compiling and running it.¹⁵

11. During the 1995-98 time frame, I became aware that a number of web sites and FTP sites that made Netrek software and source code freely available were disappearing or shutting down. Accordingly, I began to acquire a private collection of Netrek software, software source code, and other files that I had downloaded from the publicly accessible sites that still existed, in order to preserve those copies and/or use them for my own purposes. I eventually published my collection as the "Netrek FTP Archive" on an FTP site at <ftp://ftp.netrek.org/pub/netrek/>. I announced that FTP site in a posting to <rec.games.netrek> on October 23, 1998. That posting stated my belief that my FTP site contained "almost all known Netrek software including mirrors of major Netrek FTP sites."¹⁶ Since then, I have continued to maintain that site, which currently is called the "Netrek Software Archive" and is available at <http://ftp.netrek.org/>.

12. Among the files currently available on the Netrek Software Archive are those in "mirror" directories in the </pub/netrek/mirrors/> directory. These mirrored files are complete and verbatim copies of entire public FTP sites that were downloaded to the Netrek Software Archive using a mirroring script that preserved the directory hierarchy and file time stamps of the data copied from original FTP sites. The <ftp.csua.berkeley.ued.old> and <ftp.solace.mh.se> directories are mirror copies of the public FTP sites <ftp.csua.berkeley.edu> and <ftp.solace.mh.se>, respectively,

¹⁵ Posting of Nick Trown to <rec.games.netrek>, Subject: Re: Netrek server help !, http://groups.google.com/group/rec.games.netrek/browse_frm/thread/e728557051dc0c13/4f1af10b05d68ac8 (Nov. 18, 1994), a true and correct copy of which is contained in attached Exhibit M.

¹⁶ Posting of Dave Ahn to <rec.games.netrek>, Subject: www.netrek.org - no longer the game?, <http://groups.google.com/group/rec.games.netrek/msg/ee9a7af9f7a39305> (Oct. 23, 1998), a true and correct copy of which is contained in attached Exhibit N.

Declaration of David Ahn

which are referenced in a December 14, 1994 posting of the Netrek FTP List to the newsgroup rec.answers.¹⁷

13. Among the files currently available in these two mirror directories are the BRM-Hadley 1.7 client source code files and the Vanilla 2.5pl4 server source code files. The BRM-Hadley 1.7 source code files are contained in a compressed archive file named BRMH-1.7.tar.gz and located at <http://ftp.netrek.org/pub/netrek/mirrors/ftp.csua.berkeley.edu.old/netrek/old/>. The Vanilla 2.5pl4 source code files are contained in a compressed archive file named Server2.5pl4.tar.gz and located at <http://ftp.netrek.org/pub/netrek/mirrors/ftp.solace.mh.se/netrek/servers/vanilla/>.¹⁸

14. The BRMH-1.7.tar.gz and Server2.5pl4.tar.gz archive files stored on the Netrek Software Archive carry date stamps of October 16, 1993 and December 15, 1994, respectively. The date stamps can be seen alongside the filenames in the above-referenced directories.¹⁹ Each date stamp represents the date the associated archive file was created. Accordingly, by definition, each file contained in the archive must have been created on or before the indicated date, as shown in the content listings of those archive files.²⁰ To the best of my knowledge, the

¹⁷ Posting of Tom Holub to rec.answers, Subject: Netrek FTP list., <http://groups.google.com/group/rec.answers/msg/ebcb9a14c0d4de78> (Dec. 14, 1994), a true and correct copy of which is contained in attached Exhibit O.

¹⁸ For verification purposes, I wish to note that the MD5 hash of BRMH-1.7.tar.gz is 747acc63aa45b274a25d7ef0121578be, and the MD5 hash of Server2.5pl4.tar.gz is 809f80e34575add74600f21dc052bfad.

¹⁹ Attached hereto as Exhibit P is a true and correct copy of the current content listing of <http://ftp.netrek.org/pub/netrek/mirrors/ftp.csua.berkeley.edu.old/netrek/old/>, which includes the BRMH-1.7.tar.gz archive file and its time stamp, at P-1. Attached hereto as Exhibit Q is a true and correct copy of the current content listing of <http://ftp.netrek.org/pub/netrek/mirrors/ftp.solace.mh.se/netrek/servers/vanilla/>, which includes the Server2.5pl4.tar.gz archive file and its time stamp.

²⁰ Attached hereto as Exhibit R is a true and correct copy of the content listing of the BRMH-1.7.tar.gz archive. Attached hereto as Exhibit S is a true and correct copy of the content listing of the Server2.5pl4.tar.gz archive.

Declaration of David Ahn


date stamps on the BRMH-1.7.tar.gz and Server2.5pl4.tar.gz Netrek source code archive files have never been modified since being posted to the Netrek Software Archive. Furthermore, based on my experience in acquiring source code files and archives from publicly accessible sources over the years, I have no reason to believe that the date stamps on these copies of BRMH-1.7.tar.gz and Server2.5pl4.tar.gz are inaccurate and therefore believe them to correctly reflect the dates those files were created.

15. Based on my general experience with software over the past 16 years, my experience playing Netrek, my extensive involvement in the Netrek community over the past 13 years, discussions with other members of the Netrek community, my personal involvement in creating and maintaining the Netrek Software Archive, the date stamps on the BRMH-1.7.tar.gz and Server2.5pl4.tar.gz source code archive files (which I believe to accurately reflect the dates those archive files were created), my knowledge and recollection of various messages posted to rec.games.netrek, and my extensive experience downloading, experimenting with, and modifying Netrek source code, I can attest as follows. To the best of my knowledge, recollection, and understanding, the BRMH-1.7.tar.gz and Server2.5pl4.tar.gz source code archive files available on the Netrek Software Archive (1) contain versions of the Netrek BRMH client and Netrek Vanilla Server source code files, respectively; (2) became and continued to be disseminated from publicly accessible sources during or before 1994 and substantially continuously thereafter; and (3) were locatable and recognizable from 1994 onward by any person interested and ordinarily skilled in source code development, particularly including members of the Netrek community and those who participated in the rec.games.netrek newsgroup, who exercised reasonable diligence to locate them.

Declaration of David Ahn

I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct.

Executed on this 14 day of December, 2007.



David Ahn

Electronic Patent Application Fee Transmittal

Application Number:				
Filing Date:				
Title of Invention:	SERVER-GROUP MESSAGING SYSTEM FOR INTERACTIVE APPLICATIONS			
First Named Inventor/Applicant Name:	Daniel J. Samuel			
Filer:	Tracy Wesley Druce			
Attorney Docket Number:	18830.0003			
Filed as Large Entity				
ex parte reexam Filing Fees				
Description	Fee Code	Quantity	Amount	Sub-Total in USD(\$)
Basic Filing:				
Request for ex parte reexamination	1812	1	2520	2520
Pages:				
Claims:				
Miscellaneous-Filing:				
Petition:				
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				
Extension-of-Time:				

Description	Fee Code	Quantity	Amount	Sub-Total in USD(\$)
Miscellaneous:				
Total in USD (\$)				2520

Electronic Acknowledgement Receipt

EFS ID:	7804962
Application Number:	90011033
International Application Number:	
Confirmation Number:	1686
Title of Invention:	SERVER-GROUP MESSAGING SYSTEM FOR INTERACTIVE APPLICATIONS
First Named Inventor/Applicant Name:	Daniel J. Samuel
Customer Number:	37086
Filer:	Tracy Wesley Druce
Filer Authorized By:	
Attorney Docket Number:	18830.0003
Receipt Date:	14-JUN-2010
Filing Date:	
Time Stamp:	17:36:56
Application Type:	Reexam (Third Party)

Payment information:

Submitted with Payment	yes
Payment Type	Credit Card
Payment was successfully received in RAM	\$2520
RAM confirmation Number	4465
Deposit Account	
Authorized User	

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
-----------------	----------------------	-----------	-------------------------------------	------------------	------------------

1	Reexam Certificate of Service	CERTIFICATE_OF_SERVICE_523.pdf	16087 b5bfbfd9ae0ab9c93aa421f81bc5dfc920288ea4d	no	1
Warnings:					
Information:					
2	Reexam - Info Disclosure Statement Filed by 3rd Party	IDS_523_.pdf	30587 ab3bf6cc754d4a4ceb45466467e0d4747322731	no	3
Warnings:					
Information:					
3	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	PA_B_rfc1459_IRC_66pg.pdf	88827 306d5d7f73807b5d0abc3776032ff38b50168392	no	66
Warnings:					
Information:					
4	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	PA_C_Packing_Messages_Friedman_19pg.pdf	158585 3181a8b60a840c15ea194eb65a5efee944cbac62	no	19
Warnings:					
Information:					
5	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	PA_D_Van_Hook_An_Approach_to_DIS_Scaleabilty_9pg.pdf	256867 1735f39aa38a6892a351effed101acb236953e42	no	10
Warnings:					
Information:					
6	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	PA_E_IEEE_1278_1993_65pg_.pdf	8236023 8add3bb3cd3b869ab95bdc5e984c99f8bd8ac32	no	65
Warnings:					
Information:					
7	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	PA_F_5736982_Virtual_space_apparatus_with_ava_46pg_.pdf	23110919 52b5269d98cb9b8d57f114c0dd447076305f7ee7	no	46
Warnings:					
Information:					
8	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	PA_G_RING_A_Client_Server_System_10pg.pdf	3847316 4393951b51b386f19536a4bd07c905cf3ed4f856	no	10
Warnings:					
Information:					
9	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	PA_H_History_of_Netrek_McFadden_16pg_.pdf	149424 85f045ab3f6140b992ee94995bfbcc9c78aa66a1	no	16
Warnings:					
Information:					

10	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	PA_I_Macedonia_1995_cga_9pg_.pdf	2047617 ef3176b3b504ac94f5484733d5bf50ca9859786b	no	9
Warnings:					
Information:					
11	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	PAT_A_5822523_27pg_.pdf	2948698 36b5813295c100c0e25e7bc7c40cb09c04cf94b9	no	27
Warnings:					
Information:					
12	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	PAT_B_5822523_Pro_History_250pg_.pdf	6719316 b691bfef73509a0448345ea68e5768fa78ec698f	no	250
Warnings:					
Information:					
13	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	CC_A_B_523_v_NETREK_55pg.pdf	152176 9f691d274a408a71093607b997c976ce260bc7e1	no	55
Warnings:					
Information:					
14	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	CC_C_523_v_Van_hook_and_DIS_18pg.pdf	238222 0f563345ceb5a30b18702902fd7b1ffc17d577cd	no	18
Warnings:					
Information:					
15	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	CC_D_523_IRC RFC_Friedman_Claims_1_2_4_6_6pg.pdf	40163 b9c9a31971723956679696c5da1496ea2efd2185	no	6
Warnings:					
Information:					
16	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	CC_E_523_v_Ring_in_view_of_Netrek_43pg.pdf	239434 98ae3b91600f65d824d461c6a70ade8a2b959568	no	43
Warnings:					
Information:					
17	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	CC_F_523_v_RING_and_Van_Hook_22pg.pdf	249543 76302a5c5b75b997339bed79c9fb007cd7bd450	no	22
Warnings:					
Information:					
18	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	OTH_B_Paltalk_Complaint_17pg_.pdf	597025 b144c77cbb8e134e4ae1e23d06e5a8e527ba29d6	no	17
Warnings:					
Information:					

19	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	OTH_C_90010093_AhnDecl_11pg_.pdf	1293252 370df6e3ead535b334ef1c29197f33349c31a9f0	no	11
Warnings:					
Information:					
20	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	OTH_D_90010093_Order_Granteeing_Reexam_19pg_.pdf	1701541 995e34a771f99ce810bd425a8b456356b2f694ce	no	19
Warnings:					
Information:					
21	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	OTH_E_PalTalk_Opening_CC_Brief_34pg_.pdf	329171 d93bb914eb7710c8bc80600174517738956358d5	no	34
Warnings:					
Information:					
22	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	OTH_F_218_Supplemental_Claim_Construction_Order_9pg_.pdf	83815 31c8bc24f61d22534fbc37c96e579ca624559eb	no	9
Warnings:					
Information:					
23	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	OTH_G_107_Claim_Construction_Order_44pg_.pdf	205829 7ee210465108bf18ae216600ecb6d143a77526c	no	44
Warnings:					
Information:					
24	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	OTH_H_Lipstream_Claim_Construction_Order_15pg_.pdf	1742609 637ef0d497023a1c55ef6c6d83aa703a7aa99b7f	no	15
Warnings:					
Information:					
25	Receipt of Original Ex Parte Reexam Request	523_ex_parte_reexam_52pgs.pdf	371455 c61ad66eb79fb9260fcae615a2f2f76cd6d63582b6	no	52
Warnings:					
Information:					
26	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	OTH_A_0113_5_Smith_declaration_20pg.pdf	4649235 b165323ad36956ec9ea249247954bdeb668de805	no	20
Warnings:					
Information:					
27	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	OTH_I_Netgames_your_guide_to_the_games_1_to_139_of_287pgs.pdf	17114665 fd75a2663b263379f88b04c0ff0143cc5139cb590	no	139
Warnings:					
Information:					

28	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	OTH_I_Netgames_your_guide_to_the_games_140_to_287_of_287pgs.pdf	16154802 dc002a2f02cc359848a5f36015ffaa556a8fa684	no	148
Warnings:					
Information:					
29	Fee Worksheet (PTO-875)	fee-info.pdf	29839 25bd579e9d8ad9c5a0296768f9adfd030ff196b	no	2
Warnings:					
Information:					
Total Files Size (in bytes):				92803042	
<p>This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.</p> <p><u>New Applications Under 35 U.S.C. 111</u> If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.</p> <p><u>National Stage of an International Application under 35 U.S.C. 371</u> If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.</p> <p><u>New International Application Filed with the USPTO as a Receiving Office</u> If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.</p>					

(Also referred to as FORM PTO-1465)

REQUEST FOR EX PARTE REEXAMINATION TRANSMITTAL FORM

Address to:

**Mail Stop Ex Parte Reexam
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450**

Attorney Docket No.: 18330.0003Date: June 14, 2010

1. This is a request for *ex parte* reexamination pursuant to 37 CFR 1.510 of patent number 5,822,523 issued 1998-10-13. The request is made by:
 patent owner. third party requester.
2. The name and address of the person requesting reexamination is:
Novak Druce + Quigg LLP
1000 Louisiana Street, Fifty-Third Floor
Houston, TX 77002
3. a. A check in the amount of \$ _____ is enclosed to cover the reexamination fee, 37 CFR 1.20(c)(1);
 b. The Director is hereby authorized to charge the fee as set forth in 37 CFR 1.20(c)(1) to Deposit Account No. _____; **or**
 c. Payment by credit card. Form PTO-2038 is attached.
4. Any refund should be made by check or credit to Deposit Account No. 14-1437 37 CFR 1.26(c). If payment is made by credit card, refund must be to credit card account.
5. A copy of the patent to be reexamined having a double column format on one side of a separate paper is enclosed. 37 CFR 1.510(b)(4)
6. CD-ROM or CD-R in duplicate, Computer Program (Appendix) or large table
 Landscape Table on CD
7. Nucleotide and/or Amino Acid Sequence Submission
If applicable, items a. – c. are required.
a. Computer Readable Form (CRF)
b. Specification Sequence Listing on:
i. CD-ROM (2 copies) or CD-R (2 copies); **or**
ii. paper
c. Statements verifying identity of above copies
8. A copy of any disclaimer, certificate of correction or reexamination certificate issued in the patent is included.
9. Reexamination of claim(s) 1-6 is requested.
10. A copy of every patent or printed publication relied upon is submitted herewith including a listing thereof on Form PTO/SB/08, PTO-1449, or equivalent.
11. An English language translation of all necessary and pertinent non-English language patents and/or printed publications is included.

[Page 1 of 2]

This collection of information is required by 37 CFR 1.510. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.11 and 1.14. This collection is estimated to take 2 hours to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. **DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS.**

SEND TO: Mail Stop Ex Parte Reexam, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

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

12. The attached detailed request includes at least the following items:
- a. A statement identifying each substantial new question of patentability based on prior patents and printed publications. 37 CFR 1.510(b)(1)
- b. An identification of every claim for which reexamination is requested, and a detailed explanation of the pertinency and manner of applying the cited art to every claim for which reexamination is requested. 37 CFR 1.510(b)(2).
13. A proposed amendment is included (only where the patent owner is the requester). 37 CFR 1.510(e)
14. a. It is certified that a copy of this request (if filed by other than the patent owner) has been served in its entirety on the patent owner as provided in 37 CFR 1.33(c).
The name and address of the party served and the date of service are:
Rajiv P. Patel, Fenwick & West LLP, 2 Palo Alto Square, Palo Alto, CA 94306
Jordan Altman, Shearman & Sterling LLP, 599 Lexington Ave, New York NY 10022
Daniel Devito, 4 Times Square, New York, NY 10036
Date of Service: June 14, 2010; or
- b. A duplicate copy is enclosed because service on patent owner was not possible. An explanation of the efforts made to serve patent owner **is attached**. See MPEP 2220.

15. Correspondence Address: Direct all communications about the reexamination to:

 The address associated with Customer Number:

37086

OR

 Firm or
Individual Name _____

Address _____

City _____

State _____

Zip _____

Country _____

Telephone _____

Email _____

16. The patent is currently the subject of the following concurrent proceeding(s): a. Copending reissue Application No. _____ b. Copending reexamination Control No. _____ c. Copending Interference No. _____ d. Copending litigation styled:PalTalk Holdings Inc. v. Sony Computer EntertainmentAmerica Inc., et. al., Case.No. 2:09cv00274-DF (E.D. Tex.)**WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.**/Tracy W. Druce/

Authorized Signature

June 14, 2010

Date

Tracy W. Druce

Typed/Printed Name

35493

Registration No.

 For Patent Owner Requester For Third Party Requester

Privacy Act Statement

The **Privacy Act of 1974 (P.L. 93-579)** requires that you be given certain information in connection with your submission of the attached form related to a patent application or patent. Accordingly, pursuant to the requirements of the Act, please be advised that: (1) the general authority for the collection of this information is 35 U.S.C. 2(b)(2); (2) furnishing of the information solicited is voluntary; and (3) the principal purpose for which the information is used by the U.S. Patent and Trademark Office is to process and/or examine your submission related to a patent application or patent. If you do not furnish the requested information, the U.S. Patent and Trademark Office may not be able to process and/or examine your submission, which may result in termination of proceedings or abandonment of the application or expiration of the patent.

The information provided by you in this form will be subject to the following routine uses:

1. The information on this form will be treated confidentially to the extent allowed under the Freedom of Information Act (5 U.S.C. 552) and the Privacy Act (5 U.S.C. 552a). Records from this system of records may be disclosed to the Department of Justice to determine whether disclosure of these records is required by the Freedom of Information Act.
2. A record from this system of records may be disclosed, as a routine use, in the course of presenting evidence to a court, magistrate, or administrative tribunal, including disclosures to opposing counsel in the course of settlement negotiations.
3. A record in this system of records may be disclosed, as a routine use, to a Member of Congress submitting a request involving an individual, to whom the record pertains, when the individual has requested assistance from the Member with respect to the subject matter of the record.
4. A record in this system of records may be disclosed, as a routine use, to a contractor of the Agency having need for the information in order to perform a contract. Recipients of information shall be required to comply with the requirements of the Privacy Act of 1974, as amended, pursuant to 5 U.S.C. 552a(m).
5. A record related to an International Application filed under the Patent Cooperation Treaty in this system of records may be disclosed, as a routine use, to the International Bureau of the World Intellectual Property Organization, pursuant to the Patent Cooperation Treaty.
6. A record in this system of records may be disclosed, as a routine use, to another federal agency for purposes of National Security review (35 U.S.C. 181) and for review pursuant to the Atomic Energy Act (42 U.S.C. 218(c)).
7. A record from this system of records may be disclosed, as a routine use, to the Administrator, General Services, or his/her designee, during an inspection of records conducted by GSA as part of that agency's responsibility to recommend improvements in records management practices and programs, under authority of 44 U.S.C. 2904 and 2906. Such disclosure shall be made in accordance with the GSA regulations governing inspection of records for this purpose, and any other relevant (*i.e.*, GSA or Commerce) directive. Such disclosure shall not be used to make determinations about individuals.
8. A record from this system of records may be disclosed, as a routine use, to the public after either publication of the application pursuant to 35 U.S.C. 122(b) or issuance of a patent pursuant to 35 U.S.C. 151. Further, a record may be disclosed, subject to the limitations of 37 CFR 1.14, as a routine use, to the public if the record was filed in an application which became abandoned or in which the proceedings were terminated and which application is referenced by either a published application, an application open to public inspection or an issued patent.
9. A record from this system of records may be disclosed, as a routine use, to a Federal, State, or local law enforcement agency, if the USPTO becomes aware of a violation or potential violation of law or regulation.

Electronic Acknowledgement Receipt

EFS ID:	7810215
Application Number:	90011033
International Application Number:	
Confirmation Number:	1686
Title of Invention:	SERVER-GROUP MESSAGING SYSTEM FOR INTERACTIVE APPLICATIONS
First Named Inventor/Applicant Name:	Daniel J. Samuel
Correspondence Address:	- - - - - - -
Filer:	Tracy Wesley Druce
Filer Authorized By:	
Attorney Docket Number:	18830.0003
Receipt Date:	14-JUN-2010
Filing Date:	
Time Stamp:	18:02:06
Application Type:	Reexam (Third Party)

Payment information:

Submitted with Payment	no
------------------------	----

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1	Reexam Miscellaneous Incoming Letter	Transmittal_Form_523_3pgs.pdf	778145 f588926222126c15d7d6f46f6b6ece9e2c14762e	no	3
Warnings:					
Information:					
Total Files Size (in bytes):			778145		
<p>This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.</p> <p><u>New Applications Under 35 U.S.C. 111</u> If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.</p> <p><u>National Stage of an International Application under 35 U.S.C. 371</u> If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.</p> <p><u>New International Application Filed with the USPTO as a Receiving Office</u> If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.</p>					

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Control No.:

Inventors: Rothschild, Jeffrey J., Marc P.
Kwaitkowski and Daniel J. Samuel

Patent No.: 5,822,523

Filed: February 1, 1996

Issued: October 13, 1998

Title: Server-group messaging system for
interactive applications

REQUEST FOR REEXAMINATION UNDER
35 U.S.C. §§ 302-307 AND
37 C.F.R. § 1.510

Mail Stop *Ex Parte* Reexamination
ATTN: Central Reexamination Unit
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

REQUEST FOR EX PARTE REEXAMINATION OF U.S. PATENT NO. 5,822,523

TABLE OF CONTENTS

I. REQUIREMENTS FOR EX PARTE REEXAMINATION UNDER 37 C.F.R. § 1.510
7

A. 37 C.F.R. § 1.510 (b)(1) and (b)(2): Statement Pointing Out Each Substantial New Question of Patentability and Detailed Explanation of the Pertinency and Manner of Applying the Cited Prior Art to Every Claim for Which Reexamination is Requested 7

B. 37 C.F.R. § 1.510 (b)(3): Copy of Every Patent or Printed Publication Relied Upon to Present a SNQ 7

C. 37 C.F.R. § 1.510 (b)(4): Copy of the Entire Patent for which Reexamination Is Requested..... 8

D. 37 C.F.R. § 1.510 (b)(5): Certification that a Copy of the Request has been Served in its Entirety on the Patent Owner 8

E. 37 C.F.R. § 1.510 (a): Fee for Requesting Reexamination..... 8

F. Related Co-Pending Litigation Requires Treatment with Special Dispatch and Priority Over all Other Cases. 9

II. OVERVIEW OF THE ‘523 PATENT AND ITS PROSECUTION HISTORY . 9

A. Summary of Preferred Embodiments and Claims of the ‘523 Patent 9

B. ‘523 Patent Application Prosecution History 10

C. Claim Construction 12

1. Standard 12

2. Previous Litigation Claim Constructions 13

III. SUMMARY OF THE PRIOR ART 13

IV. 37 C.F.R. § 1.510 (b)(1): STATEMENT POINTING OUT EACH SUBSTANTIAL NEW QUESTION OF PATENTABILITY 19

A. Netrek Alone Presents a Substantial New Question of Patentability with Respect to Claims 1-6 of the ‘523 Patent 19

B. Netrek in Combination with McFadden Presents a Substantial New Question of Patentability with Respect to Claims 1-6 of the ‘523 Patent 19

C. Van Hook in Combination with DIS Presents a Substantial New Question of Patentability with Respect to Claims 1, 2 and 4-6 of the ‘523 Patent..... 20

D. IRC RFC in Combination with Friedman Presents a Substantial New Question of Patentability with Respect to Claims 1, 2 and 4-6 of the ‘523 Patent 21

E. RING in Combination with Netrek Provides a Substantial New Question of Patentability with Respect to Claims 1-6 of the ‘523 patent..... 22

F. RING in Combination with Van Hook Provides a Substantial New Question of Patentability with Respect to Claims 1, 2 and 4-6 of the ‘523 patent..... 23

V. DETAILED EXPLANATION UNDER 37 CFR 1.510(b) OF THE PERTINENCY AND MANNER OF APPLYING THE CITED PRIOR ART TO EVERY CLAIM FOR WHICH REEXAMINATION IS REQUESTED 23

A. Claims 1-6 Are Anticipated by Netrek Under 35 U.S.C. § 102.....	23
B. Claims 1-6 Are Rendered Obvious by Netrek in view of McFadden under 35 U.S.C. § 103.....	32
C. Claims 1, 2 and 4-6 Are Rendered Obvious by Van Hook in view of DIS under 35 U.S.C. § 103.....	32
D. Claims 1, 2 and 4-6 Are Rendered Obvious by IRC RFC in view of Friedman under 35 U.S.C. § 103.....	37
E. Claims 1-6 Are Rendered Obvious by RING in view of Netrek under 35 U.S.C. § 103 .	40
F. Claims 1, 2 and 4-6 Are Rendered Obvious by RING in view of Van Hook under 35 U.S.C. § 103.....	46
VI. CONCLUSION	51

TABLE OF EXHIBITS

LIST OF EXHIBITS

The exhibits to the present Request are arranged in four groups: prior art (“PA”); relevant portions of patent prosecution file history, patents, and claim dependency relationships (“PAT”); claim charts (“CC”); and other (“OTH”).

A. PRIOR ART (PA)

PA-SB08	USPTO Form SB/08
PA-A	Server2.5pl4.tar.gz (“Server Code”) and BRMH-1.7.tar.gz (“Client Code”) (source code dated no later than August 1994 ¹) (“Netrek”)
PA-B	J. Oikarinen et al., RFC 1459- Internet Relay Chat Protocol, published May 1993 (“IRC RFC”)
PA-C	R. Friedman et al., Packing Messages as a Tool for Boosting the Performance of Total Ordering Protocols, Dept. of Science of Cornell University, published July 7, 1995 (“Friedman”)
PA-D	Daniel J. Van Hook, James O. Calvin, Michael K. Newton, and David A. Fusco, “An Approach to DIS Scaleability,” 11 th DIS Workshop, 26-30 Sept. 1994 (“Van Hook”)
PA-E	IEEE 1278-1993 IEEE Standard for Information Technology- Protocols for Distributed Interactive Simulation Applications, approved March 18, 1993, and published in 1993 (“DIS”)
PA-F	U.S. Patent No. 5,736,982 to Suzuki (“Suzuki”)
PA-G	T. A. Funkhouser, “RING: A Client-Server System for Multi-User Virtual Environments,” Association of Computing Machinery, 1995 Symposium on Interactive 3D Graphics, Monterey CA, April 9-12, 1995 ² (“RING”)
PA-H	Andy McFadden, “The History of Netrek”, published January 1, 1994 (“McFadden”)
PA-I	Michael R. Macedonia, “Exploiting Reality with Multicast Groups”, published September 1995 (“Macedonia”)

B. RELEVANT PATENT MATERIALS (PAT)

PAT-A	U.S. Patent No. 5,822,523 (“the ‘523 patent”)
PAT-B	Prosecution history of the ‘523 patent

¹ See also, The Ahn declaration (OTH-C) at ¶¶ 7-10 and 15 (supporting public availability of Netrek source code no later than August 1994).

² See <http://portal.acm.org/toc.cfm?id=199404> (indicating the Association of Computing Machinery, 1995 Symposium on Interactive 3D Graphics, Monterey CA, including the presentation for RING, occurred between April 9-12).

C. CLAIM CHARTS (CC)

- CC-A Claim Chart comparing Claims 1-6 of U.S. Patent No. 5,822,523 to the disclosure in Netrek
- CC-B Claim Chart comparing Claims 1-6 of U.S. Patent No. 5,822,523 to the disclosure in Netrek in view of McFadden
- CC-C Claim Chart comparing Claims 1, 2 and 4-6 of U.S. Patent No. 5,822,523 to the disclosure in Van Hook in view of DIS
- CC-D Claim Chart comparing Claims 1, 2 and 4-6 of U.S. Patent No. 5,822,523 to the disclosure of IRC RFC in view of Friedman
- CC-E Claim Chart comparing Claims 1-6 of U.S. Patent No. 5,822,523 to the disclosure of RING in view of Netrek
- CC-F Claim Chart comparing Claims 1, 2 and 4-6 of U.S. Patent No. 5,822,523 to the disclosure of RING in view of Van Hook

D. OTHER DOCUMENTS (OTH)

- OTH-A Declaration of Kevin Smith (“the Smith declaration”)
- OTH-B Complaint filed in *Paltalk Holdings, Inc. v. Sony Computer Entertainment America, Inc., et. al.*, (E.D. Tex.), Case No. 2:09cv00274-DF
- OTH-C Declaration of David Ahn (“the Ahn declaration”)
- OTH-D Reexamination Ctrl. No. 90/001,093 Determination Ordering Reexamination dated February 29, 2008
- OTH-E Paltalk’s Corrected Second Opening Claim Construction Brief filed on December 31, 2007 in *Paltalk Holdings, Inc. v. Microsoft Corp.* (E.D. Tex.), Case No. 2:06-cv-00367-DF
- OTH-F Paltalk’s Second Reply Brief on Claim Construction filed on January 7, 2008 in *Paltalk Holdings, Inc. v. Microsoft Corp.* (E.D. Tex.), Case No. 2:06-cv-00367-DF
- OTH-G Claim Construction order issued on July 29, 2008 in *Paltalk Holdings, Inc. v. Microsoft Corp.* (E.D. Tex.), Case No. 2:06-cv-00367-DF
- OTH-H Claim Construction order issued on August 25, 2000 in *HearMe v. Lipstream Networks, Inc.* (N.D. Cal.), Case No. 99-04506 WHA
- OTH-I Kelly Maloni, Derek Baker and Nataniel Wice “Netgames ... Your Guide to the Games People Play on the Electronic Highway” published 1994 (“Maloni”)

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Control No:

Inventors: Rothschild, Jeffrey J., Marc P.
Kwaitkowski and Daniel J. Samuel

Patent No.: 5,822,523

Filed: February 1, 1996

Issued: October 13, 1998

Title: Server-group messaging system for
interactive applications

REQUEST FOR REEXAMINATION UNDER
35 U.S.C. §§ 302-307 AND
37 C.F.R. § 1.510

Mail Stop *Ex Parte* Reexamination
ATTN: Central Reexamination Unit
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

REQUEST FOR *EX PARTE* REEXAMINATION OF U.S. PATENT 5,822,523

Dear Sir or Madam:

The undersigned hereby respectfully requests reexamination, pursuant to 35 U.S.C. §§ 302-307 and 37 C.F.R. § 1.510, of Claims 1-6 of U.S. Patent No. 5,822,523 (“the ‘523 patent”), filed February 1, 1996, and issued October 13, 1998, to Jeffrey J. Rothschild, Marc P. Kwaitkowski and Daniel J. Samuel (Exhibit PAT-A). Reexamination is requested in view of the substantial new questions of patentability (“SNQ”) presented below. Requester reserves all rights and defenses available including, without limitation, defenses as to invalidity and unenforceability. By simply filing this Request in compliance with the Patent Rules, Requester does not represent, agree or concur that the ‘523 patent is enforceable, and by asserting the SNQ herein, Requester specifically asserts that Claims 1-6 of the ‘523 patent are in fact not patentable. As such, the U.S. Patent and Trademark Office (the “Office”) should reexamine and find Claims 1-6 unpatentable and cancel

Claims 1-6 of the '523 patent, rendering Claims 1-6 of the '523 patent null, void, and otherwise unenforceable.

Reexamination is requested in view of the teachings of the references cited herein. The SNQs established by these references teach the elements recited by Claims 1-6 of the '523 patent and, importantly, teach the elements that were argued as novel during prosecution. Further, none of the references submitted as part of this reexam were cited or discussed during prosecution of the '523 patent. As described more fully below, reexamination is appropriate in view of the patents and printed publications prior art cited herein, which alone or in combination with other prior art provide new technical teachings not previously considered with respect to the claims herein requested for reexamination.

The Requester submits that reexamination should be granted and that Claims 1-6 be found unpatentable by issuance of a Certificate of Reexamination canceling all claims.

I. REQUIREMENTS FOR EX PARTE REEXAMINATION UNDER 37 C.F.R. § 1.510

Requester satisfies each requirement for *ex parte* reexamination of the '523 patent.

A. 37 C.F.R. § 1.510 (b)(1) AND (b)(2): STATEMENT POINTING OUT EACH SUBSTANTIAL NEW QUESTION OF PATENTABILITY AND DETAILED EXPLANATION OF THE PERTINENCY AND MANNER OF APPLYING THE CITED PRIOR ART TO EVERY CLAIM FOR WHICH REEXAMINATION IS REQUESTED

A statement pointing out each substantial new question of patentability (“SNQ”) based on the cited patents, and a detailed explanation of the pertinence and manner of applying the cited patents to Claims 1-6 of the '523 patent is presented below in accordance with 37 C.F.R. § 1.510 (b)(1) and (b)(2).

The SNQs raised herein are based on prior art that was not cited or discussed during the prosecution of the '523 patent. The references, alone or in combination, are not cumulative to the prior art discussed during the original prosecution of the '523 patent. Thus, they are appropriate for use in supporting the SNQs of patentability raised herein.

B. 37 C.F.R. § 1.510 (b)(3): COPY OF EVERY PATENT OR PRINTED PUBLICATION RELIED UPON TO PRESENT A SNQ

A copy of every patent or printed publication relied upon to present a SNQ is submitted herein, pursuant to 37 C.F.R. § 1.510(b)(3), as Exhibits PA-A through Exhibits PA-I, citation of

which may be found on the accompanying Form PTO-SB/08 at Exhibit PTO-SB/08. Each of these cited prior art patents and printed publications constitutes effective prior art as to the claims of the '523 patent under 35 U.S.C. § 102 and 35 U.S.C. § 103. **PA-A was separately submitted on a compact disc to the Office, in the care of Manuel Saldana Jr., on the same date as this request was filed, June 14, 2010.**

C. 37 C.F.R. § 1.510 (b)(4): COPY OF THE ENTIRE PATENT FOR WHICH REEXAMINATION IS REQUESTED

A full copy of the '523 patent is submitted herein as Exhibit PAT-A in accordance with 37 C.F.R. § 1.510(b)(4).

D. 37 C.F.R. § 1.510 (b)(5): CERTIFICATION THAT A COPY OF THE REQUEST HAS BEEN SERVED IN ITS ENTIRETY ON THE PATENT OWNER

A copy of this request has been served in its entirety on the Patent Owner in accordance with 37 C.F.R. § 1.510(b)(5) at the following address:

DANIEL DEVITO
PATENT DEPARTMENT
SKADDEN, ARPS, SLATE, MEAGHER & FLOM LLP
FOUR TIMES SQUARE
NEW YORK NY 10036

Also as a courtesy, a copy of this request is being sent to two correspondence addresses of recent assignees:

FENWICK & WEST LLP
RAJIV P. PATEL, ESQ.
TWO PALO ALTO SQUARE
PALO ALTO, CA 94306

and

JORDAN ALTMAN 599 LEXINGTON AVENUE SHEARMAN &
STERLING LLP - IP DOCKETING
NEW YORK, NY 10022

E. 37 C.F.R. § 1.510 (a): FEE FOR REQUESTING REEXAMINATION

In accordance with 37 C.F.R. § 1.510(a), a credit card authorization to cover the fee for reexamination of \$2,520.00 is attached. If this authorization is missing or defective, please charge the Fee to the Novak Druce and Quigg Deposit Account No. 14-1437.

F. RELATED CO-PENDING LITIGATION REQUIRES TREATMENT WITH SPECIAL DISPATCH AND PRIORITY OVER ALL OTHER CASES.

The '523 patent is presently the subject of *PalTalk Holdings v. Sony Computer Entertainment America, et al.*, (E.D. Tex.) See OTH-A, Complaint filed by Paltalk Holdings.

Pursuant to 35 U.S.C. § 305, Requester respectfully urges that this Request be granted and reexamination conducted not only with “**special dispatch**,” but also with “**priority over all other cases**” in accordance with 37 C.F.R. 1.550(a) due to the ongoing nature of the underlying litigation.

II. OVERVIEW OF THE '523 PATENT AND ITS PROSECUTION HISTORY

A. SUMMARY OF PREFERRED EMBODIMENTS AND CLAIMS OF THE '523 PATENT

The '523 patent generally relates to a method for deploying interactive applications over a network containing host computers and a group messaging server. The '523 patent at Abstract. More specifically, the disclosure relates to an interactive application, wherein many messages are arriving at the group server close to one another in time. Rather than simply forwarding each message to its targeted hosts, the group messaging server aggregates the contents of the messages received during a specified time period, and then sends an aggregated message to the targeted hosts. The '523 patent at Abstract.

The method is described in the context of interactive computer applications, and specifically video game applications, wherein a plurality of users can interact through the game, although the claims are not so limited. See the '523 patent at 1:15-17; 27:35-38.

The claims recite a method of providing a group-messaging server that maintains a list of message groups. The message groups each have at least one host computer, but at least one of the message groups must have a plurality of host computers within the message group. Each host computer, as well as the group-message server, is connected by a unicast network. The '523 patent at Col. 25, lines 45-64.

A plurality of computers from within one of the message groups send messages to the group-messaging server. The group messaging server aggregates all of the messages received within a given time period according to a criterion, forms a message from the aggregation of received messages, and sends the aggregated message to a host computer within the given message group. *Id.*

B. '523 PATENT APPLICATION PROSECUTION HISTORY

The '523 patent was filed on February 1, 1996, as application serial number 08/595,323 (the "'323 application"). The '323 application contained 16 claims, but only claims 7-12 issued. Distinguishing claims 7-12 from the canceled claims in the '323 application is the presence of limitations requiring aggregating portions of messages sent from a plurality of host computers in a message into a single message, which is transmitted to a host computer within the message group. *See* PAT-B, pros. hist. of the '523 patent, claims filed February 1, 1996.

All claims of the '323 application were initially rejected under 35 U.S.C. § 103 over Page (U.S. Patent No. 5,329,619) in view of Perlman (U.S. Patent No. 5,309,437) in an Office Action that issued March 20, 1997. The Examiner found that Page taught all of the limitations of the claims except for the message server and the unicast network; however, the Examiner found that the message server would have been obvious in view of the functions of Page's broker server. *See* PAT-B, pros. hist. of the '523 patent, Office Action issued March 20, 1997. The Examiner also found that Perlman taught the required unicast network and that the teachings of the two references were combinable in view of the fact that a unicast implementation was well known in the art. *See* PAT-B, pros. hist. of the '523 patent, Office Action issued March 20, 1997.

On June 25, 1997, the Applicants of the '323 application (the "Applicants") responded by canceling all claims, except for claims 7-12. *See* PAT-B, pros. hist. of the '523 patent, Office Action Response filed June 25, 1997. As addressed above, claims 7-12 were distinguishable from the canceled claims because they recited various "aggregating" limitations. By canceling the claims that did not recite the "aggregating" limitations, the Applicants have effectively admitted that the only potentially novel features of their claims relate to aggregation. The Applicants' remarks support this proposition, as they argue only that Page does not teach any of the "aggregating" limitations. *See* PAT-B, pros. hist. of the '523 patent, Response filed June 25, 1997 at p. 2 (See p. 2 reproduced below for convenience).

1. **Rejection of Claims 7-12 Under 35 U.S.C. §103 Over Page in View of Perlman.**

The Examiner has rejected claims 7-12 as obvious over Page et al in view of Perlman et al. The Applicant respectfully traverses this rejection. In particular, claims 7-12 require the steps of

- sending, by a plurality of host computers belonging to a first message group, messages to said server . . . , said messages containing a payload portion . . . ;
- aggregating, by said server in a time interval determined in accordance with a predefined criterion, said payload portions of said messages to create an aggregated payload;
- forming an aggregated message using said aggregated payload

Page does not teach these claim elements. In particular, Page teaches a service broker that manages service requests and responsive services communicated between servers and clients. Page teaches three modes of communication: message processing, conversational communication, and remote procedure call.

None of Page's modes of communication aggregate payloads of messages into an aggregated payload where the payloads are being sent from a plurality of host computers. See Page, e.g. Col. 5, line 38 - Col. 6, line 68. None of Page's other features relate to aggregating payloads of messages being sent from a plurality of host computers.

The cleanup manager identified by the Examiner is a part of the service broker. The clean-up manager processes timeouts that have occurred. Please see Page, Col. 25, line 48 - Col. 27, line 42. It does not aggregate payloads of network messages. It recovers entries in various tables of the service broker for reuse. Please see Col. 27, lines 10-12.

Perlman does not overcome this deficiency of Page. Perlman involves a device that couples segments of an extended local area network such that messages that employ "inter-network protocols" will be handled without the difficulties usually associated with bridges and without the complexity and expense of full IP routers. Perlman does not teach aggregating payloads of messages.

Accordingly, the combination of Page and Perlman does not teach the inventions of claims 7-12.

The Examiner apparently agreed with the Applicants' remarks and issued a Notice of Allowability on July 9, 1997. While the Examiner does not provide an explicit reason for allowance, it can be deduced that the Examiner agreed that the only potential novel and non-obvious limitation of the pending (and now issued) claims was the "aggregating" limitation.

In view of the above, a substantial new question of patentability ("SNQ") is raised herein by each of the several proposed SNQs because each demonstrates that aggregating multiple messages sent from a plurality of host computers within a message group, and transmitting the aggregated messages to a host computer in the message group, was well known before the '523 patent was filed.

C. CLAIM CONSTRUCTION

1. STANDARD

Requester notes that for purposes of this Request, the claim terms are presented by the Requester in accordance with 37 C.F.R § 1.555(b) and MPEP § 2111. Specifically, each term of the claims is to be given its "broadest reasonable construction" consistent with the specification. MPEP § 2111; *In re Trans Texas Holding Corp.*, 498 F.3d 1290, 1298 (Fed. Cir. 2007) (citing *In re Yamamoto*, 740 F.2d 1569, 1571 (Fed. Cir. 1984)). As the Federal Circuit noted in *Trans Texas*, the Office has traditionally applied this standard during reexamination, and does not interpret claims as a court would interpret claims. MPEP § 2111. The Office is not bound by any prior district court claim construction. *Trans Texas*, 498 F.3d at 1297, 1301. Rather:

the PTO applies to the verbiage of the proposed claims the broadest reasonable meaning of the words in their ordinary usage as they would be understood by one of ordinary skill in the art, taking into account whatever enlightenment by way of definitions or otherwise that may be afforded by the written description contained in the applicant's specification.

In re Morris, 127 F.3d 1048, 1054 (Fed. Cir.1997). The rationale underlying the "broadest reasonable construction" standard is that it reduces the possibility that a claim, after issue or certificate of reexamination, will be interpreted more broadly than is justified. 37 C.F.R § 1.555(b), MPEP § 2111.

Because the standards of claim interpretation used in the courts in patent litigation are different from the claim interpretation standards used in the Office in claim examination proceedings (including reexamination), any claim interpretations submitted herein for the

purpose of demonstrating a SNQ are neither binding upon litigants in any litigation related to the '523 patent, nor do such claim interpretations correspond to the construction of claims under the legal standards that are mandated to be used by the courts in litigation. See 35 U.S.C. § 305; MPEP § 2286 II (determination of a SNQ is made independently of a court's decision on validity because of different standards of proof and claim interpretation employed by the District Courts and the Office); see also *Trans Texas*, 498 F.3d at 1298 (Court upheld Office rejections that conflicted with outcome of litigation); *In re Zletz*, 893 F.2d 319, 322, 13 USPQ2d 1320, 1322 (Fed. Cir. 1989).

2. PREVIOUS LITIGATION CLAIM CONSTRUCTIONS

In previous litigation PalTalk had made statements regarding the construction of the claim terms of the '523 patent. See OTH-E and OTH-F, Claim Construction briefs submitted by PalTalk in *PalTalk Holdings, Inc. v. Microsoft Corp.*

In addition two courts have construed terms of the '523 patent. See OTH-G and OTH-H, Claim construction orders from *PalTalk Holdings, Inc. v. Microsoft Corp.* and *HearMe v. Lipstream Networks, Inc.*

As noted above, the claim construction standard used in litigation is not as broad as the standard applied in reexamination. In this reexamination proceeding, the Examiner should apply the broadest construction consistent with the specification. However, the broadest reasonable construction should be at least as broad as constructions argued by the Patent Owner in litigation or adopted by a District Court.

III. SUMMARY OF THE PRIOR ART

1. Netrek

i. Netrek is a printed publication that was published no later than August 1994

Netrek is source code for a client-server game, loosely based on the *Star Trek* television show, which game is played over a computer network. *Server/docs/README* at lines 221, 238. Because both the client and the server each needed their own code for the game, the source code is broken into two parts, the "BRMH-1.7.tar.gz" Client Code and the "Server2.5pl4.tar.gz" Source Code.

In an unrelated reexamination proceeding, Control No. 90/010,093, the reexamination request submitted the exact same “BRMH-1.7.tar.gz” Client Code and the “Server2.5pl4.tar.gz” Source Code relied on in the instant proceeding as prior art. In the Order granting the reexamination request in that proceeding, the Examiner agreed that the Netrek source code was a printed publication available as prior art based on the declaration of David Ahn (“Ahn Declaration”). See OTH-D, Order granting reexamination at 9-11; See also OTH-C, Declaration of David Ahn. The Ahn Declaration equally supports the use of Netrek as prior art in the instant reexamination request. Moreover, OTH-I, Maloni, is offered as additional evidence of the popularity and public availability of the Netrek source code on various FTP servers and Usenet newsgroups (e.g., “ftp rtfm.mit.edu” and “rec.games.netrek”). Maloni at pp. 48, 49.

As stated in the Ahn Declaration, the BRMH-1.7 Client Code was publically available at least by October 15, 1993. OTH-D, Ahn Declaration at ¶ 10. Further, the Server2.5pl4.tar.gz file, called the Vanilla 2.5 pl4 by Ahn, was available by at least August 17, 1994. *Id.* The “BRMH-1.7.tar.gz” and “Server2.5pl4.tar.gz” files submitted in this reexamination request were downloaded from the web addresses specified by Ahn. *Id.* at 13. Ahn declares that these files had been publically available and accessible continuously during or before 1994. *Id.* at 15.

Therefore Netrek is prior art no later than August 17, 1994, which is when the Server2.5pl4.tar.gz Server Code was publically available (the BRMH-1.7 Client Code was publically available prior to 1994).

ii. Netrek establishes an SNQ

As discussed above, Netrek was published no later than August 1994, and accordingly is prior art under 35 U.S.C. § 102(b). Netrek was not cited or discussed in the prosecution of the ‘523 patent.

Netrek discloses a client-server game, loosely based on the *Star Trek* television show, which game is played over a computer network. *Server/docs/README* at lines 221, 238. The clients communicate among each other through the server using messages over an Internet protocol (e.g., TCP/IP). *Server\ntserv\input.c* at line 195 and *Server\ntserv\socket.c* at line 688. Specifically, the server reads a socket containing data sent from a client. *Server\ntserv\input.c* at line 195. The server places the client messages into shared memory. *Server\ntserv\socket.c* at lines 1825-2044. After a specified period of time, or when the buffer is full, the server will form

aggregated messages from the received client message payloads and send these aggregated messages out to the clients. *Server\ntserv\socket.c* at lines 603-90. Examples of messages that clients can send to each other via the server include text chat, torpedo (indicating a torpedo has been launched), and plasma messages (indicating plasma has been fired). *brmh-1.7\packets.h*. These messages can be used, for example, by Federation star ships to destroy rival Klingon starships. *Server\robots\basep.c* at line 33 and *Server\ntserv\socket.c* at lines 1125-92. Accordingly, Netrek teaches aggregating various incoming messages, thereby allowing the server to send fewer messages than it receives (e.g., “Idea: read from client often, send to client not so often”). *Server\ntserv\input.c* at lines 152-203; *Server\ntserv\redraw.c* at lines 21-115; *Server\ntserv\socket.c* at lines 603-90.

2. McFadden

McFadden was published no later than May 1, 1994, and accordingly is prior art under 35 U.S.C. § 102(b). McFadden was not cited in the prosecution of the ‘523 patent.

McFadden is a FAQ and history of the Netrek online game. McFadden at p. 2, § 0.2; p. 8, §, 2.1.2 Architecture; and p. 11, §3.3.1 Client/Server. McFadden shows inherent characteristics of Netrek and is further presented in this request to provide context and understanding to the Netrek source code. McFadden describes Netrek as “a real-time graphical multiplayer arcade/strategy game played over the Internet. Players form into teams and fight for control of the galaxy, dogfighting and taking planets.” McFadden at p. 2, § 0.2. McFadden is presented in this request to provide context and understanding to the Netrek source code.

3. Van Hook

Van Hook was published September 1994, and accordingly is prior art under 35 U.S.C. § 102(b). Van Hook was not cited or discussed in the prosecution of the ‘523 patent.

Van Hook discloses techniques that have been developed and deployed for Advanced Research Projects Agency’s (ARPA) Synthetic Theater of War - Europe (“STOW-E”) computer battlefield simulation program, which uses the Distributed Interactive Simulation (“DIS”) battlefield simulation protocol. Van Hook at p. 1, 1.0 Introduction. In the STOW-E program, a virtual world simulates battlefield conditions, and “[e]xplicit representations of command, control, and communication are required to permit command forces to transmit orders to and

receive reports from a new generation of more intelligent semi-automated forces.” Van Hook at p. 1, 1.0, Introduction. The simulation disclosed in Van Hook is deployed over the Defense Simulation Internet Wide Area Network utilizing DIS 2.0.3 protocols, wherein protocol data units (PDUs) containing entity-state information are exchanged between host computers via an Application Gateway (AG) server. *Id.* at pp. 1, 2. In Van Hook, the AG server bundles the PDUs from host computers into larger transmission packets to be distributed. The host computers in Van Hook can also form message groups such as cell sets., exercises, and forces. *Id.* at p. 66, 4.6 Bundling.

4. IEEE 1278-1993: IEEE Standard for Information Technology- Protocols for Distributed Interactive Simulation Applications (“DIS”)

DIS was published in 1993 and accordingly is prior art under 35 U.S.C. § 102(b). DIS was not cited or discussed in the prosecution of the ‘523 patent.

DIS is part of a proposed set of standards for distributed interactive simulation wherein a “synthetic environment is created through real-time exchange of data units [PDUs] between distributed, computationally autonomous simulation applications in the form of simulations . . . interconnected through standard computer communicative services.” DIS at pp. 3, 3.8 distributed interactive simulation (DIS). The standard computer services in DIS consist of a communication architecture that supports multicast data packets. DIS at p. 10, 4.3 communication services. The PDU packets disclosed in DIS include a PDU header, an ID denoting a host computer (Entity ID), an exercise (Exercise ID) as well as a message group (Force ID), and the PDU message relating to positional information of the entity. DIS at pp. 40-41, Table 18.

5. Macedonia

Macedonia was published in September 1995 and accordingly is prior art under 35 U.S.C. § 102(a). DIS was not cited or discussed in the prosecution of the ‘523 patent.

Macedonia discloses an implementation of DIS that seeks to expand the number of users capable of participating in a simulation. Macedonia at 38. Macedonia describes groups of participants, e.g., those within proximity of other players, and the ability to join these groups using messages (e.g., “Join Request PDUs”). Macedonia at 42.

6. RING

RING was published in April 1995 and accordingly is prior art under 35 U.S.C. § 102(a). RING was not cited or discussed during the prosecution of the '523 patent.

RING discloses a system that supports real-time visual interaction between a large number of users in a shared 3-D environment. RING at Abstract. RING facilitates communication over a unicast network amongst a plurality of hosts via a centralized server, or collection of servers. RING at p. 91.

To reduce the number of messages sent between the servers and the host computers, the centralized servers cull, augment, and alter the messages to send only relevant messages to relevant hosts in a limited number of communications. RING at p. 87. This is accomplished by determining the visibility of each host's virtual representation in the virtual environment. *Id.* Only information pertaining to objects within the line of sight of a host's virtual representation is transmitted to the host. *Id.* All other information is culled. *Id.* In this way, the number of messages sent to each host can be markedly reduced.

Further, related to the claims of the subject patent, RING discloses a server that communicates with a plurality of hosts. All the hosts, or a subset of all the hosts, can send messages to the server via a unicast network informing the server, and thereby the other hosts, of the movements of that host's virtual representation in the virtual environment. If another host cannot see that host, the two hosts are not part of the same group and the message is culled. But if the several hosts are all in the line of sight of each other, each host will receive a message transmitted from the server describing the movement of each host's virtual representation. *Id.*

7. IRC RFC

IRC RFC was published May 1993, and accordingly is prior art under 35 U.S.C. § 102(b). IRC RFC was not cited or discussed in the prosecution of the '523 patent.

IRC RFC discloses protocols for implementing Internet Relay Chat (IRC). IRC was not cited or discussed during the prosecution of the '523 patent. "The main goal of IRC is to provide a forum which allows easy and efficient conferencing (one-to-many conversations)." IRC RFC at p. 11, § 3.2 One-to-many. IRC uses a client-server configuration where a client sends a

channel message to a server and the server distributes the message to the other clients who have joined that channel. IRC RFC at p. 11, § 3.2.2; see also Fig. 2 (reproduced below).

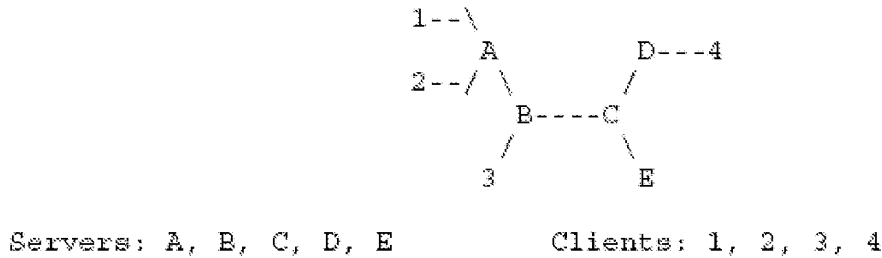


Figure 2 of IRC RFC at 3.0

Clients can join channels using the “JOIN” command (or, if the channel does not exist, it is created in response to the join request). IRC RFC at p. 5, § 1.3 Channels; *See also Id.* at p. 19, § 4.2.1. Join message. Messages sent from clients to the server include the name of the destination channel and the data for the message. IRC RFC at p. 32, § 4.4.1. Private messages. The servers maintain a list of all channels and the clients joined to those channels. IRC RFC at p. 63, § 9.2.2 Channels.

8. Friedman

Friedman was published July 7, 1995, and accordingly is prior art under 35 U.S.C. § 102(a). Friedman was not cited or discussed during the prosecution of the ‘523 patent.

Friedman discloses the results of the latency and throughput of standard network protocols compared to modified protocols that aggregate packets by packing multiple packets into a single packet. Friedman at p. 1. Friedman notes that a protocol modified to use aggregation (*e.g.*, “packing messages”) “improves both the latency and throughput.” *Id.* One example of a modified protocol is the Dynseq protocol, which aggregates based on a preset time interval, which is less than the expected user latency, such as one millisecond. Friedman at p. 5. Friedman discloses that the reasons for the improved efficiency are due to the reduction of packet headers, reduced link contention, and less CPU interrupts. Friedman at p. 12. Friedman teaches to one of ordinary skill in the art that one way to increase network efficiency is to aggregate packets before sending.

**IV.37 C.F.R. § 1.510 (b)(1): STATEMENT POINTING OUT EACH
SUBSTANTIAL NEW QUESTION OF PATENTABILITY**

**A. NETREK ALONE PRESENTS A SUBSTANTIAL NEW QUESTION OF PATENTABILITY
WITH RESPECT TO CLAIMS 1-6 OF THE '523 PATENT**

Netrek was not cited or discussed during the prosecution of the '523 patent and thus is new prior art with respect to the '523 patent. Netrek raises a SNQ with regards to claims 1-6 because Netrek discloses the technical teachings of a client-server game, loosely based on Star Trek, that is played over a computer network. *Server/docs/README* at lines 221, 283 and McFadden at p. 2, § 0.2; p. 8, §2.1.2 Architecture; and p. 11, §3.3.1 Client/Server. Netrek further discloses the technical teachings of a system comprising clients communicating with each other through the server using messages over an Internet protocol (e.g., TCP/IP). *Server\ntserv\input.c* at line 195 and *Server\ntserv\socket.c* at line 688.

The prosecution history of the '523 patent, as discussed above, suggests that the aggregation elements of the claims were the allegedly novel aspects of the claims. Netrek presents a substantial new question that was not previously discussed or considered in the prosecution of the '523 patent. Specifically, Netrek teaches aggregating messages received from clients to send them together as an aggregated message (e.g., "Idea: read from client often, send to client not so often"). *Server\ntserv\input.c* at lines 152-203; *Server\ntserv\redraw.c* at lines 21-115; *Server\ntserv\socket.c* at lines 603-90. Therefore, a reasonable examiner would consider Netrek important in deciding the patentability of claims 1-6 and accordingly presents a SNQ. Further, the SNQ of Netrek is not cumulative of any of the technical teachings discussed or suggested during the prosecution history of the '523 patent.

**B. NETREK IN COMBINATION WITH MCFADDEN PRESENTS A SUBSTANTIAL NEW
QUESTION OF PATENTABILITY WITH RESPECT TO CLAIMS 1-6 OF THE '523 PATENT.**

Netrek was not cited or discussed during the prosecution of the '523 patent, and thus is new prior art with respect to the '523 patent. As discussed above, Netrek raises a SNQ with regards to claims 1-6. Further, McFadden is a FAQ and history of the online game Netrek. McFadden at p. 2, § 0.2; at p .8, §2.1.2 Architecture; and p. 11, §3.3.1 Client/Server. McFadden was specifically written to teach people interested in Netrek about the game. Accordingly,

people interested in playing and/or modifying Netrek code would have been motivated to review McFadden to gain an overview of the game.

The combination of Netrek and McFadden presents a substantial new question that was not previously discussed or considered in the prosecution of the '523 patent. Specifically, the combination of Netrek and McFadden teaches aggregating messages received from clients to send them together as an aggregated message (*e.g.*, “Idea: read from client often, send to client not so often”). *Server\ntserv\input.c* at lines 152-203; *Server\ntserv\redraw.c* at lines 21-115; *Server\ntserv\socket.c* at lines 603-90. Therefore, a reasonable examiner would consider the combination of Netrek and McFadden important in deciding the patentability of claims 1-6 and accordingly the combination presents a SNQ. Further, the SNQ raised by the combination of Netrek and McFadden is not cumulative of any of the technical teachings discussed or suggested during the prosecution history of the '523 patent.

C. VAN HOOK IN COMBINATION WITH DIS PRESENTS A SUBSTANTIAL NEW QUESTION OF PATENTABILITY WITH RESPECT TO CLAIMS 1, 2 AND 4-6 OF THE '523 PATENT

Van Hook was not cited or discussed during the prosecution of the '523 patent, and thus is new prior art with respect to the '523 patent. Van Hook in combination with DIS raises a SNQ with regards to claims 1, 2 and 4-6 because as discussed above, Van Hook discloses the technical teaching of aggregating group messages into a single packet by bundling the packets. Van Hook at 2. Van Hook states, “[t]he dominant effect of bundling is to reduce packet rates. Additionally, bundling reduces bit rates because fewer packet headers are sent.” *Id.* Further, Van Hook discloses techniques that have been developed and deployed for ARPA’s Synthetic Theater of War - Europe (“STOW-E”) program and Distributed Interactive Simulation (“DIS”), wherein a virtual world simulates battlefield conditions, and “[e]xplicit representations of command, control, and communication are required to permit command forces to transmit orders to and receive reports from a new generation of more intelligent semi-automated forces.” Van Hook at p. 1.

Likewise, as its name indicates, DIS is part of a proposed set of standards for the Distributed Interactive Simulation (“DIS”) used in conjunction with the STOW-E program described in Van Hook. DIS at p. 3. Van Hook provides for bundling of the PDUs from host computers by the Application Gateway (AG) server into larger transmission packets to be distributed to other packets. Van Hook at pp. 2 and 7. DIS goes one step further to discuss the

anatomy of a packet, as the PDU packets disclosed in DIS include a PDU header, an ID denoting a host computer (Entity ID), an exercise (Exercise ID), as well as a message group (Force ID), and the PDU message relating to positional information of the entity. DIS at pp. 40-41, Table 18.

The prosecution history, as discussed above, suggests that the aggregation elements of the claims were the allegedly novel aspect of the claimed invention. Van Hook provides a SNQ, as suggested above, because of its aggregation teachings. “Additionally, bundling reduces bit rates because fewer packet headers are sent.” *Id.* Accordingly, the combining the technical teachings (*e.g.*, teachings of bundling packets, or PDUs), in a Distributed Interactive Simulation, as disclosed in Van Hook, with the technical teachings of the contents of a PDU in a Distributed Interactive Simulation, as disclosed in DIS, raises an additional SNQ that was not discussed or cited in the prosecution history of the ‘523. A reasonable examiner would consider that combination important in deciding the patentability of claims 1, 2 and 4-6 since it would have been obvious to those skilled in the art to combine the teachings of the two references, as explicitly taught by Van Hook.

D. IRC RFC IN COMBINATION WITH FRIEDMAN PRESENTS A SUBSTANTIAL NEW QUESTION OF PATENTABILITY WITH RESPECT TO CLAIMS 1, 2 AND 4-6 OF THE ‘523 PATENT

IRC RFC discloses protocols for implementing Internet Relay Chat (IRC). IRC RFC was not cited or discussed during the prosecution of the ‘523 patent. “The main goal of IRC is to provide a forum which allows easy and efficient conferencing (one-to-many conversations).” IRC RFC at p. 11, § 3.2 One-to-many. IRC uses a client-server configuration, where a client sends a channel message to a server and the server distributes the message to the other clients who have joined that channel, as discussed in detail above.

IRC RFC does not disclose aggregating payload portions, but Friedman discloses that messages are buffered and then the payloads are aggregated (*e.g.*, packed) before sending. Friedman at p. 5. In addition, IRC RFC states that “[t]he main goal of IRC is to provide a forum which allows easy and efficient conferencing (one-to-many conversations).” IRC RFC at p. 11, § 3.2 One-to-many. Friedman discloses that the aggregation of message packets improves both latency and throughput compared to non-aggregating communication protocols. Friedman at p. 1.

The combination of RFC IRC and Friedman provides the technical teaching of aggregating the group messages of IRC RFC (*e.g.*, channel messages) in order to increase the efficiency of the network, which was a main goal of IRC RFC. The prosecution history of the '523 patent, as discussed above, suggests that the aggregation elements of the claims were the allegedly novel aspects of the claims in the '523 patent. IRC RFC alone does not provide the aggregation teachings; however, when IRC RFC is combined with Friedman, a substantial new question is presented that was not previously discussed or considered in the prosecution of the '523 patent. Further, a reasonable examiner would consider the teachings of IRC RFC in combination with Friedman because the teachings of aggregation are present and it would have been obvious to those skilled in the art to combine the teachings of the two references.

E. RING IN COMBINATION WITH NETREK PROVIDES A SUBSTANTIAL NEW QUESTION OF PATENTABILITY WITH RESPECT TO CLAIMS 1-6 OF THE '523 PATENT

RING was not cited or discussed during the prosecution of the '523 patent, and thus RING is new prior art with respect to the '523 patent. RING, as discussed above, presents a substantial new question of patentability alone. Similarly, Netrek discloses the technical teaching of clients and servers communicating over a network using messages. *See Server Code, Server\ntserv\newstartd.c* at lines 129-73, lines 179-311, lines 146-70; *Server\ntserv\main.c* at lines 135-43; *Server\ntserv\socket.c* at lines 442-88.

Netrek further discloses aggregating packets to reduce the number of packets sent from the server (*e.g.*, "Idea: read from client often, send to client not so often.") *Server\ntserv\input.c* at lines 152-203; *Server\ntserv\redraw.c* at lines 21-115; *Server\ntserv\socket.c* at lines 603-90. RING in combination with Netrek further raises a SNQ with regards to claims 1-6 because they provide the technical teachings of increasing network efficiency by applying the aggregation teachings of Netrek to aggregate messages in RING to increase network efficiency. Therefore, a reasonable examiner would consider the combination of RING and Netrek important in deciding the patentability of claims 1-6, and accordingly the combination presents a SNQ. Further, the SNQ of the combination RING and Netrek is not cumulative of any of the technical teachings discussed or suggested during the prosecution history of the '523 patent.

F. RING IN COMBINATION WITH VAN HOOK PROVIDES A SUBSTANTIAL NEW QUESTION OF PATENTABILITY WITH RESPECT TO CLAIMS 1, 2 AND 4-6 OF THE ‘523 PATENT

RING was not cited or discussed during the prosecution of the ‘523 patent, and thus is new prior art with respect to the ‘523 patent. RING, as discussed above, presents a substantial new question of patentability alone. Similarly, Van Hook discloses host computers and servers communicating over a network using messages packets. Van Hook at pp. 1, 2. RING in combination with Van Hook further raises a SNQ with regards to claims 1, 2 and 4-6 because the combination provides the technical teachings of increasing network efficiency by applying the aggregation teachings of Van Hook to aggregate messages in RING. Therefore, a reasonable examiner would consider the combination of RING and Van Hook important in deciding the patentability of claims 1, 2 and 4-6 and accordingly, this combination presents a SNQ. Further, the SNQ of the combination of RING and Van Hook is not cumulative to any of the technical teachings discussed or suggested during the prosecution history of the ‘523 patent.

V. DETAILED EXPLANATION UNDER 37 CFR 1.510(b) OF THE PERTINENCY AND MANNER OF APPLYING THE CITED PRIOR ART TO EVERY CLAIM FOR WHICH REEXAMINATION IS REQUESTED

A. CLAIMS 1-6 ARE ANTICIPATED BY NETREK UNDER 35 U.S.C. § 102

Please see the attached Exhibit CC-A presenting claim charts for comparison of Netrek with claims 1-6 of the ‘523 patent.

Server2.5pl4.tar.gz [hereinafter “Server Code”] and BRMH-1.7.tar.gz [hereinafter “Client Code”] contain the source code for the game Netrek. Together, the Server Code and Client Code define computer instructions for an online game based on a client-server network architecture. Specifically, the Server Code defines the computer instructions for the portion of the game running on the server; the Client Code defines the portion of the game running on the client or host computers. See, e.g., *Server\ntserv\socket.c* at lines 1390-1590 and lines 603-90; *brmh-1.7\socket.c* at lines 1537-1634; and *brmh-1.7\packets.h* at lines 523-29. McFadden is provided to add additional context and teaches inherent game play features of Netrek, in which clients connect to a server over the Internet, which allows players to “form into teams and fight for control of the galaxy, dogfighting and taking planets.” McFadden at p. 2, § 0.2; p. 8, §2.1.2 Architecture; and at p. 11, §3.3.1 Client/Server. Even though Netrek is presented as anticipating

the claims, McFadden is properly presented since it shows inherent characteristics of Netrek. See MPEP § 2131.01(III) and *Continental Can Co. USA v. Monsanto Co.*, 948 F.2d 1264, 1268 (“Normally, only one reference should be used in making a rejection under 35 U.S.C. § 102. However, a 35 U.S.C. § 102 rejection over multiple references has been held to be proper when extra references are cited to show that a characteristic not disclosed in the reference is inherent.”) To the extent that the examiner disagrees, Requester submits Netrek in view of McFadden to reject claims 1-6 under 35 U.S.C. § 103.

CLAIM 1

A method for providing group messages to a plurality of host computers connected over a unicast wide area communication network, comprising the steps of:

Netrek utilizes group messaging to send game state updates over the Internet, a unicast wide area network, to maintain a consistent and shared gaming experience among a number of host computers. McFadden at p. 2, § 0.2; at p. 8, §2.1.2 Architecture; and at p. 11, §3.3.1 Client/Server. Netrek also utilizes group messaging to allow players to communicate with other players in the game or players on a specific team. See Server Code; and *brmh-1.7\socket.c* at lines 1537-1634 (“*sendServerPacket(packet)*”).

providing a group messaging server coupled to said network, said server communicating with said plurality of host computers using said unicast network and maintaining a list of message groups, each message group containing at least one host computer;

The Netrek server is a group messaging server that is coupled to the Internet. See *Server Code*, *Server\ntserv\newstartd.c* at lines 129-73 (the server program *newstartd* loops while waiting for a network connection from a host computer), lines 179-311 (the function *connectionAttemptDetected* initializes the server’s network connection so that the server can listen for host computer connections), lines 146-70 (*newstartd* spawns a *ntserv* process on the server for each new host computer that connects); *Server\ntserv\main.c* at lines 135-43 (*ntserv* maintains the connection from the server to the host computer by calling *ConnectToClient*); *Server\ntserv\socket.c* at lines 442-88 (the function *ConnectToClient* defined).

The Netrek server communicates with the plurality of host computers using the Internet, a unicast network. See *Id.*, *Server\ntserv\newstartd.c* at lines 179-311 (a TCP/IP connection, socket type SOCK_STREAM, is created on the server to listen for incoming host computer

connections); *Server\ntserv\socket.c* at lines 442-88 (the function *ConnectToClient* maintains a TCP/IP connection, socket type SOCK_STREAM, between the server and each host computer), lines 1747-802 (the server communicates with a host computer by calling the function *flushSockBuf*, which calls the function *gwrite*), lines 2607-73 (the function *gwrite* defined). See generally *Id.*, *Server\ntserv\packets.h* (the header file defines all of the types of messages that can be sent during a Netrek game).

The Netrek server also maintains message groups in multiple aspects. Examples of message groups include the group of all host computers in the game, the group of host computers on a particular team, and the group of host computers in a player location (*i.e.* within the same proximity or geographic area) in the game. See *Id.*, *Server\ntserv\struct.h* at lines 471-82 (the server determines who is in a message group by examining the data structure *struct memory*, which contains an array of players: “*struct player players [MAXPLAYER];*”), lines 208-84 (definition for *struct player*, which includes a field for identifying the team/message group that the player is on, “*short p_team;*”, and fields for identifying the geographical vicinity (*i.e.*, group) of each player’s ship, “*int p_x;*” and “*int p_y;*”); *Server\ntserv\defs.h* at lines 120-134 (contains the definitions for the different teams and the group of all players); *Server\ntserv\socket.c* at lines 1125-92 (the function *updateTorps* determines whether or not a player is in a torpedo message’s proximity-based message group), lines 1194-255 (the function *updatePlasmas* determines whether or not a player is in a plasma message’s proximity-based message group), lines 1257-94 (the function *updatePhasers* determines whether or not a player is in a phaser message’s proximity-based message group), lines 1390-590 (the function *updateMessages* determines whether or not a player should receive a text message based on who the text message is addressed to).

Each team or message group on the Netrek server contains at least one host computer. See *Id.*, *Server\ntserv\main.c* at lines 183-325 (when a host computer joins a game, the server prompts the player to join or create a team, inserts the player into the array of players, and then initializes the player’s team field and location fields by calling *enter*); *Server\ntserv\enter.c* at lines 30-232 (definition to the function *enter*).

sending, by a plurality of host computers belonging to a first message group, messages to said server via said unicast network, said messages containing a payload portion and a portion for identifying said first message group;

Host computers, belonging to specific message groups, send out multiple types of messages to the Netrek server over the Internet, a unicast network. See *Client Code, brmh-1.7\socket.c* at lines 1537-634 (the function *sendServerPacket* sends messages from the host computer to the server using either TCP/IP or UDP/IP), *brmh-1.7\data.c* at line 26 (the player data structure contains a host computer's information about the message groups it belongs to), *brmh-1.7\struct.h* at lines 134-92 (*struct player* includes the fields “*short p_team;*”, “*int p_x;*”, and “*int p_y;*”, which identify the message groups a host computer belongs to). The messages a host computer sends out contain a payload portion and a portion for identifying a message group. See *Id., brmh-1.7\packets.h* (the header file defines all of the types of messages that can be sent by a host computer, for example, a torpedo message allows a user to fire a torpedo at another player). For example, when a host computer in Netrek sends a text message to a team/group (message type *CP_MESSAGE*), the message contains both a payload portion and a portion for identifying the message group. The payload portion is stored in the field “*char mesg[80];*”. See *Id., brmh-1.7\packets.h* at lines 523-29. The portion identifying a message group can be stored in, for example, stored in “*char group;*” and “*char indiv;*”. *Id.*

When a host computer in Netrek fires a torpedo, the host computer sends a torpedo message to the server. See *Id., brmh-1.7\defs.h* at line 222 (the function *sendTorpReq* sends the server a torpedo message). The host torpedo message sent to the server contains two fields: a field for storing the Netrek message type and a field for storing the direction of the torpedo. See *Id., brmh-1.7\packets.h* at lines 293-99 (“*char type;*” stores the message type and message type “*unsigned char dir;*” stores the direction). The fields storing the message type of the torpedo and direction of the torpedo represent the portion of the message that identifies the message group, and the field storing the direction of the torpedo represents the payload portion of the message.

When the server receives a host torpedo message, the message type field of the torpedo message directs the server to store the information for identifying the message group, as well as the payload portion of the message, into shared memory. See *Id., Server\ntserv\socket.c* at lines 121-97, 1976-2011, 2046-50; *Server\ntserv\torp.c* at lines 41-132 (the message type field of the torpedo causes the server to call the message handling function, *ntorp*, to store into shared memory the direction of the torpedo, which comes from the host torpedo message, and the X,Y coordinates of the torpedo, which is determined by the server based on the location of the host's

ship in the game); *Server\ntserv\daemonII.c* at lines 1161-1246 (the function *udtorps* regularly examines the shared memory and updates the locations of all torpedoes in the game using the direction and X, Y coordinates of each torpedo); *Server\ntserv\socket.c* at lines 1125-1192 (the aggregation function *updateTorps* uses the information stored in shared memory by the function *ntorp* to determine which players should receive which torpedo messages by comparing each torpedo's location with a host's ship location, *i.e.* proximity).

aggregating, by said server in a time interval determined in accordance with a predefined criterion, said payload portions of said messages to create an aggregated payload;

The Netrek server aggregates, in a time interval determined in accordance with predefined criterion, the payload portions of messages that are received from host computers to create an aggregated payload. *See Server Code, Server\ntserv\input.c* at line 195 (the function *input* calls the function *readFromClient* to receive messages sent by the host computers and then places the messages into shared memory (*e.g.*, “buf”) so that they can be aggregated by the server; *See also*, OTH-A, The Smith declaration at ¶¶ 7, 18, 25-39

For example, one of the comments states, “Idea: read from client often, send to client not so often”), lines 152-203 (the server sets the aggregation interval to a pre-defined time stored in *timerDelay*); *Server\ntserv\data.c* at line 76 (aggregation interval set to 200,000 microseconds, “int timerDelay=200000;”); *Server\ntserv\socket.c* at lines 603-90 (definition for the function *updateClient* that calls the other update functions which handle aggregation). *See generally Server Code, Server\ntserv\socket.c* (contains the update functions that handle aggregation).

For example, the Netrek server receives text messages, addressed to specific teams or all players, from the host computers and stores them into the server's shared memory. *See Server Code, Server\ntserv\input.c* at line 195 (the function *input* calls *readFromClient* to receive messages sent by the host computers and then places the messages into shared memory so that they can be aggregated by the server); *Server\ntserv\socket.c* at lines 1825-2044 (*readFromClient* calls *doRead*, which stores information into *buf* at line 1956). In Netrek, players can use text messages to communicate attack and defensive strategies or to make comments to the opposing teams, a player's own team, or all players. Because the groups of all players and teams each consist of multiple players, and players may send multiple messages simultaneously in the heat of battle, Netrek aggregates these multiple text messages with each

other, torpedo messages, and other types of Netrek messages during gameplay to make efficient use of the network and increase network throughput. The server, after waiting 0.2 seconds, calls the function *updateClient* for each host computer in the game. See *id.*, *Server\ntserv\input.c* at lines 52, 154-168, 197 (server sets the aggregation interval to a pre-defined time stored in *timerDelay*). The function *updateClient* calls multiple update functions, including the function *updateMessages*. See *id.*, *Server\ntserv\socket.c* at lines 603-90 (definition for the function *updateClient* that calls the other update functions which handle aggregation). The function *updateMessages* examines the server's shared memory and copies the appropriate text messages onto the aggregation buffer to create an aggregated payload intended for a target host computer. See *id.*, *Server\ntserv\socket.c* at lines 1390-590 (definition for the *updateMessages* function), lines 1603-744 (definition for the function *sendClientPacket*, which places individual messages onto the aggregation buffer). During gameplay, multiple ships in proximity of each other may fire multiple torpedoes at one another. Netrek aggregates these multiple torpedo messages with each other, text messages, and other types of Netrek messages during gameplay to make efficient use of the network and increase network throughput. When the Netrek server receives a torpedo message from a host computer, the server stores the message into its shared memory. See *Server Code*, *Server\ntserv\input.c* at line 195 (*input* calls the function *readFromClient* to receive messages sent by the host computers and then places the messages into shared memory so that they can be aggregated by the server); *Server\ntserv\socket.c* at lines 1825-2044 (*readFromClient* calls *doRead*, which stores information into *buf* at line 1956). The server, after waiting 0.2 seconds, calls the function *updateClient* for each host computer in the game. See *id.*, *Server\ntserv\input.c* at lines 52, 154-168, 197 (server sets the aggregation interval to a pre-defined time stored in *timerDelay*). The function *updateClient* calls multiple update functions, including *updateTorps*. See *id.*, *Server\ntserv\socket.c* at lines 603-90 (definition for the function *updateClient* that calls the other update functions which handle aggregation). The function *updateTorps* examines the server's shared memory and copies the appropriate torpedo messages, based on proximity, onto the aggregation buffer to create an aggregated payload intended for a target host computer. See *id.*, *Server\ntserv\socket.c* at lines 1125-92 (definition for the *updateTorps* function), lines 1603-744 (definition for the function *sendClientPacket*, which places individual messages onto the aggregation buffer).

forming an aggregated message using said aggregated payload; and

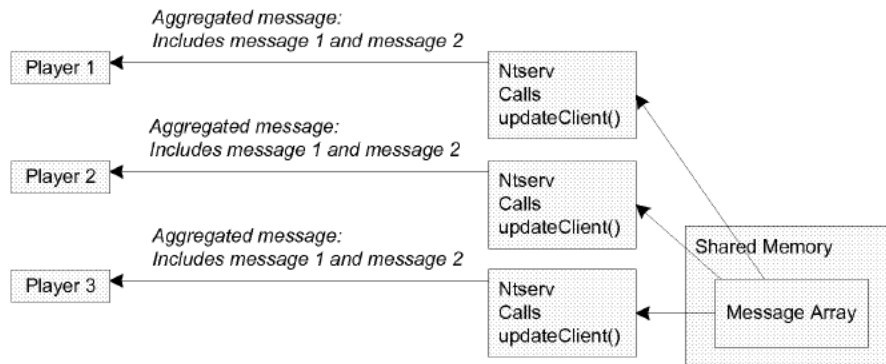
The Netrek server forms an aggregated message using the aggregated payload that was created in the aggregation buffer. See *Server Code, Server\ntserv\socket.c* at line 688 (the function *updateClient* calls the function *flushSockBuf* after filling the aggregation buffer to form and transmit an aggregated message using the TCP/IP or UDP/IP protocol), lines 1603-744 (if the aggregation buffer becomes full, the function *sendClientPacket* will call the function *gwrite* to form and transmit an aggregated message using the TCP/IP or UDP/IP protocol), lines 1747-802 (definition of the function *flushSockBuf*), lines 2607-73 (definition of the function *gwrite*). An example of such an aggregated message appears in Figure 6c of the Smith declaration:

Netrek server's IP address	Player 3's IP address	Server message packet 1	Server message packet 2
----------------------------	-----------------------	-------------------------	-------------------------

Smith declaration at Figure 6c.

“**Figure 6c.** The aggregated message sent to Player 3 included Internet header information and both messages. Server message packet 1 was based on the payload of the message from Player 1 and server message packet 2 was based on the payload of the message from Player 2.” Smith declaration at p. 18.

transmitting, by said server via said unicast network, said aggregated message to a recipient host computer belonging to said first message group.



Smith declaration at Figure 6b.

“**Figure 6b.** Each host's *ntserv* process called *updateClient()*, which in turn called *updateMessages()*. *UpdateMessages()* found all messages intended for that host in the message array, aggregated them into a buffer (not shown), and transmitted the buffer contents to the host. In this example, Players 1-3 are on the same team, Players 1 and 2 had earlier indicated that their messages (labeled message1 and message2 in Figure 6a above) should be sent to their entire team. Thus, Players 1-3 all received an aggregated message that included both messages.” *Id.*

The Netrek server transmits the aggregated message via the unicast network to the host computers belonging to the appropriate message groups. See *Server Code, Server\ntserv\socket.c* at line 688 (the function *updateClient* calls the function *flushSockBuf* after filling the aggregation buffer to form and transmit an aggregated message using the TCP/IP or UDP/IP protocol), lines 1603-744 (if the aggregation buffer becomes full, the function *sendClientPacket* will call the function *gwrite* to form and transmit an aggregated message using the TCP/IP or UDP/IP protocol), lines 1747-802 (definition of the function *flushSockBuf*), lines 2607-73 (definition of the function *gwrite*). As an example, a series of torpedo messages can be collected and sent (e.g., fired) to a competing player in the game. *Server\ntserv\socket.c* at lines 1125-92. In the case that all messages aggregated during the aggregation interval belong to the same message group, every computer belonging to the message group will receive the same message because only group messages have been aggregated. Therefore, Netrek anticipates transmitting the same “aggregated message” to each group member as recited by the claim.

CLAIM 2

The method of claim 1 wherein said time interval is a fixed period of time.

The Netrek server called its aggregation function, *updateClient*, every 0.2 seconds. See *Server Code, Server\ntserv\input.c* at lines 152-168 (server sets the aggregation interval to a pre-defined time stored in *timerDelay*); *Server\ntserv\data.c* at line 76 (aggregation interval set to 200,000 microseconds, “int timerDelay=200000;”).

CLAIM 3

The method of claim 1 wherein said time interval corresponds to a time for said server to receive at least one message from each host computer belonging to said first message group.

Netrek discloses a *readFromClient* function that receives messages from each of the hosts (e.g., clients) into a shared memory (e.g., “buf”) for aggregation. See *Server\ntserv\input.c* at line 195 and *Server\ntserv\socket.c* at lines 1825-2044. Netrek further discloses a time for the server to receive at least one message from each host—each host that joins has a corresponding *ntserv* process running on the server, which waits for 0.2 seconds for at least one message to aggregate in *buf* from each host computer belonging to the first message group (e.g., all players in the game or on a particular team)—and after waiting the 0.2 seconds, the aggregated messages

are sent to the clients. See *Server\ntserv\socket.c* at lines 1825-2044, 603-90; *Server\ntserv\input.c* at lines 152-203; *Server\ntserv\redraw.c* at lines 21-115.

CLAIM 4

The method of claim 1 further comprising the step of creating, by one of said plurality of host computers, said first message group by sending a first control message to said server via said unicast network.

Host computers in Netrek create message groups by sending create messages to the server. Specifically, the first player to join a team, or the first player to join the game sends a create message. See *Client Code, brmh-1.7\entrywin.c* at lines 57-353 (host computer prompts user to select a team to create); *brmh-1.7\socket.c* at lines 1800-09 (host computer sends a create message to the server specifying the team that the host wants to create, “*outfitReq.team = team;*”).

CLAIM 5

The method of claim 4 further comprising the step of joining, by some of said plurality of host computers, said first message group by sending control messages via said unicast network to said server specifying said first message group.

Host computers in Netrek send join messages to the server specifying a message group to be joined. See *Client Code, brmh-1.7\entrywin.c* at lines 57-353 (host computer prompts user to select a team to join); *brmh-1.7\socket.c* at lines 1800-09 (host computer sends a join message to the server specifying the team that the host wants to join, “*outfitReq.team = team;*”).

CLAIM 6

The method of claim 1 wherein said network is Internet and said server communicates with said plurality of host computers using a session layer protocol.

Netrek is a network game that runs over the Internet. See *Id.*, *Server\ntserv\newstartd.c* at lines 179-311 (a TCP/IP connection, socket type SOCK_STREAM, is created on server to listen for incoming host computer connections); *Server\ntserv\socket.c* at lines 442-88 (the function *ConnectToClient* maintains a TCP/IP connection, socket type SOCK_STREAM, between the server and each host computer); *Server\packets.h* (the header file defines the packet types that exist in the Netrek session layer protocol). As was known in the art, TCP/IP

connections implement session layer functionality in the transport layer and/or application layer, which means that Netrek inherently includes a session layer functionality.

B. CLAIMS 1-6 ARE RENDERED OBVIOUS BY NETREK IN VIEW OF MCFADDEN UNDER 35 U.S.C. § 103

Please see the attached Exhibit CC-B presenting claim charts for comparison of Netrek and McFadden with claims 1-6 of the '523 patent.

To the extent that the Office finds that the “BRMH-1.7.tar.gz” Client Code, the “Server2.5pl4.tar.gz” Server Code and McFadden does not teach inherent characteristics of Netrek, Requester submits that it would have been obvious to combine “BRMH-1.7.tar.gz” Client Code, the “Server2.5pl4.tar.gz” Server Code (together “Netrek”) in view of McFadden to the claims 1-6 of the '523 patent under 35 U.S.C. § 103.

Requester incorporates by reference the arguments made above in § IV-A to show that even if claim 1-6 of the '523 patent are not anticipated by Netrek, they are rendered obvious by Netrek in view of McFadden. As discussed above, Netrek discloses all of the elements of the claims under 35 U.S.C. § 102, particularly when the inherent features of Netrek shown by McFadden are considered. Beyond the Netrek disclosure and its inherent characteristics, Netrek and McFadden when considered together render the claims obvious under 35 U.S.C. § 103 for the reasons given below.

Reason to Combine:

“BRMH-1.7.tar.gz” Client Code and the “Server2.5pl4.tar.gz” Server Code are both used to play Netrek. See OTH-C, Ahn Declaration at ¶ 5. Therefore, one of skill in the art would look to combine the Client Code and the Server Code in order to enable playing Netrek.

McFadden is a FAQ and history of the online game Netrek game. McFadden at p. 2, § 0.2; p. 8, §2.1.2 Architecture; and p. 11, §3.3.1 Client/Server. McFadden was specifically written to teach people interested in Netrek about the game. Accordingly, one of ordinary skill in the art would look to McFadden to help provide context of the source code of Netrek in order to understand how certain lines of code affected actual gameplay experience.

C. CLAIMS 1, 2 AND 4-6 ARE RENDERED OBVIOUS BY VAN HOOK IN VIEW OF DIS UNDER 35 U.S.C. § 103

Please see the attached Exhibit CC-C presenting claim charts for comparison of Van Hook in view of DIS with claims 1, 2 and 4-6 of the '523 patent.

Reason to Combine:

Van Hook discloses techniques that have been developed and deployed for ARPA's Synthetic Theater of War - Europe ("STOW-E") program and Distributed Interactive Simulation ("DIS"), wherein a virtual world simulates battlefield conditions, and "[e]xplicit representations of command, control, and communication are required to permit command forces to transmit orders to and receive reports from a new generation of more intelligent semi-automated forces." Van Hook at p. 1. Likewise, as its name indicates, DIS is part of a proposed set of standards for the Distributed Interactive Simulation ("DIS") used in conjunction with the STOW-E program in Van Hook. DIS at pp. 1-3. Van Hook provides for bundling of the PDUs from host computers by the AG server into larger transmission packets to be distributed to other host computers. Van Hook at pp. 2 and 7. DIS goes one step further to discuss the anatomy of a packet, as the PDU packets disclosed in DIS include a PDU header, an ID denoting a host computer (Entity ID), an ID for an exercise (Exercise ID), an ID denoting which team the host computer belongs to (Force ID) and the positional information of the entity. DIS at pp. 40-41, Table 18. It would have been obvious to a person of ordinary skill in that the art to combine the teachings of bundling packets, or PDUs, in a Distributed Interactive Simulation disclosed in Van Hook with the teachings of the contents of a PDU in a Distributed Interactive Simulation as disclosed in DIS since Van Hook explicitly teaches using the DIS protocol to exchange information in STOW-E.

CLAIM 1

A method for providing group messages to a plurality of host computers connected over a unicast wide area communication network, comprising the steps of:

Van Hook in view of DIS discloses a method for providing group messages, such as the protocol data units ("Data messages, called protocol data units (PDUs)") disclosed in DIS, to a plurality of host computers connected over a wide area network ("WAN"). Van Hook at pp. 1, 4, 5 and Figure 1; DIS at Abstract, pp. 3, 4, 5 and 10. Van Hook discloses "some of the innovative techniques being developed and deployed" for the Synthetic Theater of War-Europe ("STOW-E") exercise. Van Hook at p. 1. STOW-E would use the Distributed Interactive Simulation ("DIS") protocols to exchange information between DIS-based simulators. Thus, Van Hook discloses providing group messages (*e.g.*, "Data messages, called protocol data units (PDUs)") to a plurality of host computers (*e.g.*, "network sites") connected over a wide area communication network (*e.g.*, the "Defense Simulation Internet (DSI) Wide Area Network

(WAN)"). DIS discloses providing group messages (*e.g.*, "Data messages, called protocol data units (PDUs)") to a plurality of "host computers" over a wide area network. DIS at Abstract, pp. 3, 4, 5 and 10.

Van Hook does not expressly disclose that the wide area communication network is unicast. Instead, Van Hook discloses that the LANs run in "promiscuous mode" (*i.e.*, broadcast) and the WAN is multicast. Van Hook at 5. However, DIS was designed to operate over a unicast network. DIS at 5, 10 ("*Delivery*. The communication architecture must support either, multicast, broadcast, or unicast packets."). Thus, a person of ordinary skill in the art at the time of filing would have found it obvious to modify the system disclosed in Van Hook and DIS to operate on a unicast network.

providing a group messaging server coupled to said network, said server communicating with said plurality of host computers using said unicast network and maintaining a list of message groups, each message group containing at least one host computer;

Van Hook in view of DIS discloses providing a group messaging server (*e.g.*, the "Application Gateway" ("AG")) coupled to said network, said server communicating with said plurality of host computers participating in the DIS exercise, using said network. Van Hook at Figs. 1 and 5; pp. 1, 4, 6; DIS at p. 36. Van Hook further discloses that each AG maintains a list of message groups (*e.g.*, "cell sets," "forces," or "exercises") each message group containing at least one host computer. Van Hook at Figs. 1, 5, pp. 1, 4 and 6.

sending, by a plurality of host computers belonging to a first message group, messages to said server via said unicast network, said messages containing a payload portion and a portion for identifying said first message group;

Van Hook in view of DIS discloses sending, by a plurality of host computers belonging to a first message group, messages (*e.g.*, PDUs) to said server via said network, said messages containing a payload portion and a portion for identifying said first message group (*e.g.*, first coordinates). Van Hook at pp. 2 and 5; DIS at Table 18, pp. 5, 14, 36 and 39-41. For example, the payload portion of the PDU can be "state information that is necessary for the receiving simulation application to represent the issuing entity in the simulation application's own simulation"). DIS at p. 14. The portion for identifying said first message group can be, for example, the positional information (*i.e.*, coordinates), exercise information "Exercise ID", or force information "Force ID" in the PDU. While the coordinates themselves do not indicate a

particular message group, they are used by the AGs “for identifying” the group (*e.g.*, “cell set,” Force, or Exercise) to which the PDU should be transmitted. Van Hook at 6.

aggregating, by said server in a time interval determined in accordance with a predefined criterion, said payload portions of said messages to create an aggregated payload;

Van Hook discloses aggregating (*e.g.*, bundling), by said server (*e.g.*, AG) in a time interval determined in accordance with a predefined criterion (*e.g.*, “maximum delay time”), said payload portions of said messages to create an aggregated payload. Van Hook at pp. 2 and 7.

forming an aggregated message using said aggregated payload; and

Van Hook discloses forming an aggregated message (*e.g.*, “bundled packets”) using said aggregated payload. *Id.*

transmitting, by said server via said unicast network, said aggregated message to a recipient host computer belonging to said first message group.

Van Hook, in view of DIS, discloses transmitting, by said server (*e.g.*, “AG”) via said network, (*e.g.*, “WAN”) said aggregated message (*e.g.*, “bundled PDU”) onto the WAN. Van Hook at 7. The other AGs on the WAN receive the aggregated message (*e.g.*, “bundled packet”), unbundle it, and determine which hosts in the group (*e.g.*, “cell set,” “Force ID” or “Exercise ID”) should receive the PDU. The AG then transmits the PDUs individually to those recipient host computer in the group (*e.g.*, “cell set,” “force,” or “exercise”). Van Hook at Figures 1 and 5; pp. 1, 2, 4, 6, 7; DIS at p. 5.

The recipient host computer does not receive the aggregated message (*e.g.*, “bundled PDU”) because it is unbundled by an AG after being received from the WAN and before being retransmitted to the host computer over the LAN. Van Hook at 7, section 4.6. Nevertheless, Requester submits that the broadest reasonable interpretation of this element does not require receiving, by a recipient host computer, said aggregated message. Instead, the step of “transmitting ... said aggregated message” is performed when the AG transmits the bundled PDU out onto the WAN, even though the packet may be de-aggregated prior to being received by the recipient host computer.

CLAIM 2

The method of claim 1 wherein said time interval is a fixed period of time.

Van Hook discloses the method of claim 1 wherein said time interval is a fixed period of time (*e.g.*, the maximum time delay). Van Hook at pp. 2 and 7.

CLAIM 4

The method of claim 1 further comprising the step of creating, by one of said plurality of host computers, said first message group by sending a first control message to said server via said unicast network.

Van Hook in view of DIS discloses creating a message group (*e.g.*, “cell set,” “Force” or “Exercise”) by establishing the group at initialization or during the simulation, when, for example, a participant enters a region of the simulation or establishes a new force Van Hook at 6 and DIS at 36.

CLAIM 5

The method of claim 4 further comprising the step of joining, by some of said plurality of host computers, said first message group by sending control messages via said unicast network to said server specifying said first message group.

Van Hook in view of DIS discloses some host computers (*e.g.*, participants in the same vicinity, exercise or force) joining a message group (*e.g.*, “cell set,” “Force” or “Exercise”) by sending a control message (*e.g.*, join PDU or moving into the vicinity of other group members) specifying the message group (*e.g.*, coordinates, “Exercise” or “Force”). Van Hook at 6 and DIS at 36. Join PDUs are inherent in the operation of STOW-E because there must be some method for multiple computers to join the simulation during initialization or while the game is in progress; this is exemplified in Macedonia, which discloses Join PDUs in a DIS system. Macedonia at 42. Therefore, Van Hook, in view of DIS, inherently, or at least obviously, includes Join PDUs.

CLAIM 6

The method of claim 1 wherein said network is Internet and said server communicates with said plurality of host computers using a session layer protocol.

Van Hook discloses the method of claim 1 wherein the network is the Defense Simulation Internet Wide Area Network, and the AG server communicates with said plurality of

host computers using a session layer protocol (*e.g.*, the “DIS protocol”, which inherently runs over a session layer protocol or its equivalent). Van Hook at p. 1.

**D. CLAIMS 1, 2 AND 4-6 ARE RENDERED OBVIOUS BY IRC RFC IN VIEW OF
FRIEDMAN UNDER 35 U.S.C. § 103**

Please see the attached Exhibit CC-D presenting claim charts for comparison of IRC RFC in view of Friedman with claims 1, 2 and 4-6 of the ‘523 patent.

Reason to Combine:

IRC RFC does not disclose aggregating payload portions, but Friedman discloses that messages are buffered and then the payloads are aggregated (*e.g.*, packed) before sending. Friedman at p. 5. In addition, IRC RFC states that “[t]he main goal of IRC is to provide a forum which allows easy and efficient conferencing (one-to-many conversations).” IRC RFC at p. 11, § 3.2, One-to-many. Friedman discloses that the aggregation of message packets improves both latency and throughput compared to non-aggregating communication protocols. Friedman at p. 1. In addition both IRC RFC and Friedman are both directed to messaging groups connecting over a network. IRC RFC at p. 11, § 1, Introduction and Friedman at pg. 2. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to aggregate the group messages of IRC RFC (*e.g.*, channel messages) in order to increase the efficiency of the network, which was a main goal of IRC RFC.

CLAIM 1

A method for providing group messages to a plurality of host computers connected over a unicast wide area communication network, comprising the steps of:

IRC RFC discloses a text-based protocol designed to provide real-time Internet text messaging or synchronous text-based conferencing through the use of a client-server model. IRC RFC provides a method for providing group messages (*e.g.*, chat messages on a specific channel) to a plurality of host computers (*e.g.*, clients). IRC RFC at p. 4, §1 Introduction and p. 11, §3.2.2, To a group. The host computers are connected over a unicast wide area communication network, the Internet, and communicate with one another via TCP/IP. IRC RFC at abstract, § 1. Messages “are sent only once to that server [connected to the client] and then sent to each client on the channel” rather than sent directly to multiple other clients. *Id.* at § 3.2.2.

providing a group messaging server coupled to said network, said server communicating with said plurality of host computers using said unicast network and maintaining a list of message groups, each message group containing at least one host computer;

IRC RFC discloses a group messaging server (*e.g.*, the server) coupled to a network (*e.g.*, the Internet), said server, communicating with a plurality of host computers (*e.g.*, clients) using a unicast network, and maintaining a list of message groups (*e.g.*, channels). IRC RFC at p. 4, §1, Introduction, 5, §, 1.3, Channels and 11, § 3.2.2, To a group (channel). Each server “know[s] about all channels, their inhabitants and properties.” *Id.* at p. 64, § 9.2.2, Channels.

sending, by a plurality of host computers belonging to a first message group, messages to said server via said unicast network, said messages containing a payload portion and a portion for identifying said first message group;

IRC RFC discloses that clients can send a message to a message group (*e.g.*, a channel). IRC RFC at p. 32, § 4.4.1, Private messages. The messages contain a payload portion, *i.e.* <text to be sent>, and a portion identifying the message group (*e.g.*, <receiver> which can be a channel name). *Id.*

aggregating, by said server in a time interval determined in accordance with a predefined criterion, said payload portions of said messages to create an aggregated payload;

IRC RFC discloses that sending individual messages to each user in a list is the least efficient method of group communication because duplicate messages are sent along the same path. IRC RFC at p. 11, § 3.2.1, To a list. IRC instead suggests sending the message to a message group (*i.e.* a channel) such that “the message text is sent only once to that server and then sent to each client on the channel.” IRC RFC at § 3.2.2. IRC RFC does not disclose aggregating payload portions, but Friedman discloses such. Friedman discloses that messages are buffered and then the payloads are aggregated (*e.g.*, packed) before sending. Friedman at p. 5. It would have been obvious to aggregate the group messages of IRC RFC (*e.g.*, channel messages) in order to increase the efficiency of the network. Friedman at p. 1.

forming an aggregated message using said aggregated payload; and

Friedman discloses that aggregated payloads (*e.g.*, packed messages) are formed into an aggregated message (*e.g.*, packed message having a single header). Friedman at pp. 5 and 12.

transmitting, by said server via said unicast network, said aggregated message to a recipient host computer belonging to said first message group.

IRC RFC discloses that the server sends messages addressed to a channel to each other host (e.g., client), which is a member of the message group (e.g., channel). IRC RFC at p. 11, § 3.2.2, To a group.

CLAIM 2

The method of claim 1 wherein said time interval is a fixed period of time.

Friedman discloses that messages are buffered for a time interval that is fixed (e.g., one millisecond). Friedman at p. 5.

CLAIM 4

The method of claim 1 further comprising the step of creating, by one of said plurality of host computers, said first message group by sending a first control message to said server via said unicast network.

IRC RFC discloses creating a message group (e.g., a channel) by sending a control message (e.g., the “JOIN” command creates a new channel when the first client joins) with the channel name. IRC RFC at p. 5, §1.3 Channels and 19, §4.2.1, Join message.

CLAIM 5

The method of claim 4 further comprising the step of joining, by some of said plurality of host computers, said first message group by sending control messages via said unicast network to said server specifying said first message group.

IRC RFC discloses some host computers (e.g., some clients) joining a message group (e.g., a channel) by sending a control message (e.g., the “JOIN” command) with the channel name. IRC RFC at p. 5, §1.3 Channels, and 19, §4.2.1, Join message.

CLAIM 6

The method of claim 1 wherein said network is Internet and said server communicates with said plurality of host computers using a session layer protocol.

IRC RFC discloses running on systems using the TCP/IP network protocol suite, which necessarily includes communication using a session layer protocol. IRC RFC at p. 4, § 1 Introduction. As was known in the art, TCP/IP connections implement session layer

functionality in the transport layer and/or application layer, which means that IRC RFC inherently includes a session layer functionality.

E. CLAIMS 1-6 ARE RENDERED OBVIOUS BY RING IN VIEW OF NETREK UNDER 35 U.S.C. § 103

Please see the attached Exhibit CC-E presenting claim charts for comparison of RING in view of Netrek with claims 1-6 of the '523 patent.

Reasons to Combine:

RING discloses communicating messages over a network. RING at Figs. 5 and 7, pp. 88, 87 and 91. Similarly, Netrek discloses clients and servers communicating over a network using messages. *See Server Code, Server\ntserv\newstartd.c* at lines 129-73, lines 179-311, lines 146-70; *Server\ntserv\main.c* at lines 135-43; *Server\ntserv\socket.c* at lines 442-88. Netrek further discloses aggregating packets to reduce the number of packets sent from the server. (*e.g.*, “Idea: read from client often, send to client not so often”). *Server\ntserv\input.c* at lines 152-203; *Server\ntserv\redraw.c* at lines 21-115; *Server\ntserv\socket.c* at lines 603-90. A person of ordinary skill in the art, looking to increase network efficiency, would have looked to related methods of communicating messages over a network. Accordingly, a person of ordinary skill in the art would have looked to the aggregation teachings of Netrek to aggregate messages in RING to increase network efficiency.

CLAIM 1

A method for providing group messages to a plurality of host computers connected over a unicast wide area communication network, comprising the steps of:

RING discloses a method for providing group messages (*e.g.*, “update messages”) to a plurality of host computers (*e.g.*, “client workstations”) connected over RING’s unicast wide-area communication network. RING at Abstract, pp. 85, 86, 90 and 91.

providing a group messaging server coupled to said network, said server communicating with said plurality of host computers using said unicast network and maintaining a list of message groups, each message group containing at least one host computer;

RING discloses providing a group messaging server coupled to the network, (*e.g.*, RING’s unicast wide-area communication network), wherein the server communicates with the

plurality of host computers (*e.g.*, “client workstations”) using the unicast network and maintaining a list of message groups. RING at Figs. 5 and 7, pp. 88, 87 and 91. As illustrated in Figure 7 (reproduced below), for example, RING discloses that clients A and C belong to client B’s message group, and therefore this particular message group contains at least one host computer, or client workstation, including A, B and C. RING at Fig. 7.

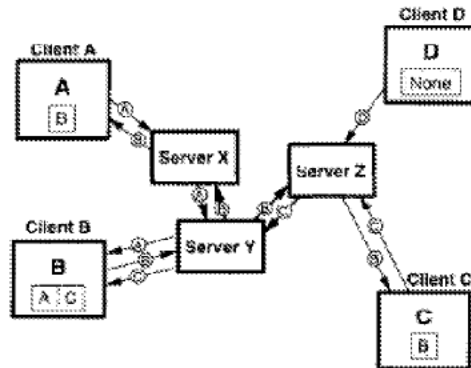


Figure 7: Flow of update messages (labeled arrows) for updates to entities A, B, C, and D arranged in a virtual environment as shown in Figure 4.

Figure 7 of RING at p. 88.

Message groups can consist of all clients connected to RING servers, or clients that are visible to each other and can send messages to each other. RING at pp.87-88. A server, such as server Y in Figure 7, maintains a list of message groups, as “servers keep track of which cells contain which entities by exchanging ‘periodic’ update messages when entities cross cell boundaries,” and thus become visible to other clients. RING at p. 87.

sending, by a plurality of host computers belonging to a first message group, messages to said server via said unicast network, said messages containing a payload portion and a portion for identifying said first message group;

RING discloses sending, by a plurality of host computers (*e.g.*, “client workstations”) belonging to a first message group (*e.g.*, other clients participating in the same distributed simulation and in the same cell), messages to the server via the unicast network. RING at pp. 87 and 91. The messages (*e.g.*, “update messages”) contain 40 bytes, and consist of a portion for identifying a first message group, such as an “entity-ID,” as well as a payload portion containing message information such as “target-position,” “target-orientation,” “positional-velocity,” and “rotational velocity.” RING at pp. 87, 89, 91 and Fig. 5. While the entity-ID does not explicitly indicate a particular message group, it is used by the server “for identifying” the group (*e.g.*,

“cell”) to which the message should be transmitted. RING at p. 87 (“[S]ervers keep track of which cells contains which entities by exchanging ‘periodic’ update message when entities cross cell boundaries. Real-time update messages are propagated only to servers and client containing entities inside some cell visible to the one containing the updated entity.”)

aggregating, by said server in a time interval determined in accordance with a predefined criterion, said payload portions of said messages to create an aggregated payload;

While RING does not explicitly disclose aggregating, Netrek discloses aggregating payload portions of said messages to create an aggregated payload. *See Server Code, Server\ntserv\input.c* at line 195 (the function *input* calls the function *readFromClient* to receive messages sent by the host computers and then places the messages into shared memory (e.g., “buf”) so that they can be aggregated by the server; *See also*, OTH-A, The Smith declaration at ¶¶ 7, 18, 25-39

For example, one of the comments states, “Idea: read from client often, send to client not so often”), lines 152-203 (the server sets the aggregation interval to a pre-defined time stored in *timerDelay*); *Server\ntserv\data.c* at line 76 (aggregation interval set to 200,000 microseconds, “int timerDelay=200000;”); *Server\ntserv\socket.c* at lines 603-90 (definition for the function *updateClient* that calls the other update functions which handle aggregation). *See generally Server Code, Server\ntserv\socket.c* (contains the update functions that handle aggregation).

For example, the Netrek server aggregates torpedo messages based on a ship’s proximity to a torpedo because ships in the torpedo’s proximity may be hit by it and players will need to see it to take evasive measures. When the Netrek server receives a torpedo message from a host computer, the server stores the message into its shared memory (e.g., “buf”). *See Server Code, Server\ntserv\input.c* at line 195 (*input* calls the function *readFromClient* to receive messages sent by the host computers and then places the messages into shared memory (e.g., buf) so that they can be aggregated by the server); *Server\ntserv\socket.c* at lines 1825-2044 (*readFromClient* calls *doRead*, which stores information into buf at line 1956).

Similarly, RING takes entity proximity into account when processing what information should be sent to servers and clients. RING at p. 87. Therefore, it would have been obvious to one of skill in the art to use the teaching of aggregating message payloads from clients based on

entity proximity in Netrek to aggregate update message payloads in RING, which are also based on entity proximity, to increase network efficiency.

forming an aggregated message using said aggregated payload; and

While RING does not explicitly disclose forming an aggregated message, it teaches that it is advantageous to aggregate (*e.g.*, “augment”) a client message payload (*e.g.*, an “update message”) with “Add” and “Remove” messages. RING at p. 88 (“As entities move through the environment, servers augment update messages with ‘Add’ and ‘Remove’ messages notifying clients that remote entities have become relevant or irrelevant to the client’s local entities.”). Moreover, Netrek teaches forming an aggregated message using the aggregated payload that was created in the aggregation buffer. *See Server Code, Server\ntserv\socket.c* at line 688 (the function *updateClient* calls the function *flushSockBuf* after filling the aggregation buffer to form and transmit an aggregated message using the TCP/IP or UDP/IP protocol), lines 1603-744 (if the aggregation buffer becomes full, the function *sendClientPacket* will call the function *gwrite* to form and transmit an aggregated message using the TCP/IP or UDP/IP protocol), lines 1747-802 (definition of the function *flushSockBuf*), lines 2607-73 (definition of the function *gwrite*). An example of such an aggregated message appears in Figure 6c of the Smith declaration:

Netrek server’s IP address	Player 3’s IP address	Server message packet 1	Server message packet 2
-------------------------------	--------------------------	----------------------------	----------------------------

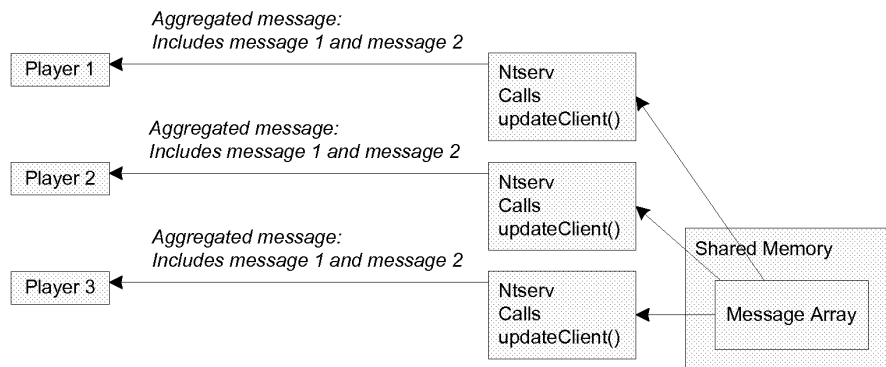
“**Figure 6c.** The aggregated message sent to Player 3 included Internet header information and both messages. Server message packet 1 was based on the payload of the message from Player 1 and server message packet 2 was based on the payload of the message from Player 2.” Smith declaration at p. 18.

transmitting, by said server via said unicast network, said aggregated message to a recipient host computer belonging to said first message group.

RING further discloses transmitting, by said server via the unicast network, said aggregated message to a recipient host computer (*e.g.*, “client workstation”) belonging to said first message group. RING at pp. 87 and 91. More specifically, RING teaches, “sending messages directly between clients, RING routes each one through at least one server and possibly two.” RING at p. 88. According to RING, client workstations belong to the first

message group if they participate in the same distributed simulation or are visible to each other. RING at p. 87.

Moreover, the Netrek server transmits the aggregated message via the unicast network to the host computers belonging to the appropriate message groups. *See Server Code, Server\ntserv\socket.c* at line 688 (the function *updateClient* calls the function *flushSockBuf* after filling the aggregation buffer to form and transmit an aggregated message using the TCP/IP or UDP/IP protocol), lines 1603-744 (if the aggregation buffer becomes full, the function *sendClientPacket* will call the function *gwrite* to form and transmit an aggregated message using the TCP/IP or UDP/IP protocol), lines 1747-802 (definition of the function *flushSockBuf*), lines 2607-73 (definition of the function *gwrite*). As an example, a series of torpedo messages can be collected and sent (*e.g.*, fired) to a competing player in the game. *Server\ntserv\socket.c* at lines 1125-92.



Smith declaration at Figure 6b.

“**Figure 6b.** Each host's *ntserv* process called *updateClient()*, which in turn called *updateMessages()*. *UpdateMessages()* found all messages intended for that host in the message array, aggregated them into a buffer (not shown), and transmitted the buffer contents to the host. In this example, Players 1-3 are on the same team, Players 1 and 2 had earlier indicated that their messages (labeled message1 and message2 in Figure 6a above) should be sent to their entire team. Thus, Players 1-3 all received an aggregated message that included both messages.” *Id.*

CLAIM 2

The method of claim 1 wherein said time interval is a fixed period of time.

RING, in view of Netrek, discloses the method of claim 1 wherein said time interval is a fixed period of time. RING at Abstract, pp. 85, 86, 87, 90 and 91. In particular, RING discloses