

EXHIBIT 1

**THIS EXHIBIT HAS BEEN
REDACTED IN ITS ENTIRETY**

EXHIBIT 2

**IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE**

ACCELERATION BAY LLC,)
)
 Plaintiff,)
) C.A. No. 16-455 (RGA)
 v.)
)
 TAKE-TWO INTERACTIVE SOFTWARE,)
 INC., ROCKSTAR GAMES, INC., and 2K)
 SPORTS, INC., Delaware Corporations,)
)
 Defendants.)
_____)

**EXPERT REPORT OF PATRICK CONLIN REGARDING TESTING
OF TAKE-TWO INTERACTIVE SOFTWARE ACCUSED PRODUCTS**

TABLE OF CONTENTS

1. Summary of Opinions1

2. Experience and Qualifications1

 a) Curriculum Vitae1

 b) Prior Testimony2

 c) Compensation2

3. Materials Considered2

4. Demonstratives3

5. Testing Methodology3

6. Assumptions and Testing Background4

7. Testing of Grand Theft Auto V Online.....6

I, Patrick G. Conlin, have been asked by Plaintiff Acceleration Bay, LLC (“Acceleration Bay”) to conduct testing of certain accused products and to testify as an expert witness in the above referenced action. I expect to testify at trial in this action regarding the opinions set forth in this report (the “Report”), as well as on any other issues for which I have submitted or will submit an expert report in this action.

1. Summary of Opinions

1. Based on tests that I conducted, it is my opinion that Grand Theft Auto V Online (“GTA”):

- uses direct peer-to-peer connections to exchange data between player consoles;
- includes players outside the United States in gameplay sessions based in the United States;
- uses a handshake protocol to establish direct peer-to-peer connections between player consoles;
- uses a handshake protocol which includes a unique ID for every player in the session;
- limits the direct exchange of data between players based on their proximity in the game world; and
- uses both direct peer-to-peer connections and relay connections to exchange data between player consoles.

2. Experience and Qualifications

a) Curriculum Vitae

2. The details of my education, work experience, and publications (including any publications authored in the last 10 years) are summarized in my curriculum vitae (“CV”) attached hereto as Appendix A of this Report.

3. In addition to my professional experience and training set forth in my CV, I have decades of experience playing games of all types on various devices and operating systems, including DOS, Windows, MacOS, iOS, Android, Xbox, PlayStation, Nintendo and Sega.

b) Prior Testimony

4. A list of any cases in which I have testified at deposition or trial or in written reports during at least the past five years is included in Appendix A of this Report.

c) Compensation

5. My rate of compensation for my work in this case is \$200 per hour plus any direct expenses incurred. My compensation is based solely on the amount of time that I devote to activity related to this case and is in no way affected by any opinions that I render. I receive no other compensation from work on this action. My compensation is not dependent on the outcome of this matter.

3. Materials Considered

6. I ran tests on the online modes of Grand Theft Auto V (“GTA”) as discussed herein. In conducting the tests, I relied on the instructions for GTA to configure the network settings. I also relied on the reference materials identified herein by citation. A list of additional materials considered is included as Appendix B of this Report.

7. I had a conversation with the infringement experts in this case, Drs. Nenad Medvidović and Michael Mitzenmacher, during which I explained to them the tests that I conducted that are discussed in this Report, including my methodology and the results of the tests.

4. Demonstratives

8. I anticipate that I may create or cause to be created demonstratives that I will use at trial to help explain various issues to the Court and jury, such as the nature of the tests I conducted described herein and the results of those tests. These demonstrative exhibits will include non-graphical illustrations (such as documents, charts, tables, etc.) and graphical illustrations (such as figures, drawings, pictures, videos, etc.). While these demonstratives have not yet been created, they will be completed and demonstrated at trial.

9. To further aid the Court and jury in understanding my opinion, I anticipate that I will demonstrate my testing discussed herein at trial, including either a live or prerecorded demonstration of my testing. A live demonstration may include having physical products in Court and/or access to the products via an Internet connection. I may create a video, or series of videos, demonstrating my testing of the accused products. I may also create a series of slides. These videos or slides may include voice overs, highlighting and call outs.

5. Testing Methodology

10. I used the following test setup and methodology for the “Direct Peer-to-Peer Connections”, “Use of Handshake Protocol” and “Usage of Relay Servers” Grand Theft Auto V scenarios:

- A Netgear hub with three connections:
 - Xbox One
 - a packet capture device: a GIGABYTE BRIX GB-BSi7HAL-6500 running Kali Linux 4.9.0-kali3-amd64 and Wireshark 2.2.5
 - A WatchGuard XTM26-W firewall
- I configured the Xbox One with a static IP address on the local network.
- I configured the WatchGuard device to forward all traffic according to the

documentation provided by the game publisher.¹

- I configured the WatchGuard device to route all traffic from the Xbox One through a unique public, static IP address, dedicated solely to traffic from the Xbox One.
- I connected the WatchGuard device to the Internet.
- I initiated a Wireshark capture session to capture network traffic to and from the Xbox One during the given test scenario.
- I initiated a gameplay session initiated.
- I analyzed the resulting capture sessions using Wireshark.

11. In addition to the above, the testing methodology for the “Observation of proximity-based data connections” GTA scenario included the following:

- A second setup identical to the setup above, with the exception that the local subnet and public IP address were unique to the second setup.
- A video capture for each gameplay session, synchronized with the associated network capture, and synchronized between the two Xbox Ones.

12. For each game scenario referenced above, several sessions and captures were performed and extensive analysis was performed on the resulting captures to characterize the communications.

6. Assumptions and Testing Background

13. I made the following assumptions in conducting my tests:

- Devices communicating on the ports designated in game documentation are transmitting game data.
- Other Xbox One devices participating in multi-player sessions are not located on Amazon AWS or other commercial infrastructure networks, Co-location or Content Distribution Network facilities.
- Game server infrastructure is not located on ISP residential customer networks.

¹ AB-TT 002818-24 (Grand Theft Auto V: <https://support.rockstargames.com/hc/en-us/articles/200525767-GTA-Online-Connection-Troubleshooting>).

14. In both the Transmission Control Protocol (“TCP”) and User Datagram Protocol (“UDP”) contexts network devices communicating over the internet via the Internet Protocol (“IP”) label their traffic with valid, routable source and destination addresses such as 166.170.58.233.² Three administrative systems that support the functionality of TCP/IP and UDP/IP communications provide information about the source and/or destination of network traffic: reverse DNS lookups (“rDNS”), Geo-location (“Geo-IP”), and the Autonomous System Number (“ASN”).

15. Where possible, I associated rDNS, Geo-IP, and ASN data with the captured network traffic to determine the location of the participant.

16. IP addresses are assigned to legal entities by various regional registries coordinated by the Internet Assigned Numbers Authority.³ If the block of addresses is large enough, the entity is given an ASN and is responsible for telling the rest of the internet how to route traffic to those IP addresses. The typical entity of this size is an ISP, but other organizations such as Microsoft, Amazon.com, Google Inc., and Electronic Arts, Inc. also have ASNs. Any “publicly routable” IP address can be associated with its registered ASN. It is reasonable to conclude that the ASN associated with an IP address indicates the network to which that IP address is connected.

17. The DNS protocol is used to map Fully Qualified Domain Names (“FQDN”) such as www.google.com to IP addresses such as 74.125.23.105.⁴ rDNS is used to map IP addresses back to FQDNs. Organizations such as ISPs use DNS and rDNS to aid in the management of their IP addresses and devices by associating IP addresses with informative FQDN entries that

² AB-TT 007677-81 (https://en.wikipedia.org/wiki/Internet_Protocol).

³ AB-TT 007682-83 (<https://www.iana.org/numbers>).

⁴ AB-TT 007657-73 (https://en.wikipedia.org/wiki/Domain_Name_System).

may contain indicators of physical or logical (network) location. For example, Comcast uses the FQDN for the IP address 67.176.165.137 (c-67-176-165-137.hsd1.il.comcast.net) to indicate the IP address itself, the type of network (hsd1: High Speed Data, a/k/a cable broadband), and the state (il: Illinois). Similarly the Comcast IP address 73.235.214.39 on their cable broadband network in California has the FQDN c-73-235-214-39.hsd1.ca.comcast.net.

18. Geolocation information is maintained by various entities and can be used to associate a given IP address with a physical location.⁵ The degree of accuracy and precision varies from IP address to IP address and from source to source. I used the MaxMind GeoLite2 City database which is considered reasonably accurate and precise.⁶

19. Combining the assumptions stated above and ASN, FQDN, and GeoIP data, I obtained an indication of whether a given endpoint is part of a game's server infrastructure or whether the endpoint is another Xbox One.

7. Testing of Grand Theft Auto V Online

20. **Direct Peer-to-Peer Connections:** Over a series of sessions in various missions and open world play, I observed direct peer-to-peer connections over the Internet with other players in the gameplay sessions, including players from outside the United States. The data I collected are attached as Appendix C to this Report.

21. According to Rockstar Games' website GTA5 primarily uses UDP Port 3074 for in-game communications.⁷

22. The following table contains a representative sample of endpoints observed during these sessions that communicated with my Xbox One during gameplay. I shaded in

⁵ AB-TT 007674-76 (<https://en.wikipedia.org/wiki/Geolocation>).

⁶ AB-TT 007655-56 (<http://dev.maxmind.com/geoip/geoip2/geolite2/>).

⁷ AB-TT 002631-35 (<https://support.rockstargames.com/hc/en-us/articles/200525767>).

yellow endpoints that I believe are Xbox One consoles, based on the analysis outlined above.

IP/FQDN	Geo IP Country	Geo IP City	ASN
c-98-226-154-185.hsd1.il.comcast.net	United States	Rockford, IL	AS7922 Comcast Cable Communications, LLC
68-184-3-197.dhcp.stls.mo.charter.com	United States		AS20115 Charter Communications
108-204-196-231.lightspeed.mmphtn.sbcglobal.net	United States	Memphis, TN	AS7018 AT&T Services, Inc.
cpe-72-190-208-109.satx.res.rr.com	United States	Seguin, TX	AS11427 Time Warner Cable Internet LLC
226.131.100.208.bendbroadband.com	United States	Bend, OR	AS21864 TDS TELECOM
c-73-247-89-231.hsd1.il.comcast.net	United States	Rockford, IL	AS7922 Comcast Cable Communications, LLC
ip98-178-179-131.tu.ok.cox.net	United States	Sand Springs, OK	AS22773 Cox Communications Inc.
173-218-23-81-grvl.mid.dyn.suddenlink.net	United States	Dardanelle, AR	AS19108 Suddenlink Communications
c-98-214-181-160.hsd1.il.comcast.net	United States	Washington, IL	AS7922 Comcast Cable Communications, LLC
CableLink64-110.telefonía.InterCable.net	Mexico	Monterrey, 19	AS11888 Televisión Internacional, S.A. de C.V.
47.186.202.215	United States	Denton, TX	AS5650 Frontier Communications of America, Inc.
162-201-218-67.lightspeed.cicril.sbcglobal.net	United States	Chicago, IL	AS7018 AT&T Services, Inc.
47.184.98.23	United States	Denton, TX	AS5650 Frontier Communications of America, Inc.
c-75-64-117-255.hsd1.tn.comcast.net	United States	Memphis, TN	AS7922 Comcast Cable Communications, LLC
47.184.83.61	United States	Denton, TX	AS5650 Frontier Communications of America, Inc.
CableLink-201-162-56-227.Hosts.InterCable.net	Mexico	San Nicolás De Los Garza, 19	AS11888 Televisión Internacional, S.A. de C.V.
c-73-220-9-166.hsd1.ca.comcast.net	United States	Orland, CA	AS7922 Comcast Cable Communications, LLC

ip68-105-119-105.sd.sd.cox.net	United States	Encinitas, CA	AS22773 Cox Communications Inc.
cpe-24-162-166-100.rgv.res.rr.com	United States	Brownsville, TX	AS11427 Time Warner Cable Internet LLC
c-73-217-205-59.hsd1.mo.comcast.net	United States	Greenwood, MO	AS7922 Comcast Cable Communications, LLC
69.55.36.82	United States	Watford City, ND	AS18780 Reservation Telephone Coop.
71-89-228-212.dhcp.reno.nv.charter.com	United States	Dayton, NV	AS20115 Charter Communications
104-183-22-244.lightspeed.rcsntx.sbcglobal.net	United States	Grand Prairie, TX	AS7018 AT&T Services, Inc.
64.146.199.49	United States	Aumsville, OR	AS11404 vanoppen.biz LLC
40.90.10.52	United States	Cheyenne, WY	AS8075 Microsoft Corporation
192.81.245.123	United States	New York, NY	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.
https-68-142-107-164.lax.llnw.net	United States	Tempe, AZ	AS22822 Limelight Networks, Inc.
108-196-101-186.lightspeed.sntcca.sbcglobal.net	United States	Santa Rosa, CA	AS7018 AT&T Services, Inc.
192.81.245.112	United States	New York, NY	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.
192.81.241.100	United States	New York, NY	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.
172.56.7.129	United States	Dallas, TX	AS21928 T-Mobile USA, Inc.
75-170-58-184.eugn.qwest.net	United States	Eugene, OR	AS209 Qwest Communications Company, LLC
131.253.29.133	United States		AS8075 Microsoft Corporation
host-76-11-211-139.newwavecomm.net	United States	Carmi, IL	AS18812 New Wave Communications
customer-GDL-228-194.megared.net.mx	Mexico	Guadalajara, 13	AS13999 Mega Cable, S.A. de C.V.
profile.xboxlive.com.akadns.net	United States		AS8075 Microsoft Corporation

xblwus.ipv6.microsoft.com.akadns.net	United States	San Jose, CA	AS8075 Microsoft Corporation
cy2.vortex.data.microsoft.com.akadns.net	United States	Cheyenne, WY	AS8075 Microsoft Corporation
fixed-187-190-163-181.totalplay.com.mx	Mexico	Toluca, 15	AS17072 TOTAL PLAY TELECOMUNICACIONES SA DE CV
99-121-123-5.lightspeed.glptms.sbcglobal.net	United States	Moss Point, MS	AS7018 AT&T Services, Inc.
40.90.10.180	United States	Cheyenne, WY	AS8075 Microsoft Corporation
8.40.254.224	United States	Purcell, OK	AS22898 ATLINK SERVICES, LLC
c-67-176-165-137.hsd1.il.comcast.net	United States	Morris, IL	AS7922 Comcast Cable Communications, LLC
c-24-22-199-127.hsd1.wa.comcast.net	United States	Everett, WA	AS7922 Comcast Cable Communications, LLC
msnbot-65-52-108-233.search.msn.com	United States	Boydton, VA	AS8075 Microsoft Corporation
cpe-172-113-127-118.socal.res.rr.com	United States		AS20001 Time Warner Cable Internet LLC
cpe-104-175-157-173.socal.res.rr.com	United States	Moreno Valley, CA	AS20001 Time Warner Cable Internet LLC
157.56.149.60	United States	Chicago, IL	AS8075 Microsoft Corporation
200.56.109.172.dsl.dyn.telnor.net	Mexico	Tijuana, 02	AS8151 Uninet S.A. de C.V.
187-167-219-78.static.axtel.net	Mexico	Monterrey, 19	AS6503 Axtel, S.A.B. de C.V.
108-206-195-144.lightspeed.cicril.sbcglobal.net	United States	Chicago, IL	AS7018 AT&T Services, Inc.
201.170.54.50.dsl.dyn.telnor.net	Mexico	Tijuana, 02	AS8151 Uninet S.A. de C.V.
201-157-91-35.internetmax.maxcom.net.mx	Mexico	Mexico, 09	AS22566 Maxcom Telecomunicaciones, S.A.B. de C.V.
vpc-elb-rpdxmweb-1283008362.us-west-2.elb.amazonaws.com	United States	Boardman, OR	AS16509 Amazon.com, Inc.
c-73-235-214-39.hsd1.ca.comcast.net	United States	Elk Grove, CA	AS7922 Comcast Cable Communications, LLC
96-3-99-228-dynamic.midco.net	United States	Bismarck, ND	AS11232 Midcontinent Communications

134.170.178.9	United States		AS8075 Microsoft Corporation
68-171-177-25.wtcks.net	United States	Wamego, KS	AS19504 WTC Communications, Inc.
cpe-45-51-220-15.socal.res.rr.com	United States	San Bernardino, CA	AS20001 Time Warner Cable Internet LLC
174-28-60-156.albq.qwest.net	United States	Rio Rancho, NM	AS209 Qwest Communications Company, LLC
149.12.1.145	United Kingdom	Aylesbury, B9	AS48945 Independent Fibre Networks Limited
109.77.207.82	Ireland	Edgeworthstown, 18	AS15502 Vodafone Ireland Limited
177.237.75.135.cable.dyn.cableonline.com.mx	Mexico	Cancún, 23	AS28512 Cablemas Telecomunicaciones SA de CV
138-229-152-117.dhcp.rvsvd.ca.charter.com	United States	California City, CA	AS20115 Charter Communications
173-217-87-21-lkch.mid.dyn.suddenlink.net	United States	Sulphur, LA	AS19108 Suddenlink Communications
47.188.188.76	United States	Irving, TX	AS5650 Frontier Communications of America, Inc.
c-73-153-81-52.hsd1.co.comcast.net	United States	Englewood, CO	AS7922 Comcast Cable Communications, LLC
d50-92-1-232.bchsia.telus.net	Canada		AS852 TELUS Communications Inc.
c-76-107-197-201.hsd1.tn.comcast.net	United States	Cordova, TN	AS7922 Comcast Cable Communications, LLC
65.55.158.118	United States	San Jose, CA	AS8075 Microsoft Corporation
d66-183-193-134.bchsia.telus.net	Canada	Surrey, BC	AS852 TELUS Communications Inc.
cpe-24-243-101-171.rgv.res.rr.com	United States	Mcallen, TX	AS11427 Time Warner Cable Internet LLC
173-216-13-210-cabt.mid.dyn.suddenlink.net	United States	Cabot, AR	AS19108 Suddenlink Communications
cpe-23-241-203-33.socal.res.rr.com	United States		AS20001 Time Warner Cable Internet LLC
cpe-107-185-191-165.socal.res.rr.com	United States	Hemet, CA	AS20001 Time Warner Cable Internet LLC
cpe-172-251-201-15.socal.res.rr.com	United States	Hemet, CA	AS20001 Time Warner Cable Internet LLC

24-217-157-124.dhcp.stls.mo.charter.com	United States	O Fallon, MO	AS20115 Charter Communications
68-112-202-126.dhcp.eucl.wi.charter.com	United States	Sun Prairie, WI	AS20115 Charter Communications
static24-72-9-246.r.rev.accesscomm.ca	Canada	Regina, SK	AS21804 Access Communications Co-operative Limited
207-118-138-185.dyn.centurytel.net	United States	Denmark, WI	AS22561 CenturyTel Internet Holdings, Inc.
ip68-7-4-19.sd.sd.cox.net	United States	San Marcos, CA	AS22773 Cox Communications Inc.
bas1-markham24-70-26-174-218.dsl.bell.ca	Canada		AS577 Bell Canada
99-125-122-142.lightspeed.rcsntx.sbcglobal.net	United States	Frisco, TX	AS7018 AT&T Services, Inc.
104-51-118-137.lightspeed.mssnks.sbcglobal.net	United States	Lees Summit, MO	AS7018 AT&T Services, Inc.
c-50-129-146-188.hsd1.il.comcast.net	United States	Carpentersville, IL	AS7922 Comcast Cable Communications, LLC
dsl-187-145-62-251-dyn.prod-infinitum.com.mx	Mexico	Querétaro, 22	AS8151 Uninet S.A. de C.V.
187.250.60.244.dsl.dyn.telnor.net	Mexico	Tijuana, 02	AS8151 Uninet S.A. de C.V.
97-87-190-195.dhcp.stls.mo.charter.com	United States	Lake Saint Louis, MO	AS20115 Charter Communications
56.4.135.77.rev.sfr.net	France	Solliès-toucas, B8	AS15557 SFR
24-196-34-5.dhcp.fdul.wi.charter.com	United States	Lakeville, MN	AS20115 Charter Communications
63.243.244.10	United States	Cambridge, MA	AS6453 TATA COMMUNICATIONS (AMERICA) INC
192.81.245.125	United States	New York, NY	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.
75-132-90-224.dhcp.stls.mo.charter.com	United States	Saint Louis, MO	AS20115 Charter Communications
b.resolvers.Level3.net	United States		AS3356 Level 3 Communications, Inc.

c-73-75-147-169.hsd1.il.comcast.net	United States	Hoffman Estates, IL	AS7922 Comcast Cable Communications, LLC
dsl-187-143-149-218-dyn.prod-infinitem.com.mx	Mexico	Coatzacoalcos, 30	AS8151 Uninet S.A. de C.V.
ip184-186-86-246.tu.ok.cox.net	United States	Bixby, OK	AS22773 Cox Communications Inc.
134.170.179.74	United States		AS8075 Microsoft Corporation
vpc-elb-rpdxmweb-1283008362.us-west-2.elb.amazonaws.com	United States	Boardman, OR	AS16509 Amazon.com, Inc.
cblmdm170-253-222-167.maxxsouthbb.net	United States	Shannon, MS	AS46687 BCI Mississippi Broadband, LLC
d104-205-174-99.abhsia.telus.net	Canada		AS852 TELUS Communications Inc.
r74-193-146-183.lkchemta01.lkchla.by.dh.suddenlink.net	United States	Greenville, MS	AS19108 Suddenlink Communications
134.170.179.21	United States		AS8075 Microsoft Corporation
google-public-dns-a.google.com	United States	Mountain View, CA	AS15169 Google Inc.
a23-196-3-157.deploy.static.akamaitechnologies.com	United States	Cambridge, MA	AS20940 Akamai International B.V.
a23-49-4-89.deploy.static.akamaitechnologies.com	United States	Cambridge, MA	AS2914 NTT America, Inc.
40.90.10.165	United States	Cheyenne, WY	AS8075 Microsoft Corporation
65.158.122.18	United States		AS209 Qwest Communications Company, LLC
134.170.178.89	United States		AS8075 Microsoft Corporation
134.170.179.25	United States		AS8075 Microsoft Corporation
192.81.245.113	United States	New York, NY	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.
192.81.245.114	United States	New York, NY	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.

192.81.245.121	United States	New York, NY	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.
40.90.10.47	United States	Cheyenne, WY	AS8075 Microsoft Corporation
40.90.10.83	United States	Cheyenne, WY	AS8075 Microsoft Corporation
msnbot-65-52-108-92.search.msn.com	United States	Boydton, VA	AS8075 Microsoft Corporation
131.253.28.135	United States		AS8075 Microsoft Corporation

23. **Use of Handshake Protocol:** I observed that GTA uses the following characteristic handshake to connect to and exchange game-play data with other players. The data I collected are attached as Appendix D and E to this Report. Two initial 52 byte blocks of data are exchanged between the peers. Each 52 byte block of data contains an identical eight byte sequence which serves to identify and confirm each peer as being a device wishing to participate in a GTA game-play session. Each 52 byte block of data also contains a unique eight byte sequence which serves as the unique identifier of the peer. Here the local Xbox (highlighted in green) identifies itself using the string **0c e2 f3 fd ba b4 f5 8a** and the remote Xbox (highlighted in yellow) identifies itself using the string **20 6c f3 fd b3 24 0c f0**.

```
00000000 60 00 00 00 00 00 3b 15 20 01 00 00 0d 5d a5 99 `.....; .....]..
00000010 20 6c f3 fd b3 24 0c f0 20 01 00 00 0d 5d a5 99 l..$.. .....]..
00000020 0c e2 f3 fd ba b4 f5 8a 01 04 00 00 00 00 04 04 ..... .....]..
00000030 00 00 00 00 .....
```

```
00000000 60 00 00 00 00 00 3b 15 20 01 00 00 0d 5d a5 99 `.....; .....]..
00000010 0c e2 f3 fd ba b4 f5 8a 20 01 00 00 0d 5d a5 99 ..... .....]..
00000020 20 6c f3 fd b3 24 0c f0 01 04 d1 4f c6 5d 04 04 l..$.. ...O.]..
00000030 01 00 00 00 .....
```

24. From this information I was able to identify in-game peer-to-peer data exchanges even if the port number in use differs from the standard UDP port 3075. The data I collected are attached as Appendix F to this Report. The following tables list a sample of in-game peer connections using non-standard ports and a breakdown of port ranges in use.

peer	source port
dsl-189-179-120-22-dyn.prod-infinitum.com.mx	64509
customer-189-216-207-166.cablevision.net.mx	57552
static-50-53-156-116.bvtn.or.frontiernet.net	49903
189.215.49.182.cable.dyn.cableonline.com.mx	48010
162.255.235.254	47808
p18.vchighlandsnv.hsbnv.net	36938
dsl-201-110-157-229-dyn.prod-infinitum.com.mx	27584
jet-jet-internet-cpr.consolidated.net	18622
fixed-187-190-76-87.totalplay.net	10184
dsl-201-127-38-151-dyn.prod-infinitum.com.mx	10080
fixed-187-188-67-139.totalplay.net	9424
customer-TOR-169-205.megared.net.mx	8732
135.sub-174-197-0.myvzw.com	7521

Peers connected for in-game data exchange using non-standard ports.

port range	peer connections	percent of total
ports 0-999	18	5.26%
ports 10,000-19,999	26	7.60%
ports 20,000-29,999	22	6.43%
ports 30,000-39,999	20	5.85%
ports 40,000-49,999	63	18.42%

ports 50,000-59,999	174	50.88%
ports 60,000-64,509	19	5.56%
all ports	342	100.00%

Connections per port range and percent of total.

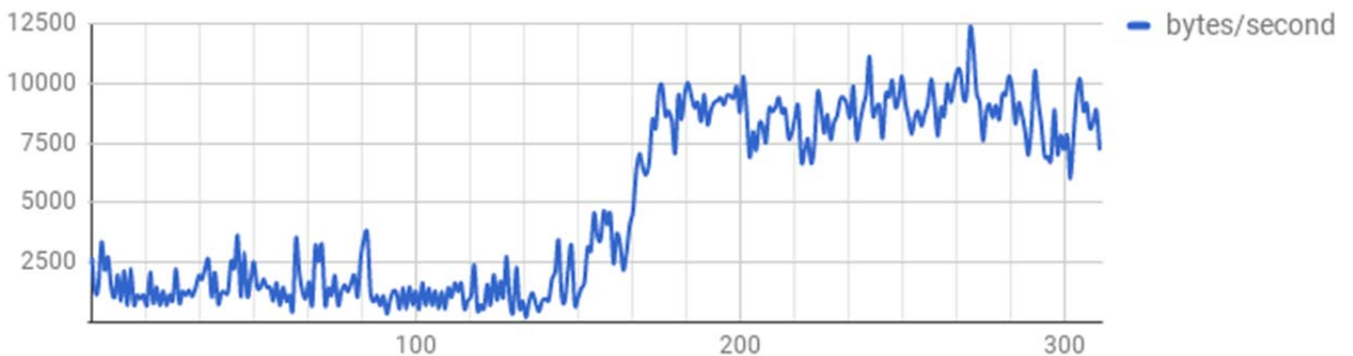
25. **Usage of Relay Servers:** I verified that GTA uses relay servers in addition to direct peer-to-peer connections to distribute gameplay data. The data I collected are attached as Appendix G to this Report. The table below shows endpoints connected during a gameplay session in which what appear to be Take-Two Interactive servers were handling gameplay traffic. The AS Number information for all of these IP addresses is listed as “AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.” and the GeoIP information locates them in New York, NY. Furthermore they all belong to a relatively narrow IP address range, something that generally indicates a single owner. All of the IP addresses in question fall within the ranges 192.81.241.114 to 192.81.241.123 and 192.81.245.111 to 192.81.245.126.

Address	AS Number	City	Country
192.81.245.124	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.	New York, NY	United States
192.81.245.126	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.	New York, NY	United States
192.81.245.114	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.	New York, NY	United States
192.81.245.125	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.	New York, NY	United States
192.81.245.112	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.	New York, NY	United States
192.81.245.111	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.	New York, NY	United States
192.81.241.123	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.	New York, NY	United States
192.81.241.114	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.	New York, NY	United States

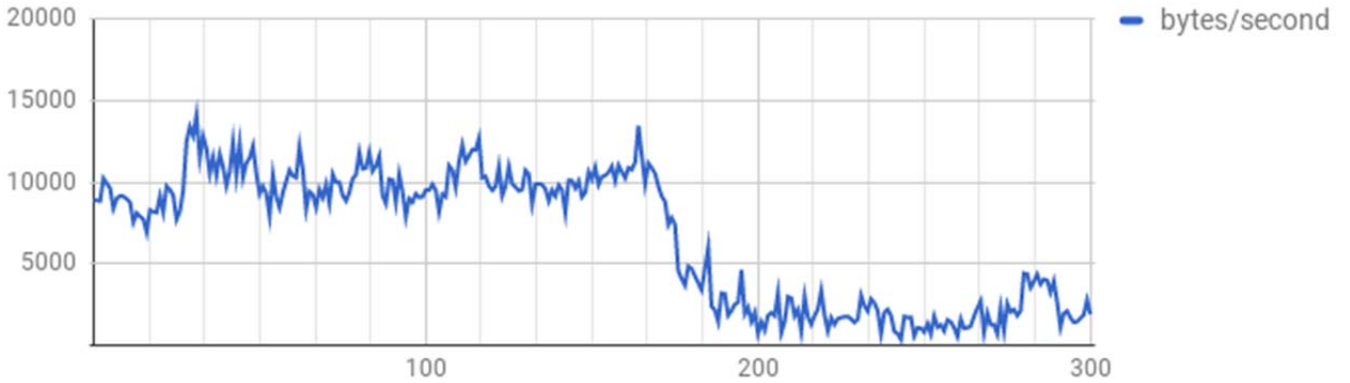
192.81.245.115	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.	New York, NY	United States
192.81.245.122	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.	New York, NY	United States
192.81.245.116	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.	New York, NY	United States
192.81.245.119	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.	New York, NY	United States
192.81.245.121	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.	New York, NY	United States
192.81.245.123	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.	New York, NY	United States
192.81.241.100	AS46555 TAKE-TWO INTERACTIVE SOFTWARE, INC.	New York, NY	United States

26. **Observation of proximity-based data connections:** I tested GTA to determine if the peer-to-peer data transfer is dependent on the in-game proximity of the players sharing the data. I ran two scenarios; one in which two players (whom I was controlling on separate Xboxes) started apart and were brought together, and one in which the players started together and then moved apart. In both cases the network captures demonstrate that a higher rate of data exchange occurs when players are closer together than when they are farther apart. The data I collected are attached as Appendix H to this Report.

27. The graph below represents the rate of data being exchanged between Test Player A and Test Player B in bytes per second as they start at remote points on the map and move closer together over the course of five minutes.



28. The graph below represents the rate of data being exchanged between Test Player A and Test Player B in bytes per second as they start from a proximal configuration and move to remote points on the map over the course of five minutes.



I declare under penalty of perjury under the laws of the United States that the foregoing is true and correct. Executed on October 2, 2017 in Newport Beach, California.

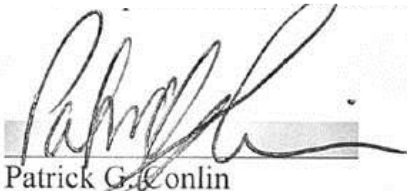

Patrick G. Conlin

EXHIBIT 3

Deathmatches in GTA Online

For other uses, see Deathmatch.

Deathmatch is a Job in *Grand Theft Auto Online*.

Contents [\[show\]](#)

Description

Deathmatches return in *GTA Online*, the multiplayer of *Grand Theft Auto V*, with three different types of match available, Standard, Team and Vehicle deathmatches, although any deathmatch map can be played as a standard deathmatch or as a team deathmatch, despite Rockstar labeling them differently in the menu and in free-roam. Up to 16 players can play in a Deathmatch. Deathmatches can be created using the Content Creator.

An attritional (no re-spawning) variation on the deathmatch game is also available, called Last Team Standing.

The jobs are accessed from Free Mode by either walking into their corona or selecting them from the Pause Menu where they can be activated from the map or the Online Jobs list or from post-job voting menus.



A deathmatch in *GTA Online*.



Object

Matches

Variab

The

- Ma
- Nu
- Ti
- Ta
- On
- Ob
- Ti
- We
- Tr
- Lo
- Ra
- Fo

Variables

Weapon

The map

unlocked

Pistols

- Pistol
- Combat Pistol
- AP Pistol
- SNS Pistol
- Heavy Pistol
- Vintage Pistol
- Marksman Pistol
- Pistol .50
- Flare Gun
- Heavy Revolver

Shotguns

- Pump
- Sawed-off Shotgun
- Assault Shotgun
- Heavy Shotgun

Share

Add to article

Deathmatch-GTAO.jpg

See full size image

▪ Bulpup  Added by Carl Johnson Jr. Posted in Deathmatches in GTA Online

▪ Double Barrel Shotgun 

▪ Sweep 1-5 of 5 gun 

MGs

▪ Micro 

▪ SMG 

▪ MG 

▪ Combat MG 

▪ Combat PD 

▪ Machine Pisto 


▪ Assault Rifle 

Rifles

▪ Assault Rifle 

▪ Carbine D9 

▪ Advanced 

▪ Special Carbine 

▪ Bulpup  93,422 more items on this wiki

▪ Comp 

Sniper Rifle

▪ Sniper 

▪ Heavy 

▪ Marksman Rifle 

Heavy

▪ Grenade 

▪ RPG 

▪ Minigun 

▪ Homing Launcher 

▪ Combat Launcher 

Grenades

▪ Grenade 

▪ Sticky 

▪ Tear Gas 

▪ Molotov 

▪ Jerry Can 

▪ Proximity Mines 

▪ Pipe Bomb 

Melee

▪ Knife 

▪ Nightstick 

▪ Baseball Bat 

▪ Crowbar 

▪ Golf Club 

▪ Bottle 

▪ Antiquities 

▪ Knuckle 

▪ Mace 

▪ Hatchet 

▪ Hammer 

▪ Switchblade 

▪ Pool Cue 

▪ Battle Axe 

▪ Pipe Wrench 

Special

▪ Armor 

▪ Health 

▪ Parachute 

List of Deathmatchese

▪ Bluffs

- Compound
- Concrete
- Del Perro
- Elysian Island
- Hospital
- Morgue
- Movie Set
- Paleta Slay
- Pavilion
- Property Values *(added in the Valentine's Day Massacre Special)*
- Richman Mansion
- Salvage
- Suppressing Fire
- Vineyard



Deathmatch icon

Team Deathmatches

- Ace Liquor
- Back Alley
- Basic Training
- Beachfront *(added in the Beach Bum Content Update)*
- Berth Brawl *(added in the enhanced version)*
- Biolab
- Biolab Redux *(added in the enhanced version)*
- Boatyard
- Bottleneck *(added in the enhanced version)*
- Canals
- Cape Catfish *(added in the Beach Bum Content Update)*
- Cargo
- Chamberlain Hills
- Chasing Shots
- Commune
- Construction
- Crossfire
- Derelict Motel
- Downtown
- El Burro Heights
- Farmhouse Fracas *(added in the enhanced version)*
- Farmland
- First Base
- Fourth Amendment *(added in the enhanced version)*
- Gentry Manor
- Getaway
- Governmental
- Grit
- Hangar
- Industrialize *(added in the enhanced version)*
- Legal Eagle *(added in The Business Update)*
- Mansion
- Maze
- North Rockford
- Observatory
- Paleta Beach *(added in the Beach Bum Content Update)*
- Palomino Highlands *(added in the Beach Bum Content Update)*
- Panic Attack *(added in the Independence Day Special)*
- Pier
- Pillbox Hill
- Procopio Beach *(added in the Beach Bum Content Update)*
- Puerto Del Sol *(added in the I'm Not a Hipster Update)*
- Pump Action
- Railyard
- Rancho Projects
- Refinery
- Reflex
- Road Tripping *(added in the Independence Day Special)*



Team Deathmatch icon

- Scrapyard
- Shotgun Wedding *(added in the Valentine's Day Massacre Special)*
- Shooting Up
- Supply
- Symmetry
- Terminal
- Trailer Park
- Vespucci Beachfront *(added in the Beach Bum Content Update)*
- Vespucci Shoreline *(added in the Beach Bum Content Update)*
- Vinewood Kills
- Wrath

Vehicle Deathmatches

- Buzz Kill
- Corporate Shell-out *(added in The Business Update)*
- Desert Storm
- Fort Zancudo
- Senora Airstrip
- Shellshocked *(added in the Beach Bum Content Update)*
- Vulture Capital *(added in The Business Update)*

Bonuses

Vehicle Deathmatch icon

- On the 7th September, 2015, there was double RP and GTA\$ on all Deathmatches in GTA Online, as part of the Labor Day Sales.

See Also

- Liberty City Survivor - *GTA Liberty City Stories* and *GTA Chinatown Wars* equivalents.
- Last Team Standing - Similar, no-respawn setup also appearing in *GTA Online*.

Navigation

Navigation		[hide]
Grand Theft Auto series	[show]	
Missions in <i>Grand Theft Auto Online</i>	[show]	
Deathmatches in <i>Grand Theft Auto Online</i>	[show]	

Retrieved from "https://gta.fandom.com/wiki/Deathmatches_in_GTA_Online?oldid=1038023"

Categories: Missions | Missions in GTA Online | Deathmatch | Game Modes

Community content is available under CC-BY-SA unless otherwise noted.

EXHIBIT 4



US006701344B1

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

(10) **Patent No.:** **US 6,701,344 B1**
(45) **Date of Patent:** ***Mar. 2, 2004**

- (54) **DISTRIBUTED GAME ENVIRONMENT**
- (75) Inventors: **Fred B. Holt**, Seattle, WA (US); **Virgil E. Bourassa**, Bellevue, WA (US)
- (73) Assignee: **The Boeing Company**, Seattle, WA (US)
- (*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 417 days.

- 5,734,865 A 3/1998 Yu
- 5,737,526 A 4/1998 Periasamy et al.
- 5,754,830 A 5/1998 Butts et al.
- 5,761,425 A 6/1998 Miller
- 5,764,756 A 6/1998 Onweller

(List continued on next page.)

OTHER PUBLICATIONS

PR Newswire, "Microsoft Boosts Accessibility to Internet Gaming Zone with Latest Release," Apr. 27, 1998, pp. 1ff.*
PR Newswire, "Microsoft Announces Launch Date for UltraCorps, Its Second Premium Title for the Internet Gaming Zone," Ma 27, 1998, pp. 1ff.*
Business Wire, "Boeing and Panthesis Complete SWAN Transaction," Jul. 22, 2002, pp. 1ff.*

(List continued on next page.)

Primary Examiner—Dung C. Dinh
Assistant Examiner—Bradley Edelman
(74) *Attorney, Agent, or Firm*—Perkins Coie LLP

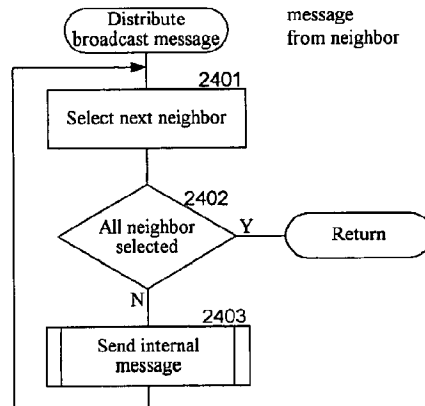
- (21) Appl. No.: **09/629,042**
- (22) Filed: **Jul. 31, 2000**
- (51) Int. Cl.⁷ **G06F 15/16**
- (52) U.S. Cl. **709/204; 709/205; 709/203; 709/243; 463/42**
- (58) **Field of Search** 709/204, 205, 709/227, 243, 203; 463/40, 42

(57) **ABSTRACT**

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

- (56) **References Cited**
- U.S. PATENT DOCUMENTS**
- 4,912,656 A 3/1990 Cain et al.
- 5,056,085 A 10/1991 Vu
- 5,058,105 A * 10/1991 Mansour et al. 370/228
- 5,079,767 A * 1/1992 Perlman 370/408
- 5,117,422 A * 5/1992 Hauptschein et al. 370/255
- 5,309,437 A 5/1994 Perlman et al.
- 5,426,637 A 6/1995 Derby et al.
- 5,459,725 A * 10/1995 Bodner et al. 370/390
- 5,471,623 A * 11/1995 Napolitano, Jr. 709/243
- 5,535,199 A 7/1996 Amri et al.
- 5,568,487 A 10/1996 Sithon et al.
- 5,636,371 A 6/1997 Yu
- 5,644,714 A * 7/1997 Kikinis 709/219
- 5,673,265 A 9/1997 Gupta et al.
- 5,696,903 A 12/1997 Mahany
- 5,732,074 A 3/1998 Spaur et al.
- 5,732,086 A * 3/1998 Liang et al. 370/410
- 5,732,219 A 3/1998 Blumer et al.

19 Claims, 39 Drawing Sheets



US 6,701,344 B1

Page 2

U.S. PATENT DOCUMENTS

5,790,548	A	8/1998	Sistanizadch et al.	
5,790,553	A	8/1998	Deaton, Jr. et al.	
5,799,016	A	8/1998	Onweller	
5,802,285	A	9/1998	Hirviniemi	
5,850,592	A	* 12/1998	Ramanathan	455/7
5,864,711	A	1/1999	Mairs et al.	
5,867,660	A	2/1999	Schmidt et al.	
5,867,667	A	2/1999	Butman et al.	
5,870,605	A	2/1999	Bracho et al.	
5,874,960	A	2/1999	Mairs et al.	
5,899,980	A	5/1999	Wilf et al.	
5,907,610	A	5/1999	Onweller	
5,925,097	A	* 7/1999	Gopinath et al.	709/200
5,928,335	A	7/1999	Morita	
5,935,215	A	8/1999	Bell et al.	
5,948,054	A	9/1999	Nielsen	
5,949,975	A	9/1999	Batty et al.	
5,956,484	A	9/1999	Rosenberg et al.	
5,970,232	A	* 10/1999	Passint et al.	709/238
5,974,043	A	10/1999	Solomon	
5,987,506	A	11/1999	Carter et al.	
6,003,088	A	12/1999	Houston et al.	
6,013,107	A	1/2000	Blackshear et al.	
6,023,734	A	2/2000	Ratcliff et al.	
6,029,171	A	2/2000	Smiga et al.	
6,032,188	A	2/2000	Mairs et al.	
6,038,602	A	3/2000	Ishikawa	
6,047,289	A	4/2000	Thorne et al.	
6,094,676	A	7/2000	Gray et al.	
6,115,580	A	* 9/2000	Chuprun et al.	455/1
6,167,432	A	* 12/2000	Jiang	709/204
6,173,314	B1	* 1/2001	Kurashima et al.	709/204
6,199,116	B1	3/2001	May et al.	
6,216,177	B1	4/2001	Mairs et al.	
6,223,212	B1	4/2001	Batty et al.	
6,243,691	B1	6/2001	Fisher et al.	
6,268,855	B1	7/2001	Mairs et al.	
6,271,839	B1	8/2001	Mairs et al.	
6,272,548	B1	* 8/2001	Cotter et al.	709/239
6,285,363	B1	9/2001	Mairs et al.	
6,304,928	B1	10/2001	Mairs et al.	
6,321,270	B1	* 11/2001	Crawley	709/238
6,463,078	B1	* 10/2002	Engstrom et al.	370/466
6,524,189	B1	* 2/2003	Rautala	463/40
2002/0027896	A1	* 3/2002	Hughes et al.	370/342

OTHER PUBLICATIONS

Azar et al., "Routing Strategies for Fast Networks," May 1992, INFOCOM '92, Eleventh Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 1, 170-179.*

Cho et al., "A Flood Routing Method for Data Networks," Sep. 1997, Proceedings of 1997 International Conference on Information, Communications, and Signal Processing, vol. 3, pp. 1418-1422.*

Komine et al., "A Distributed Restoration Algorithm for Multiple-Link and Node Failures of Transport Networks," Dec. 199 Global Telecommunications Conference, 1990, and Exhibition, IEEE, vol. 1, pp. 459-463.*

Peercy et al., "Distributed Algorithms for Shortest-Path, Deadlock-Free Routing and Broadcasting in Arbitrarily Faulty Hypercubes," Jun. 1990, 20th International Symposium on Fault-Tolerant Computing, 1990, pp. 218-225.*

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

U.S. patent application Ser. No. 09/629,577, Bourassa et al., filed Jul. 31, 2000.

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

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

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

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

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

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

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

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

The O'Reilly Network, "Gnutella: Alive, Well, and Changing Fast," Jan. 25, 2001 (5 pages) <http://www.open2p.com/1pt...> [Accessed Jan. 29, 2002].

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

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

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

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

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

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

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

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

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

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

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

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

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

* cited by examiner

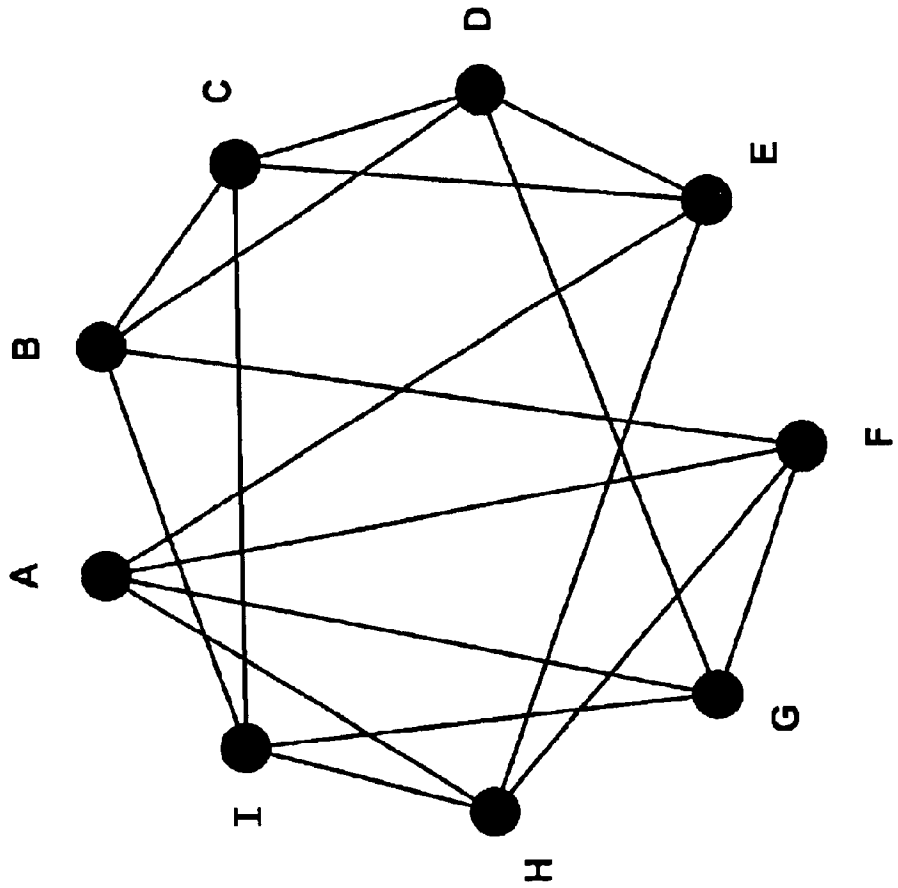


Fig. 1

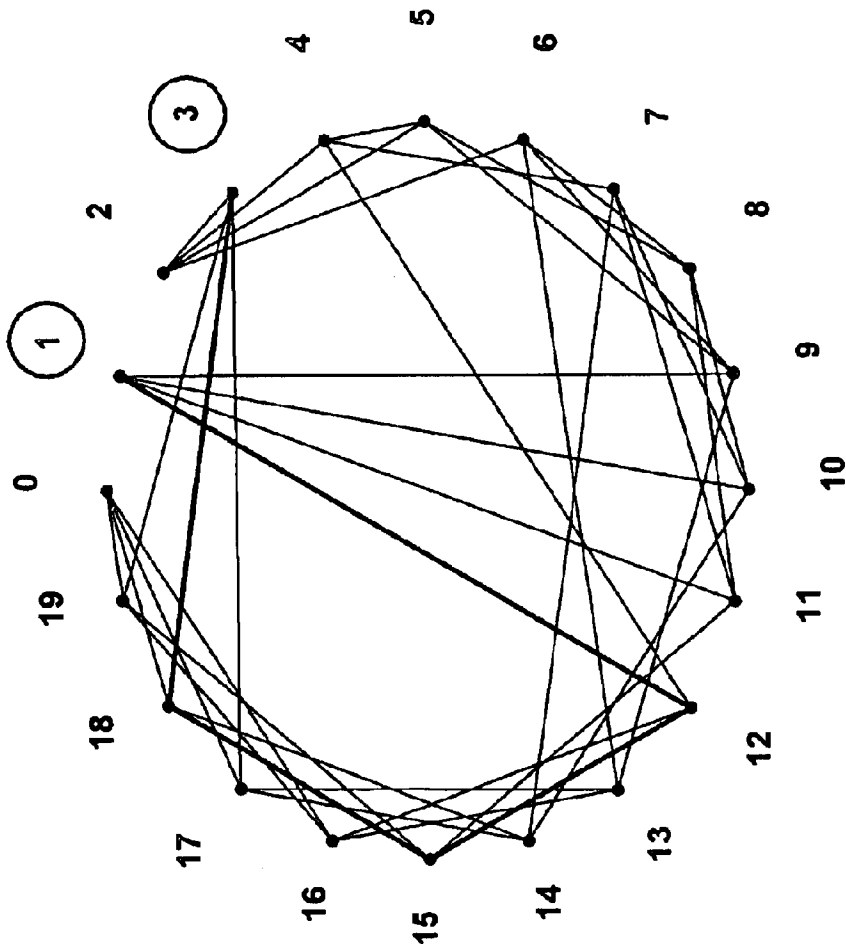


Fig. 2

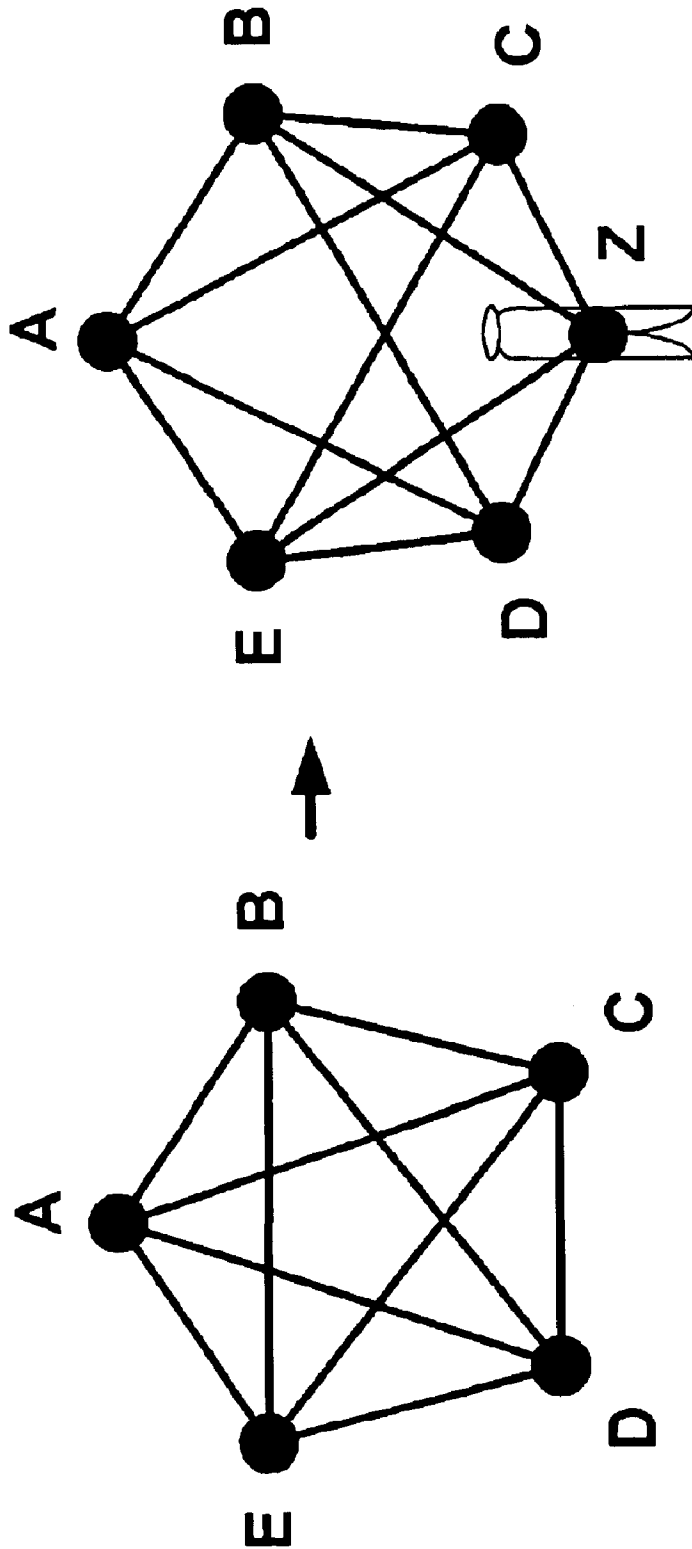


Fig. 3B

Fig. 3A

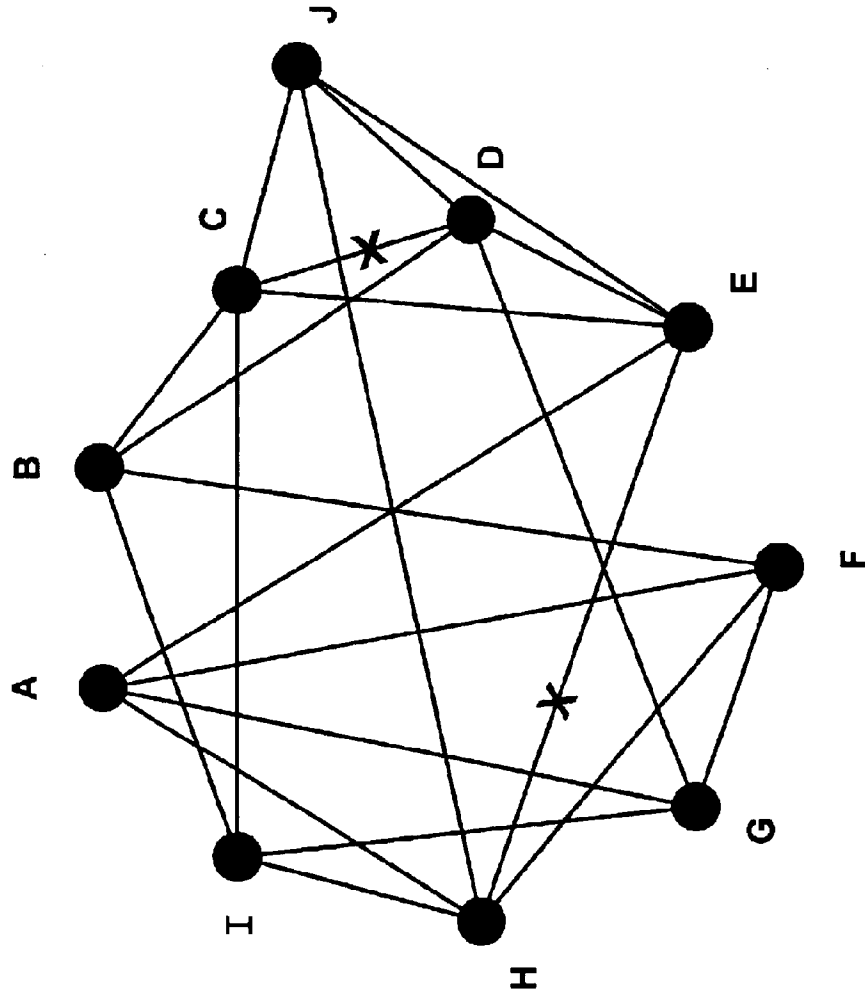


Fig. 4A

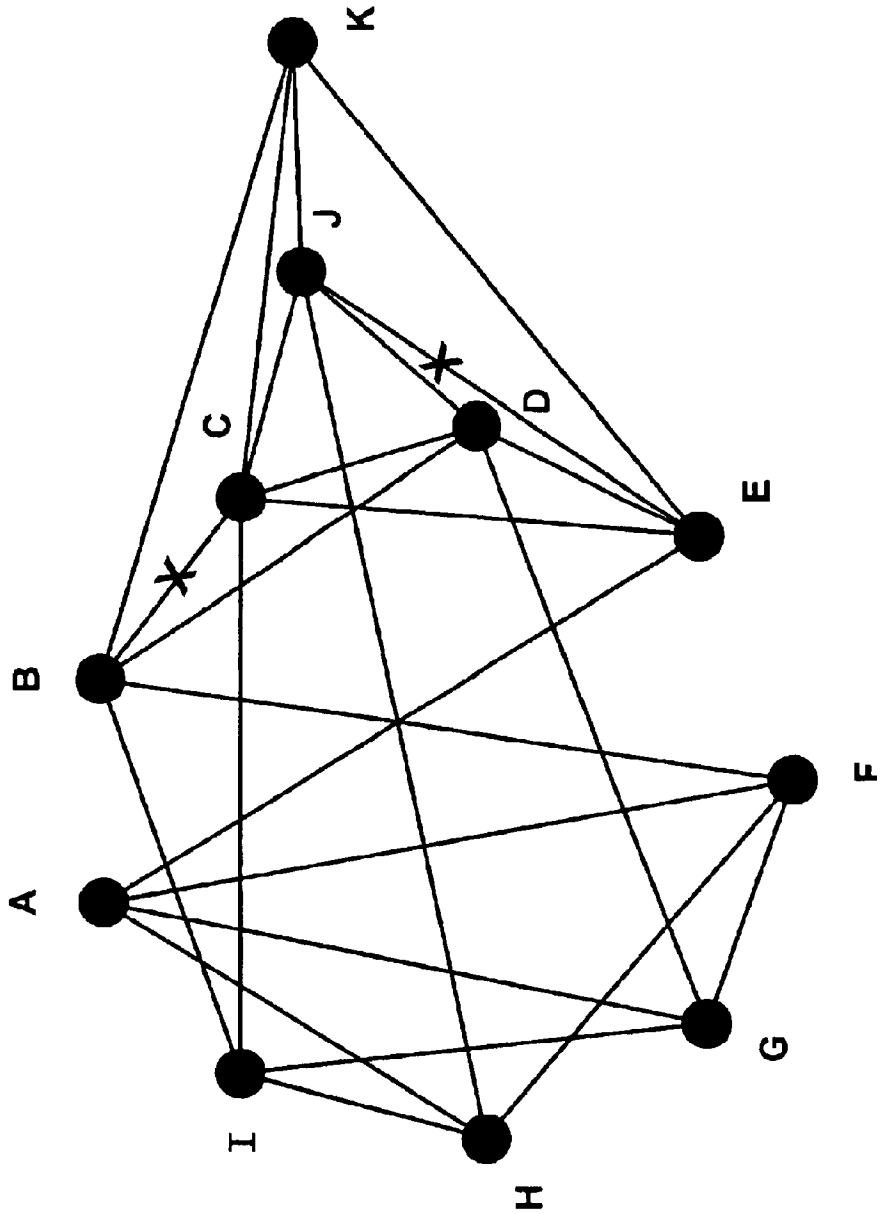


Fig. 4B

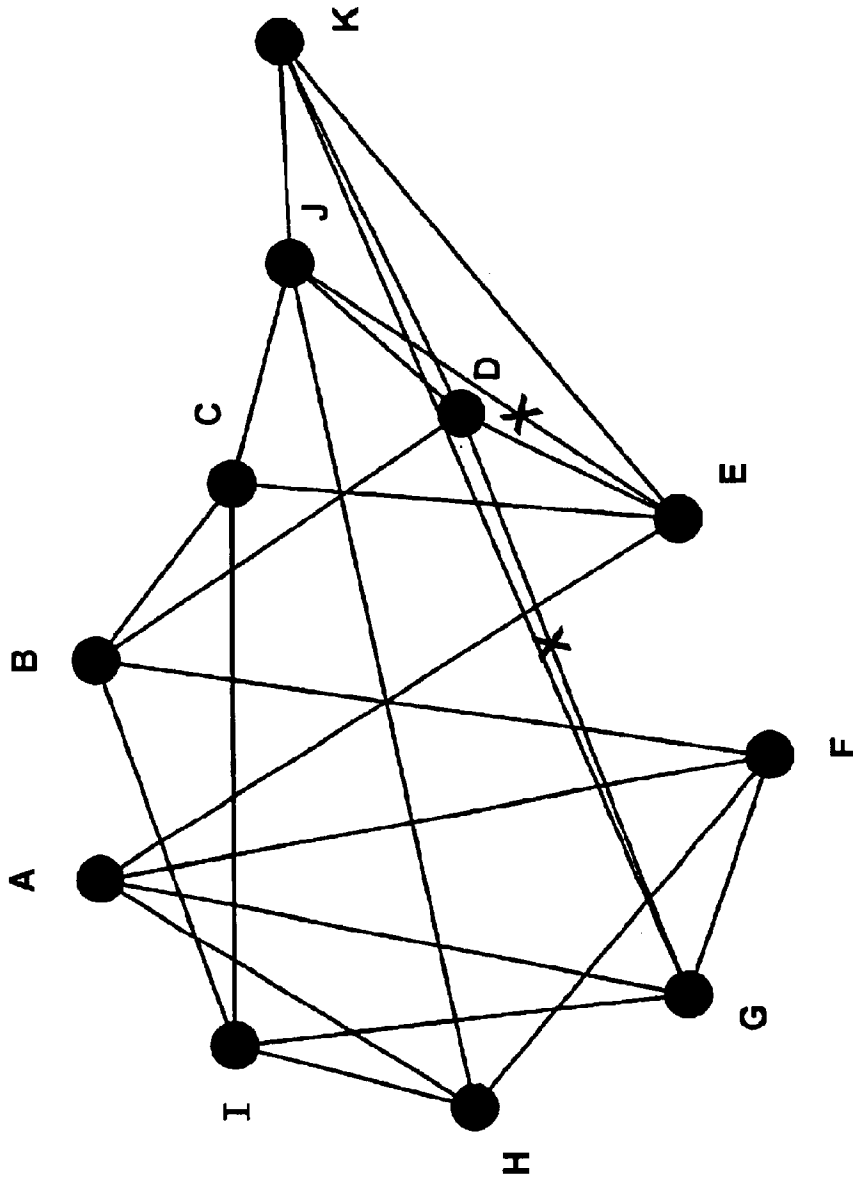


Fig. 4C

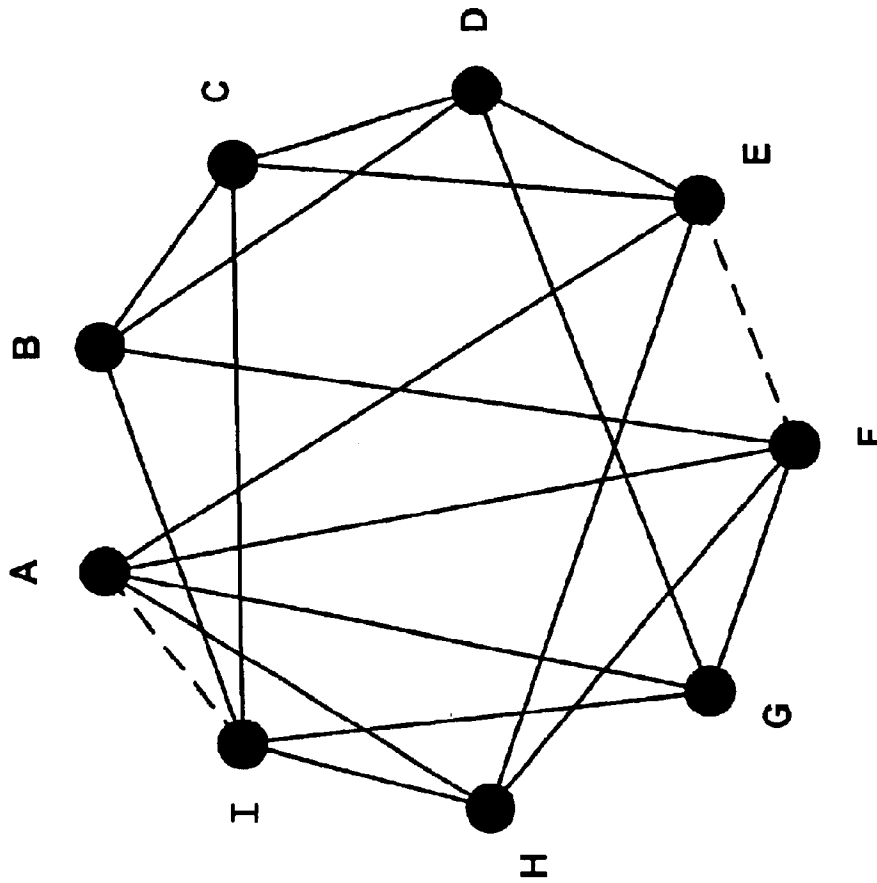


Fig. 5A

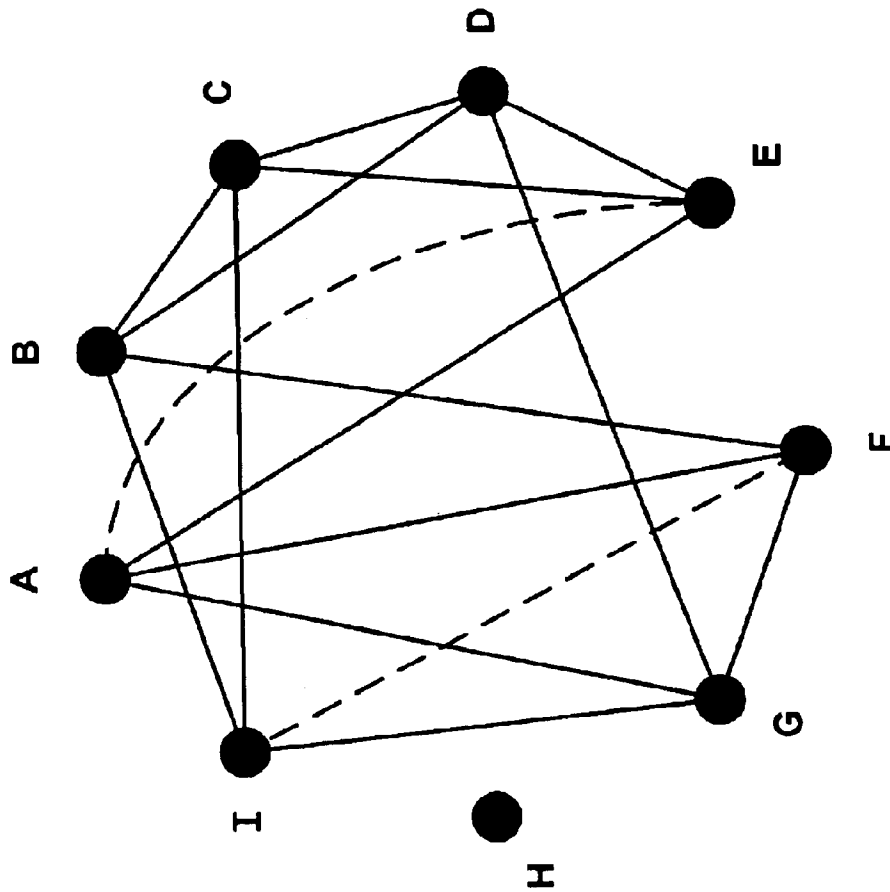


Fig. 5B

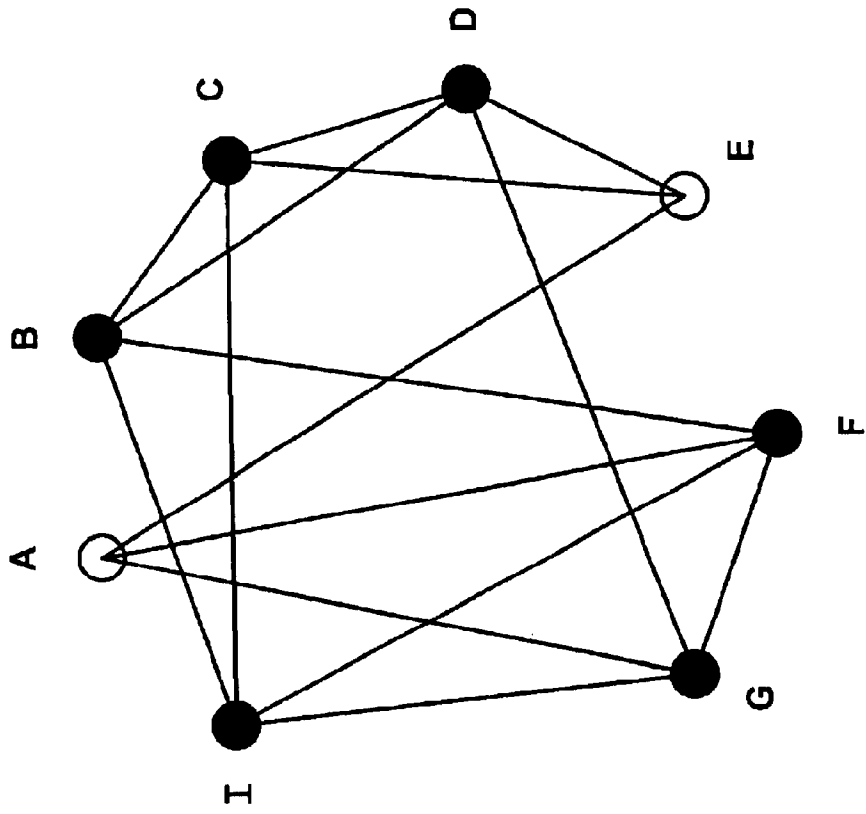


Fig. 5C

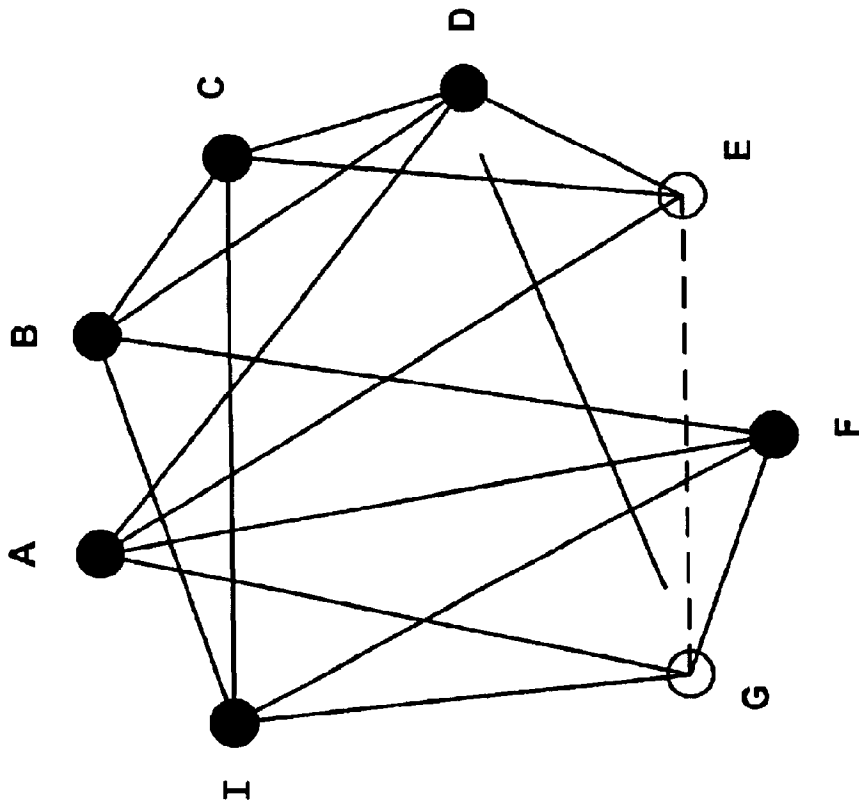


Fig. 5D

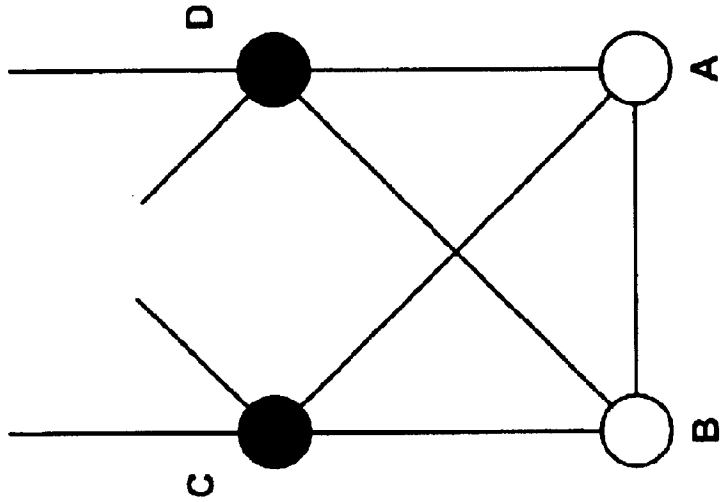


Fig. 5F

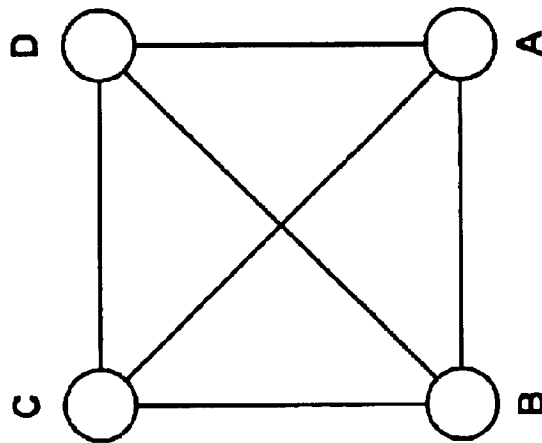


Fig. 5E

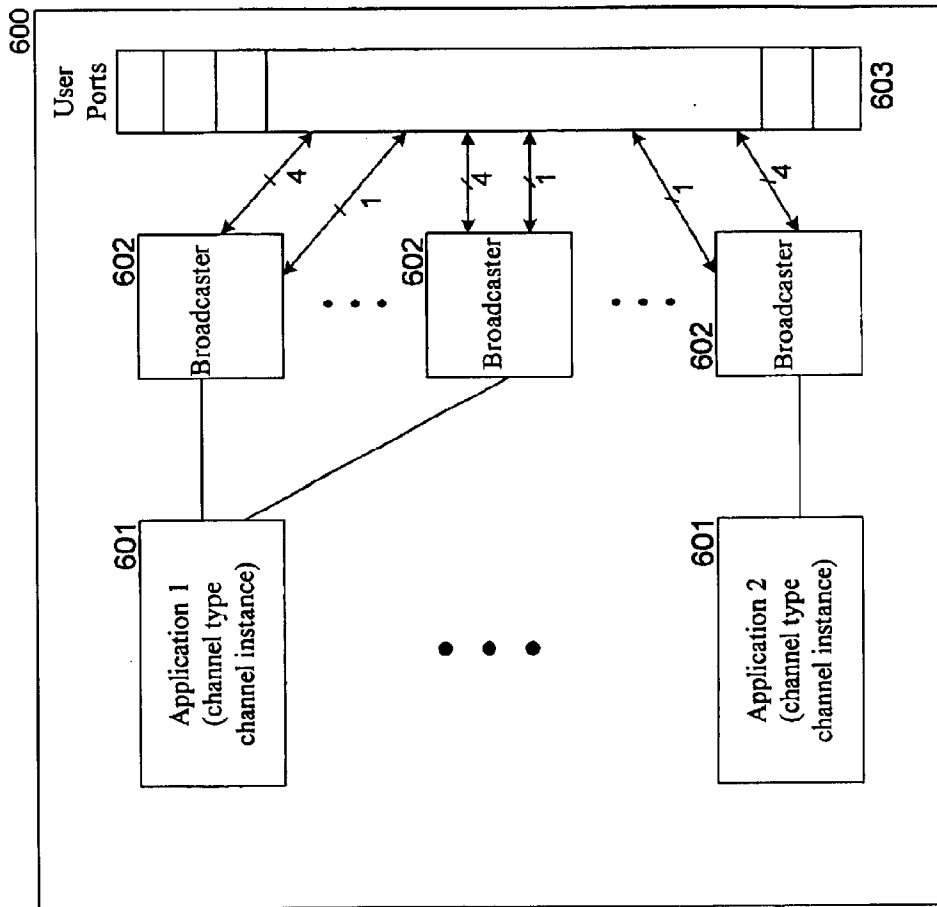


Fig. 6

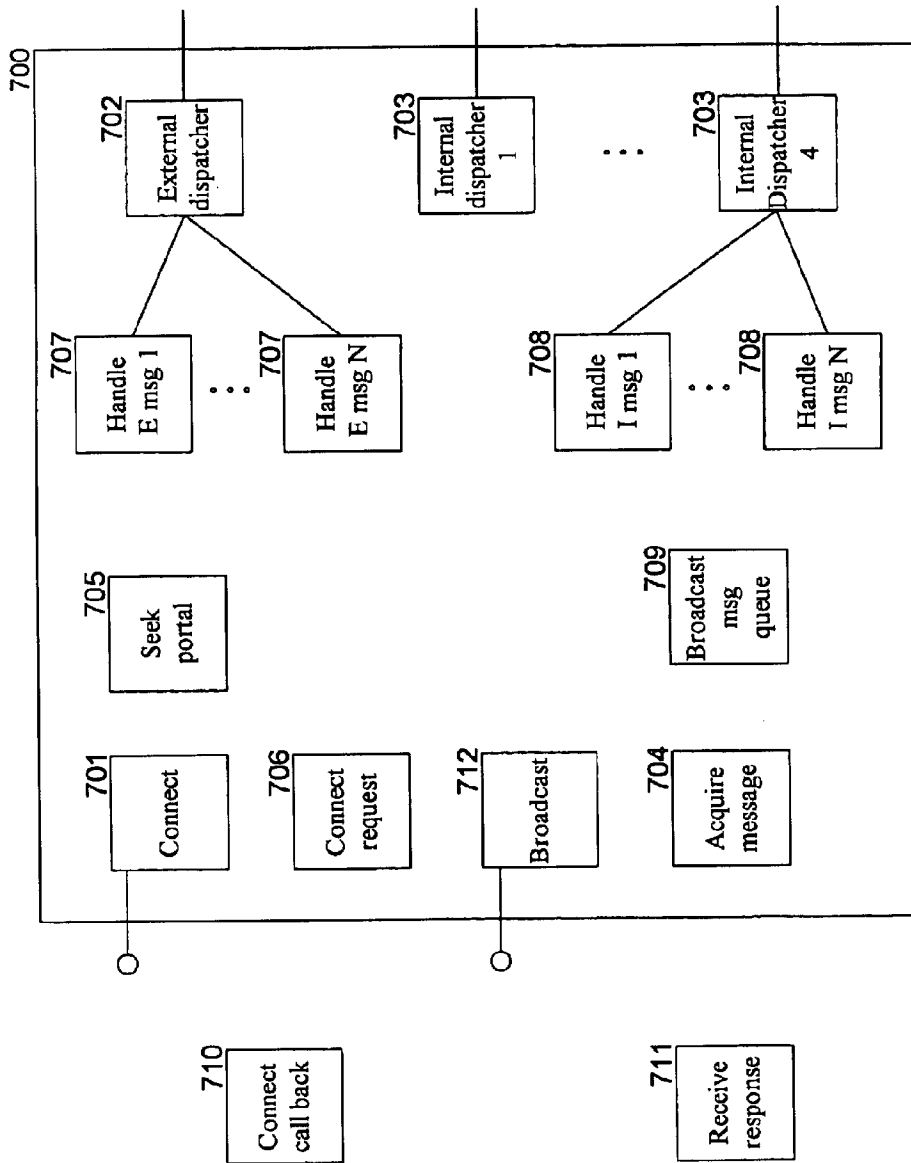
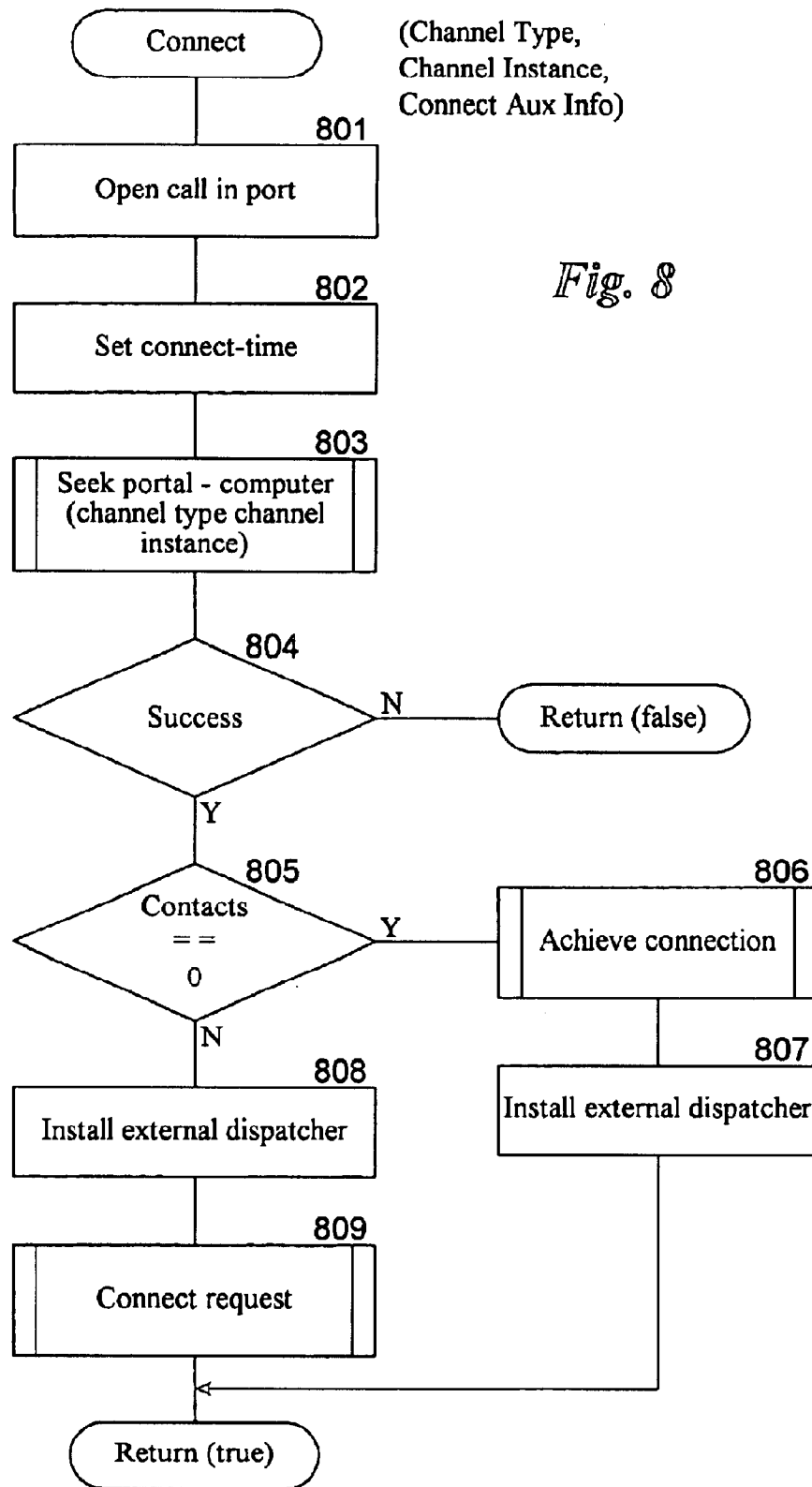


Fig. 7



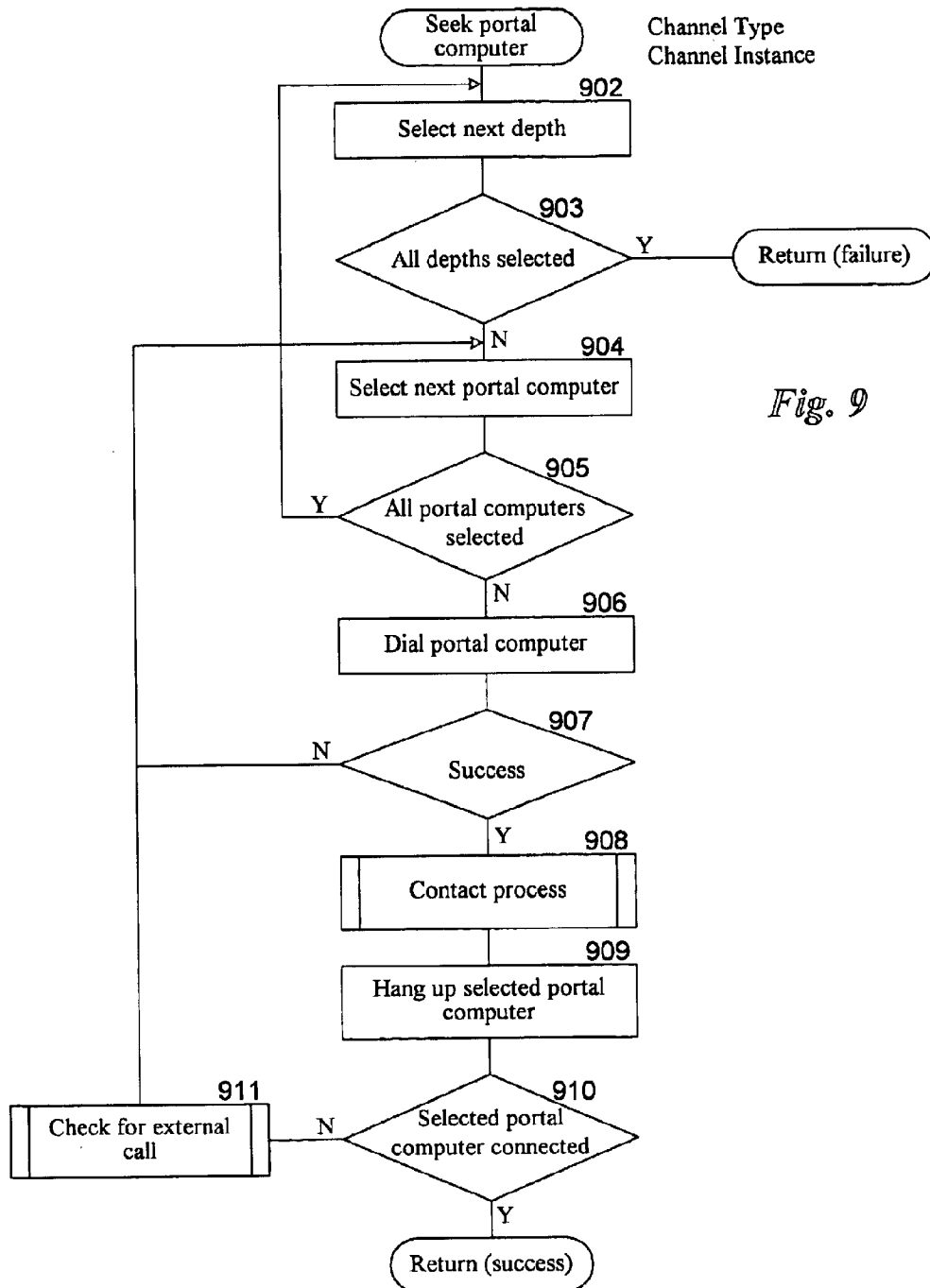


Fig. 9

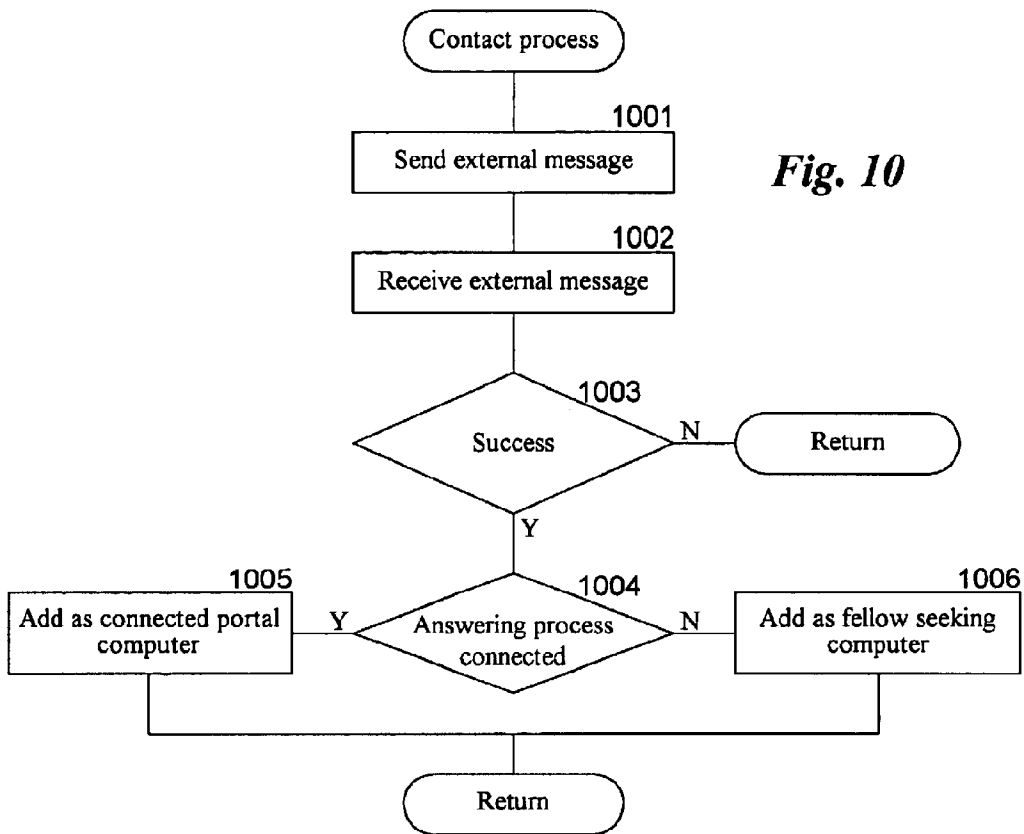


Fig. 11

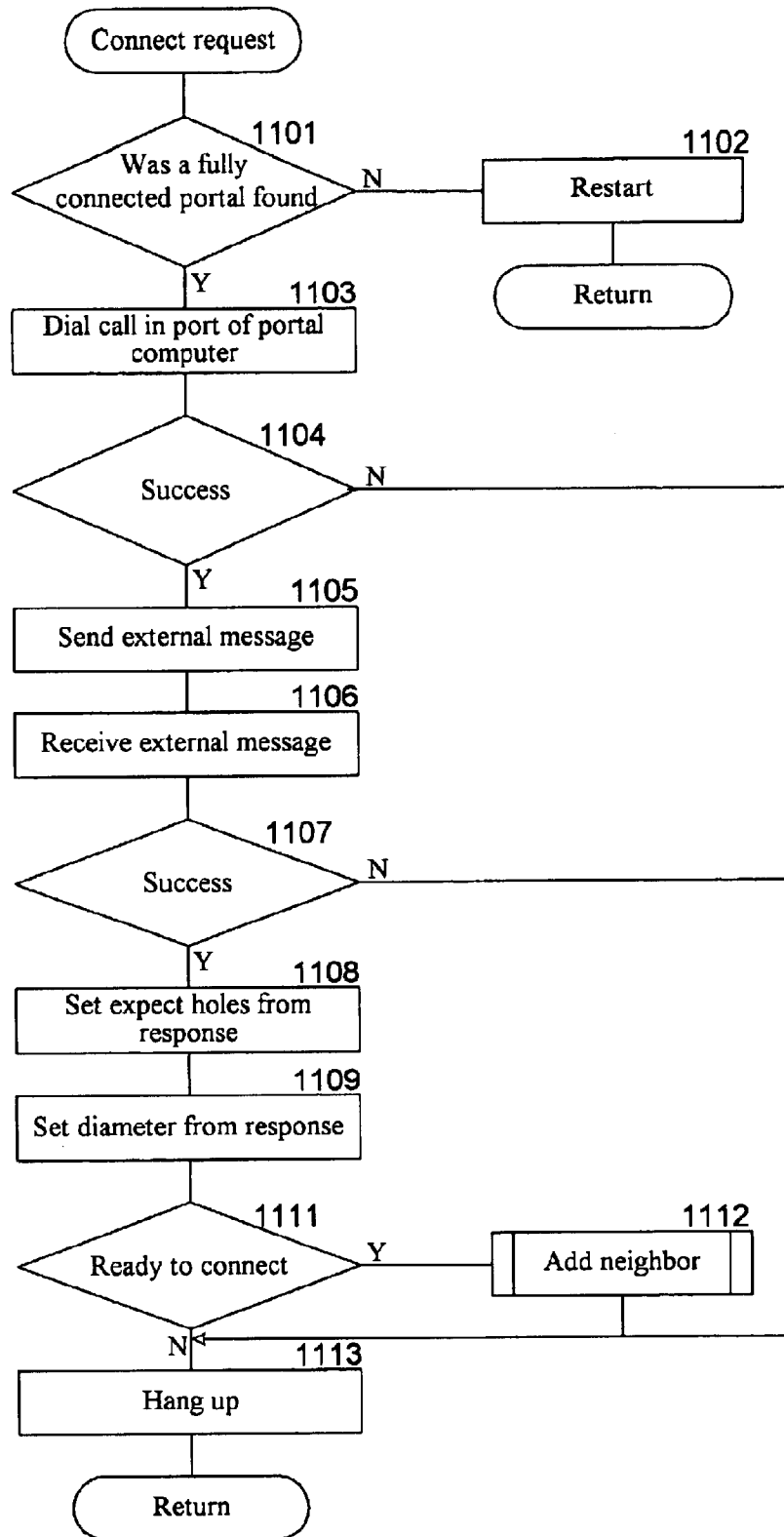
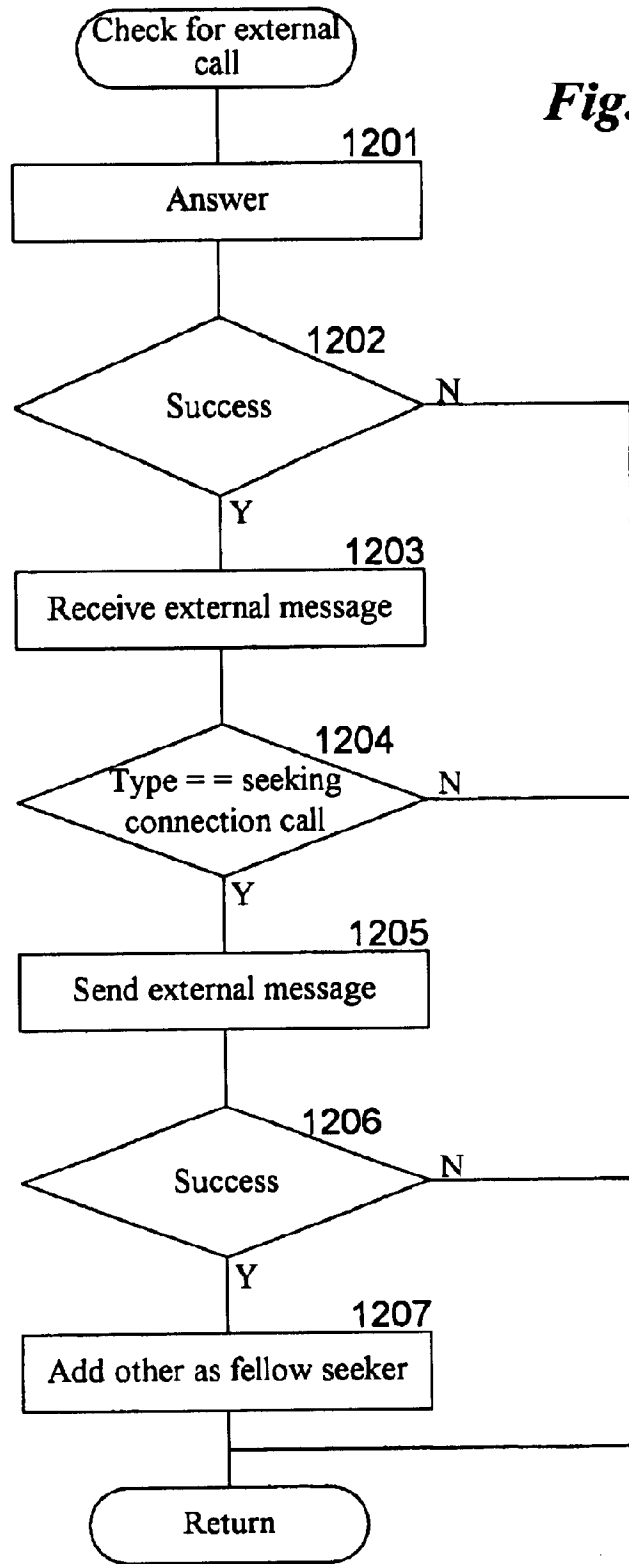


Fig. 12



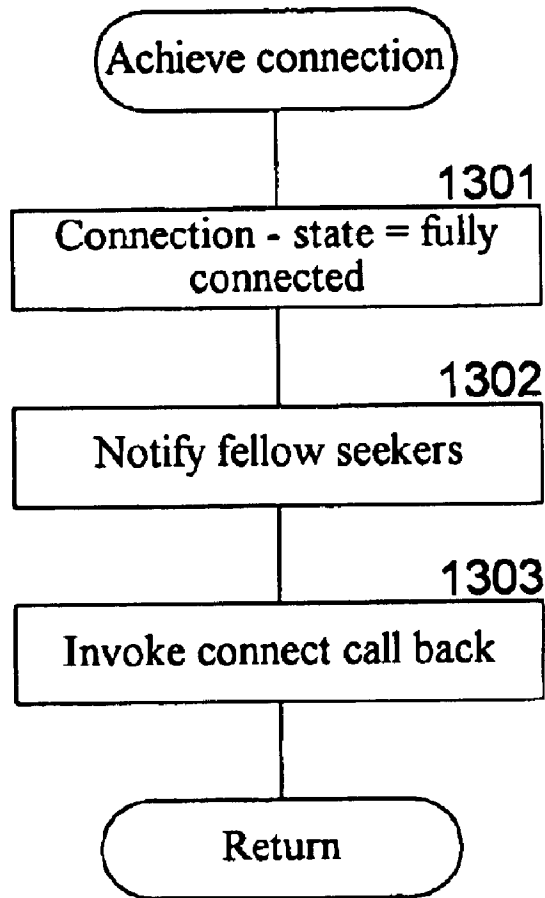
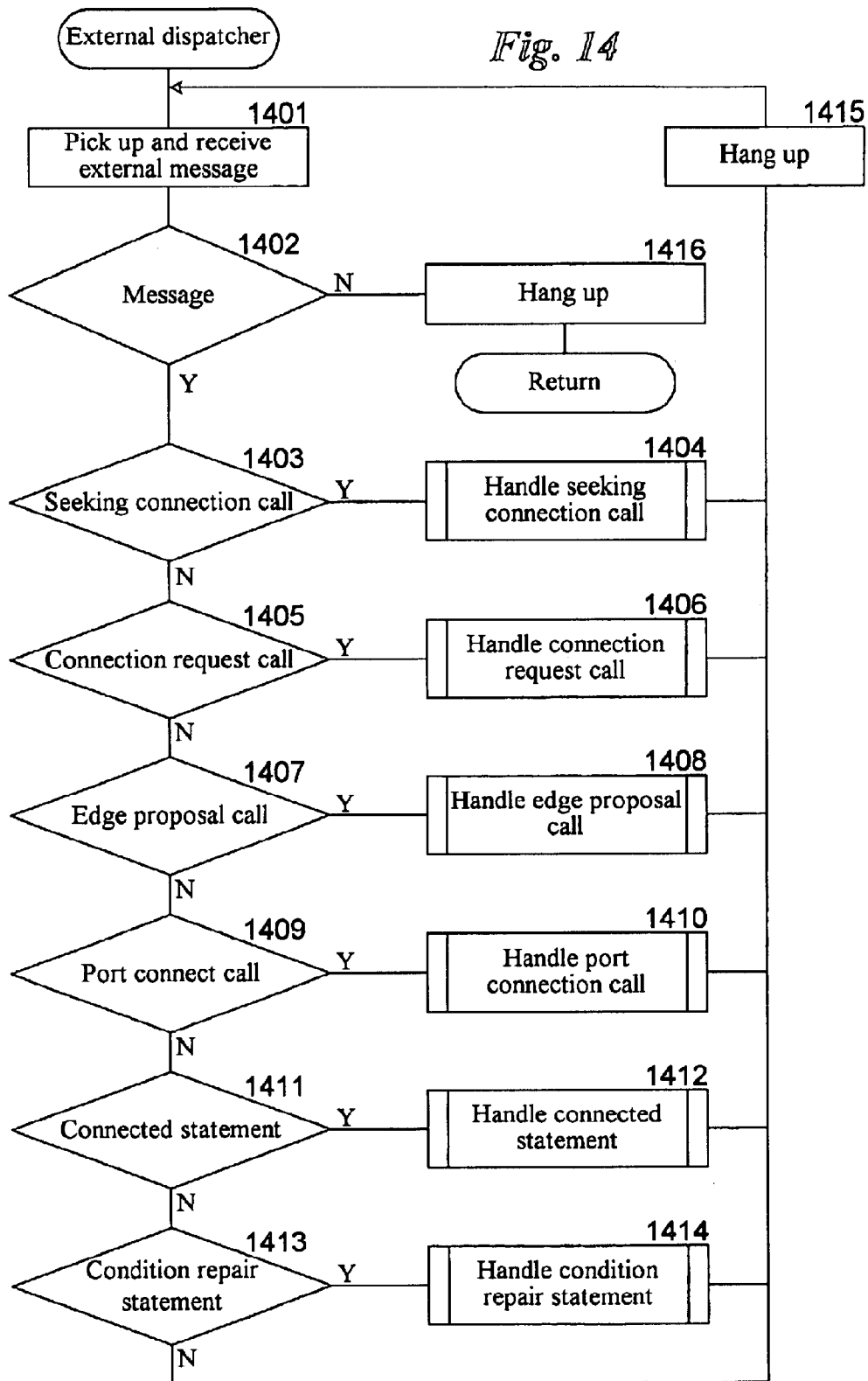
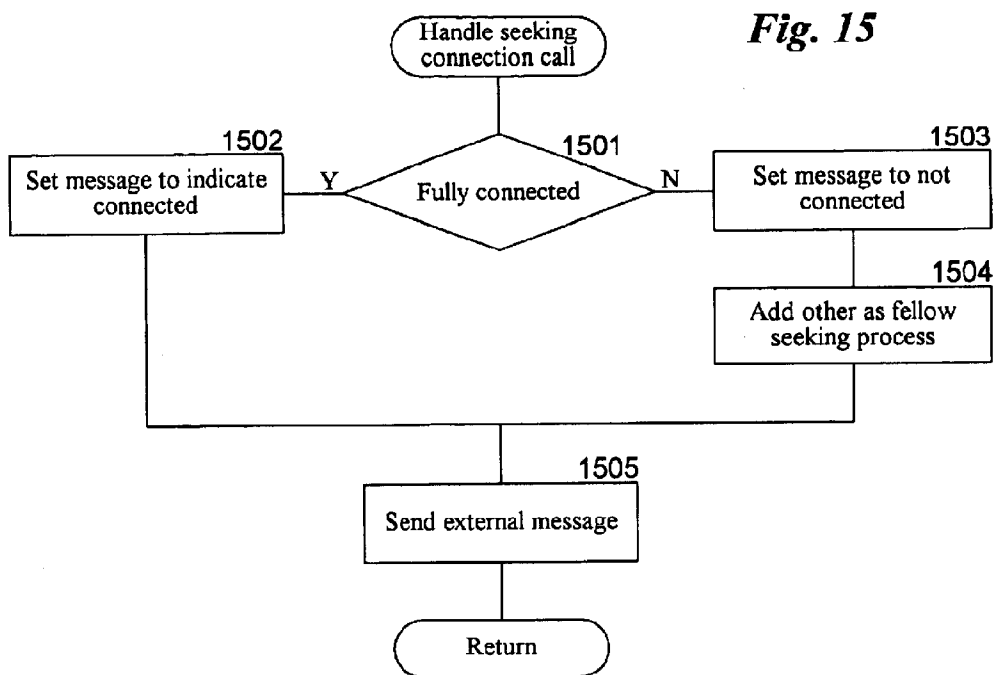


Fig. 13





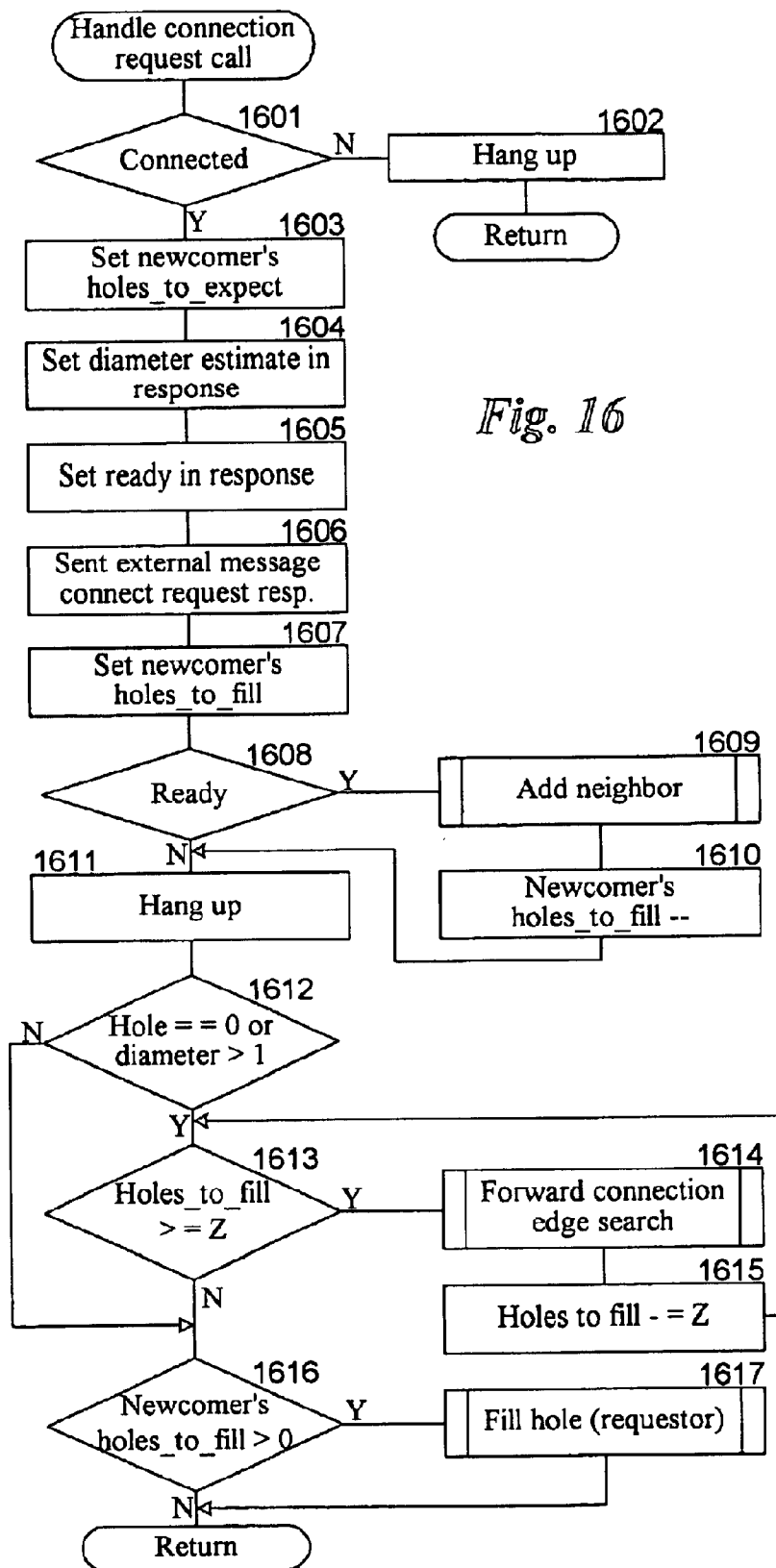


Fig. 16

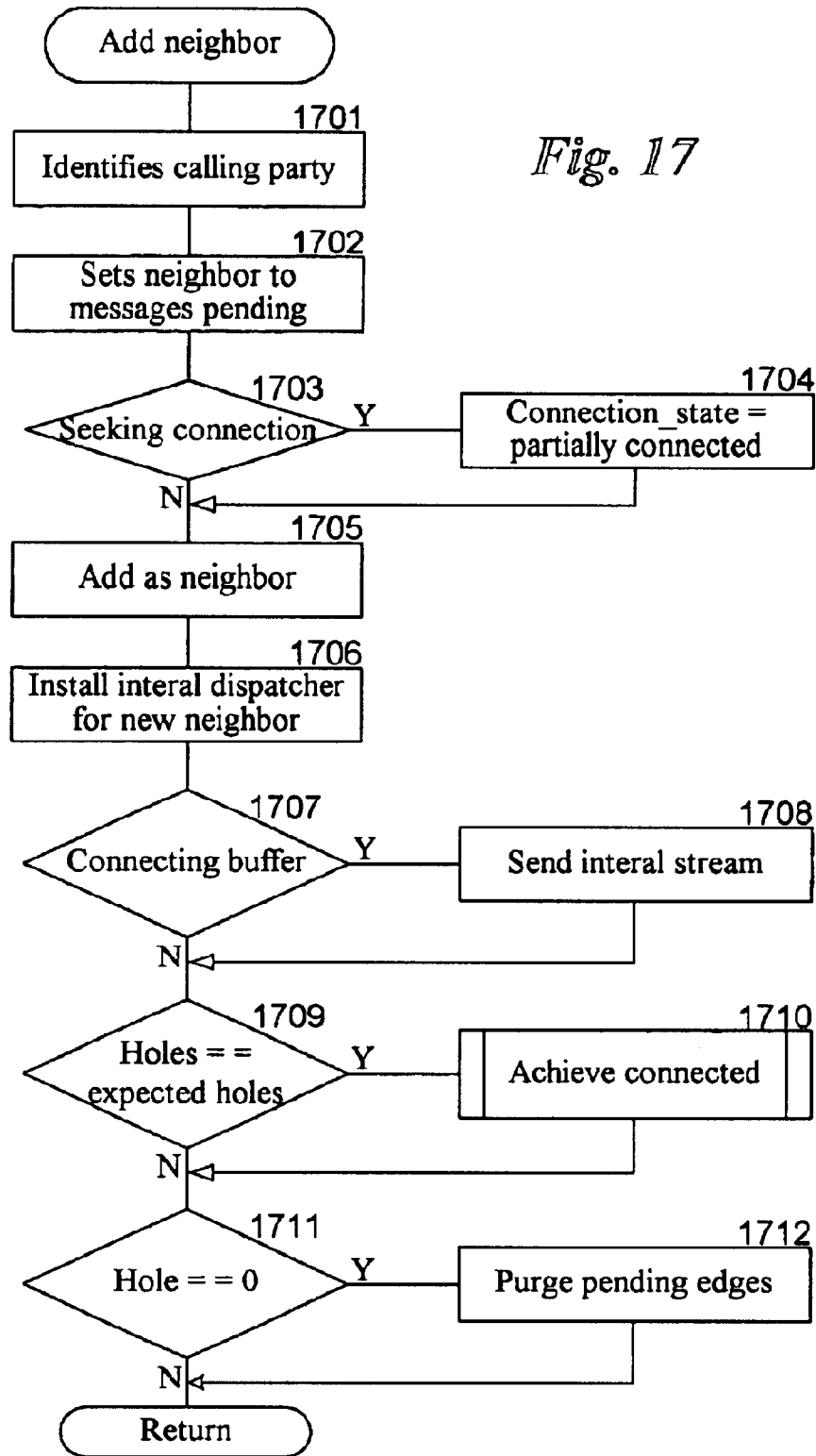
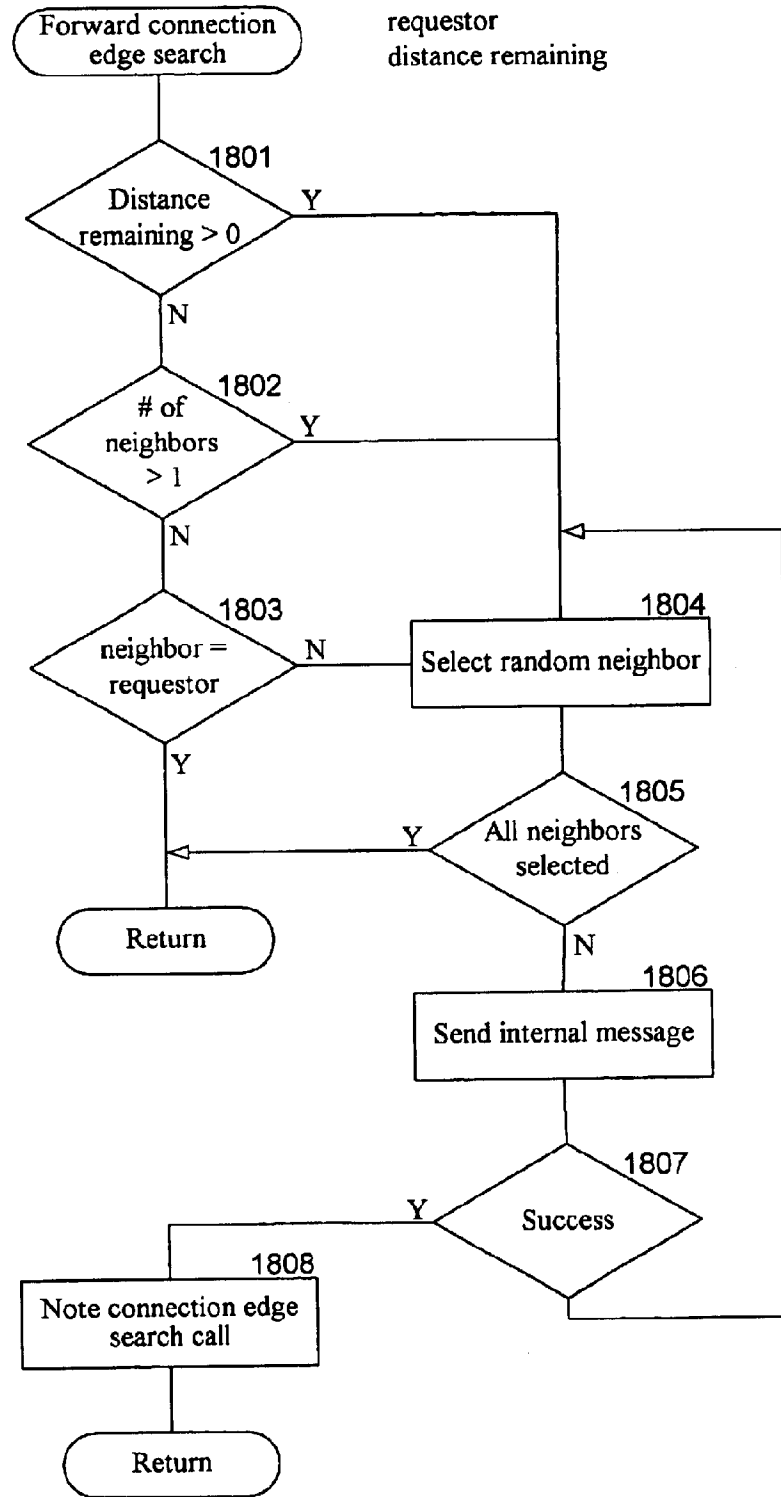
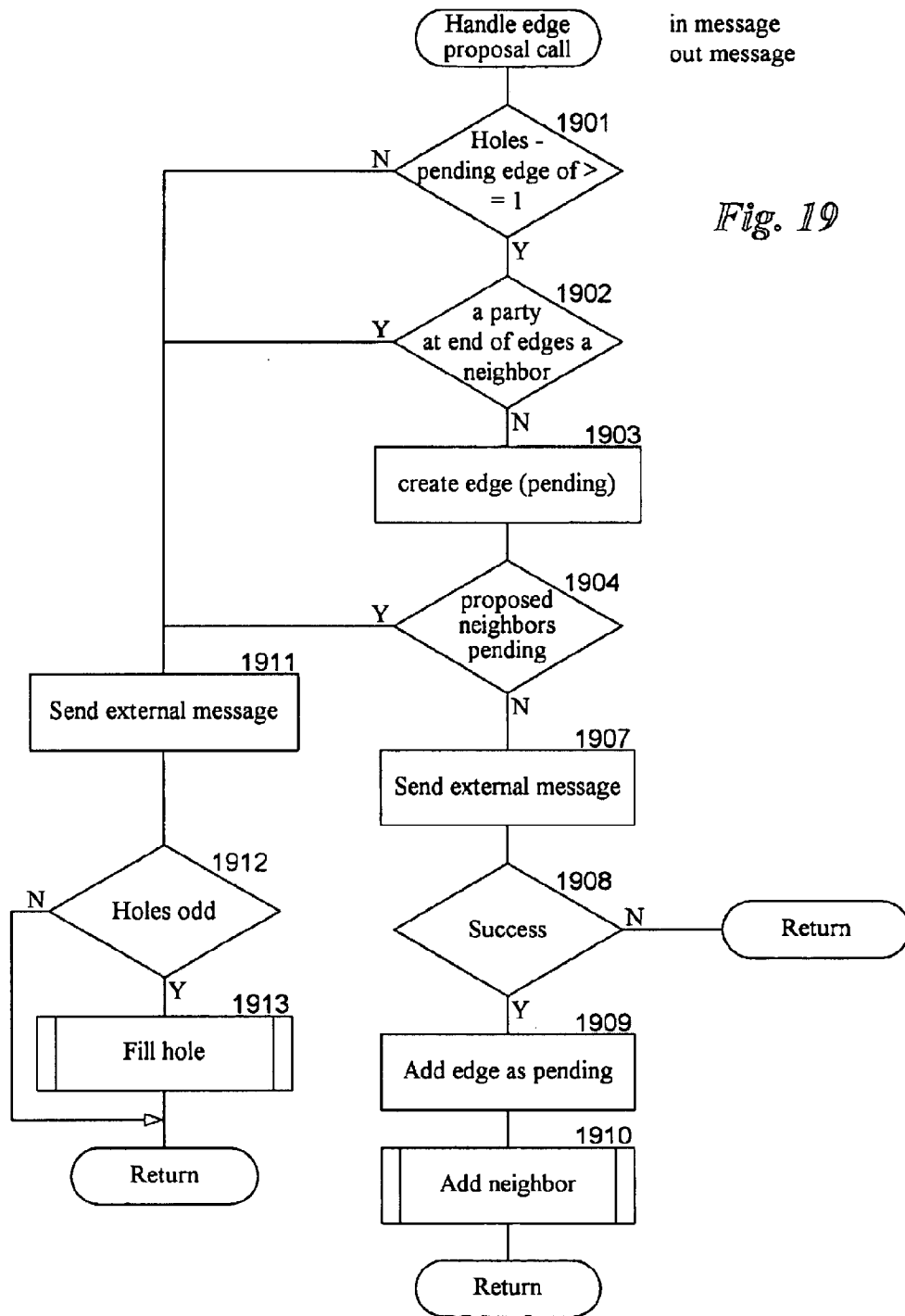


Fig. 18





in message
out message

Fig. 19

Fig. 20

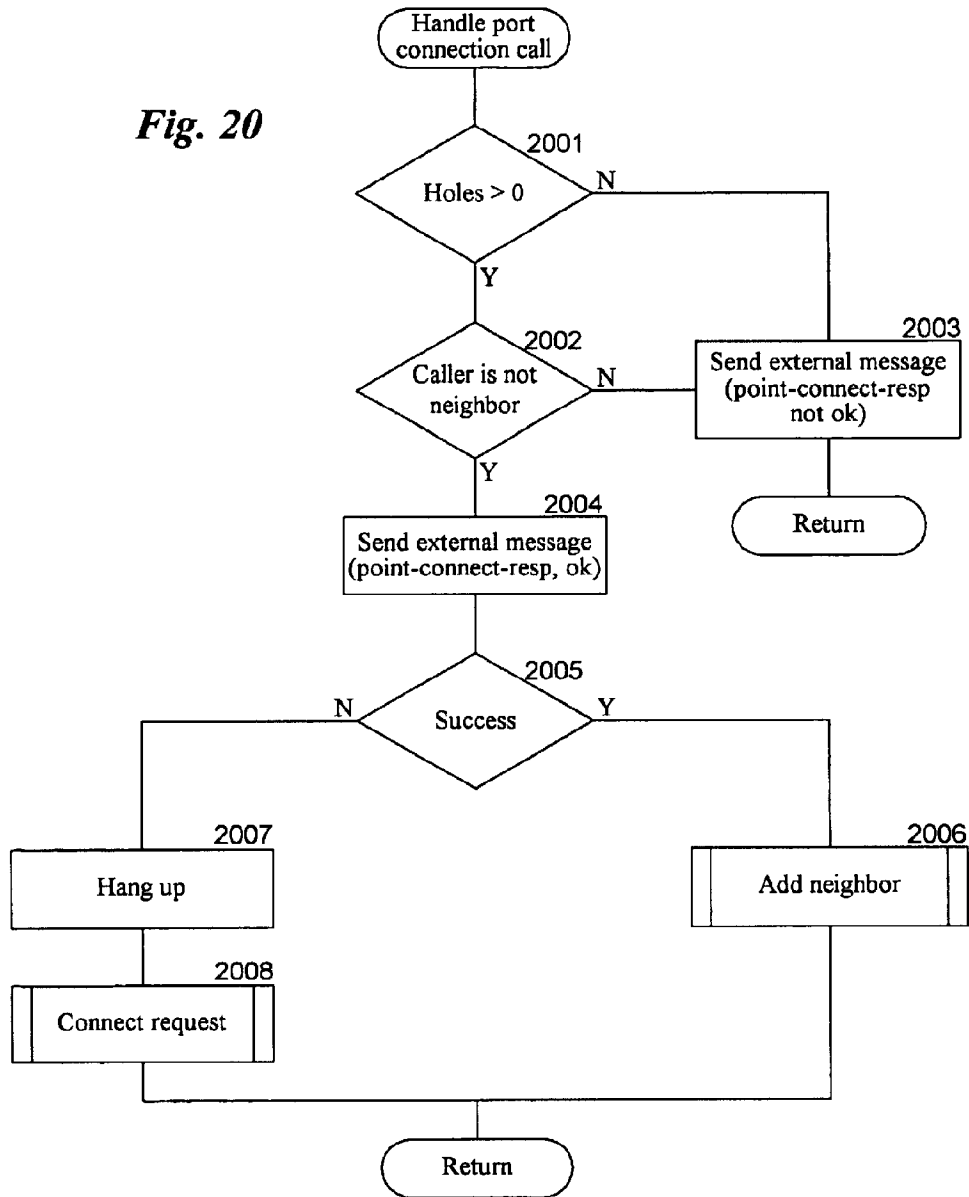


Fig. 21

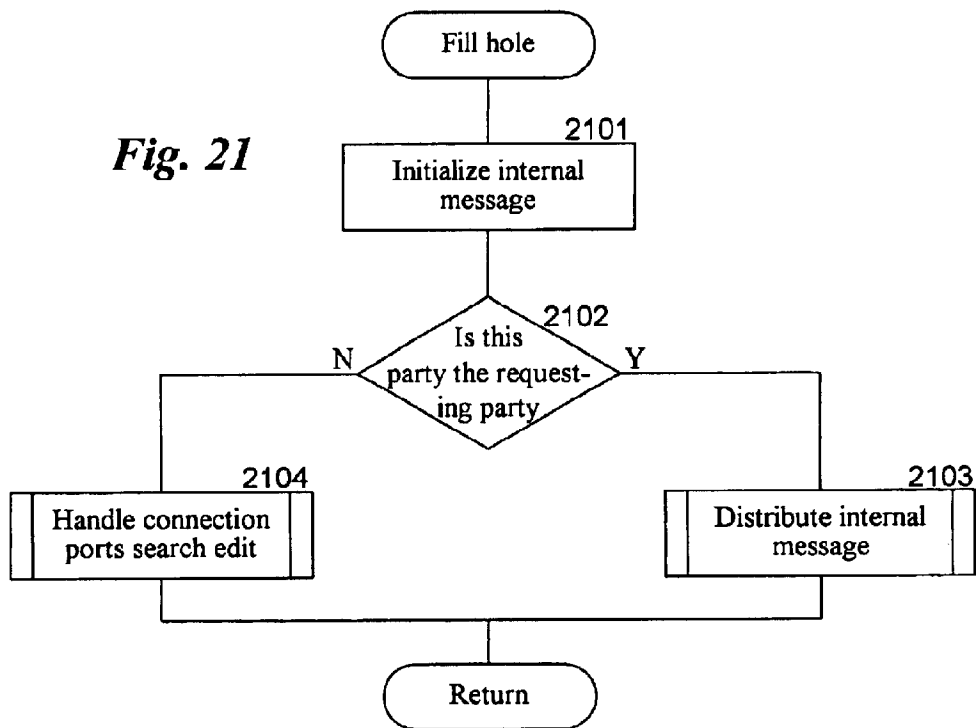


Fig. 22

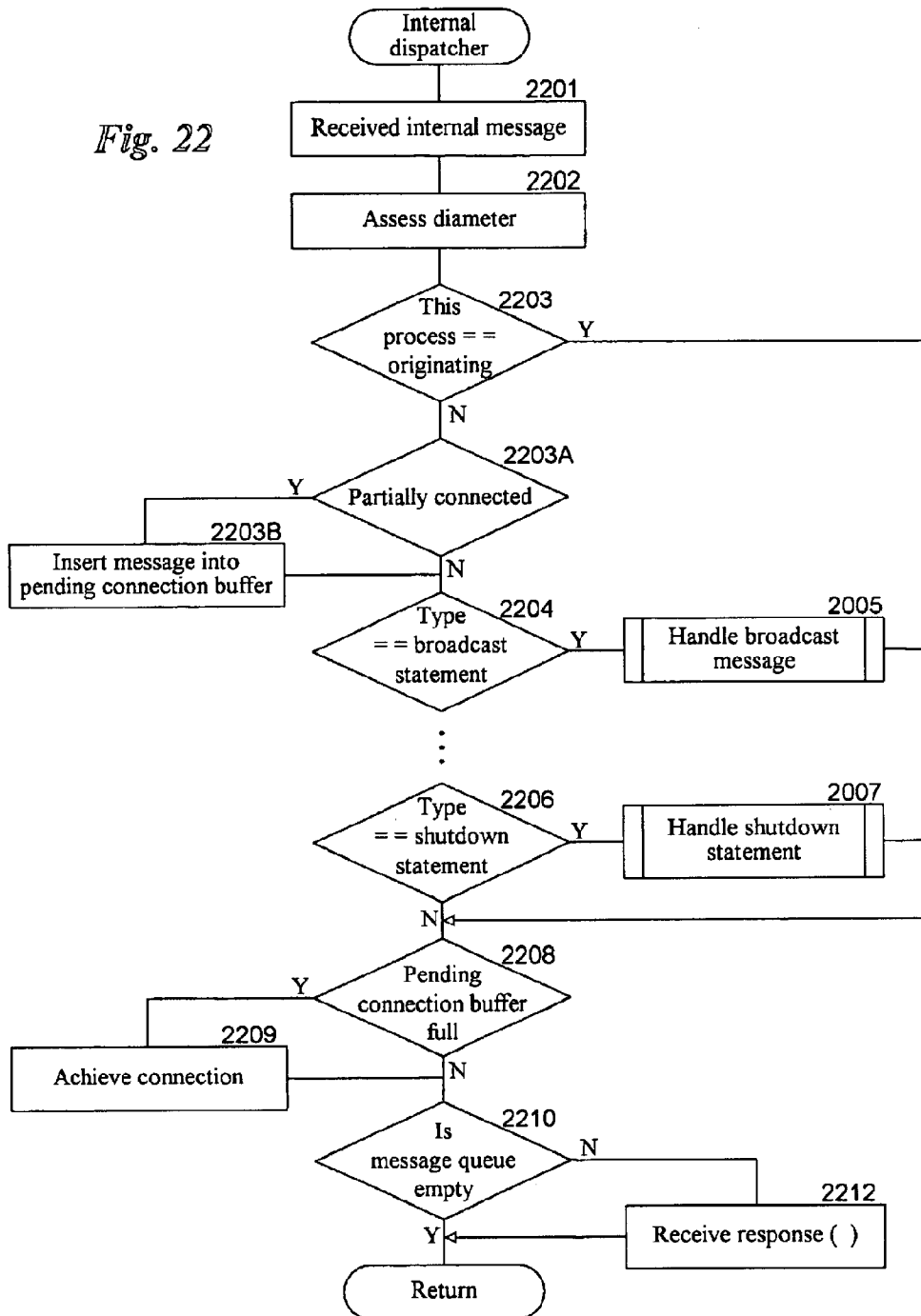


Fig. 23

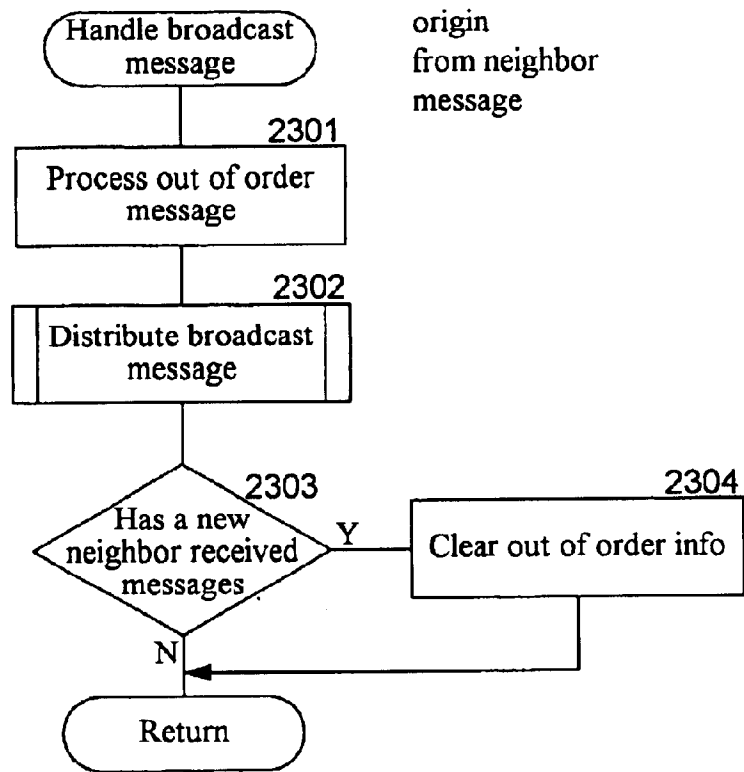
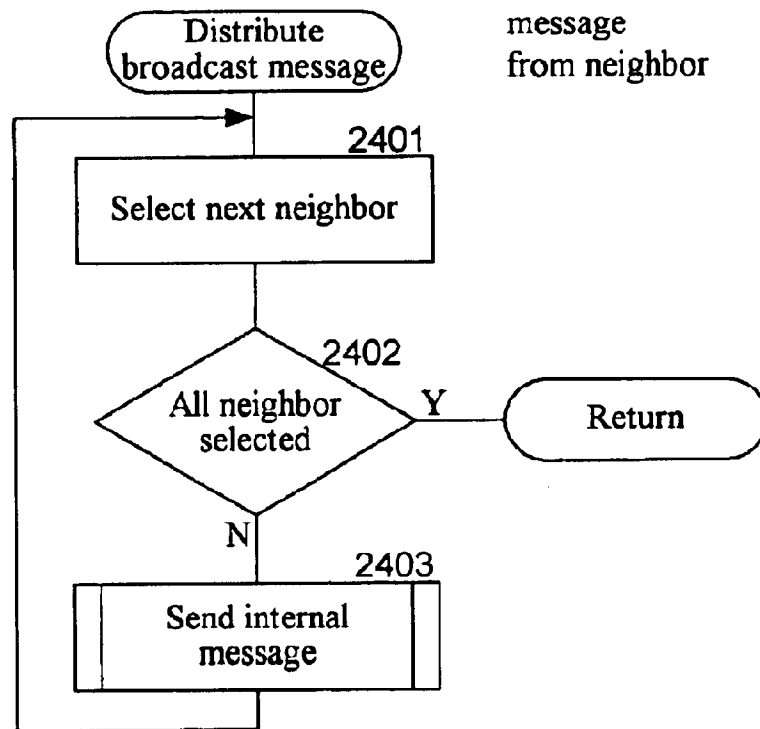
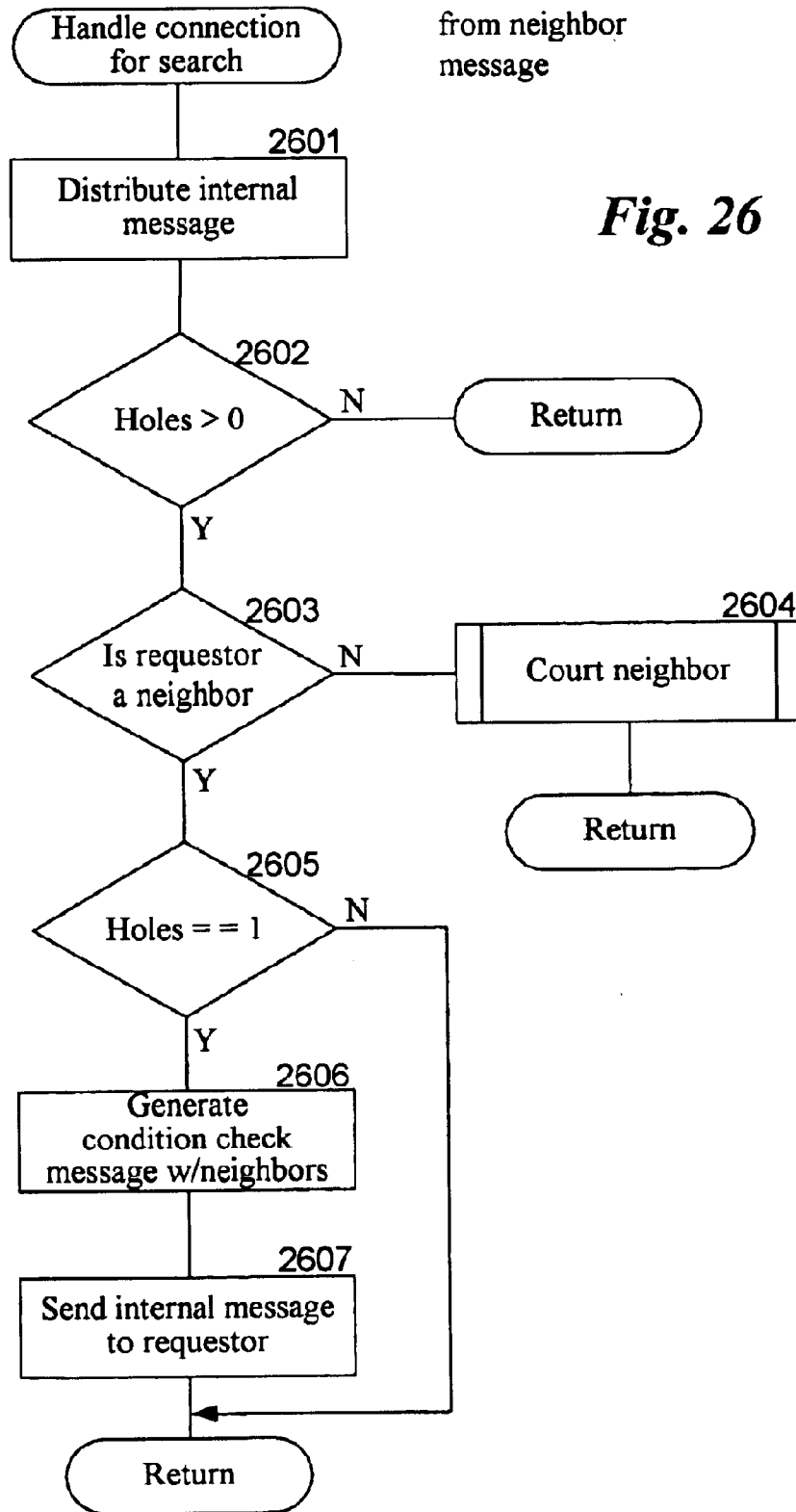


Fig. 24



message
from neighbor



from neighbor message

Fig. 26

Fig. 27

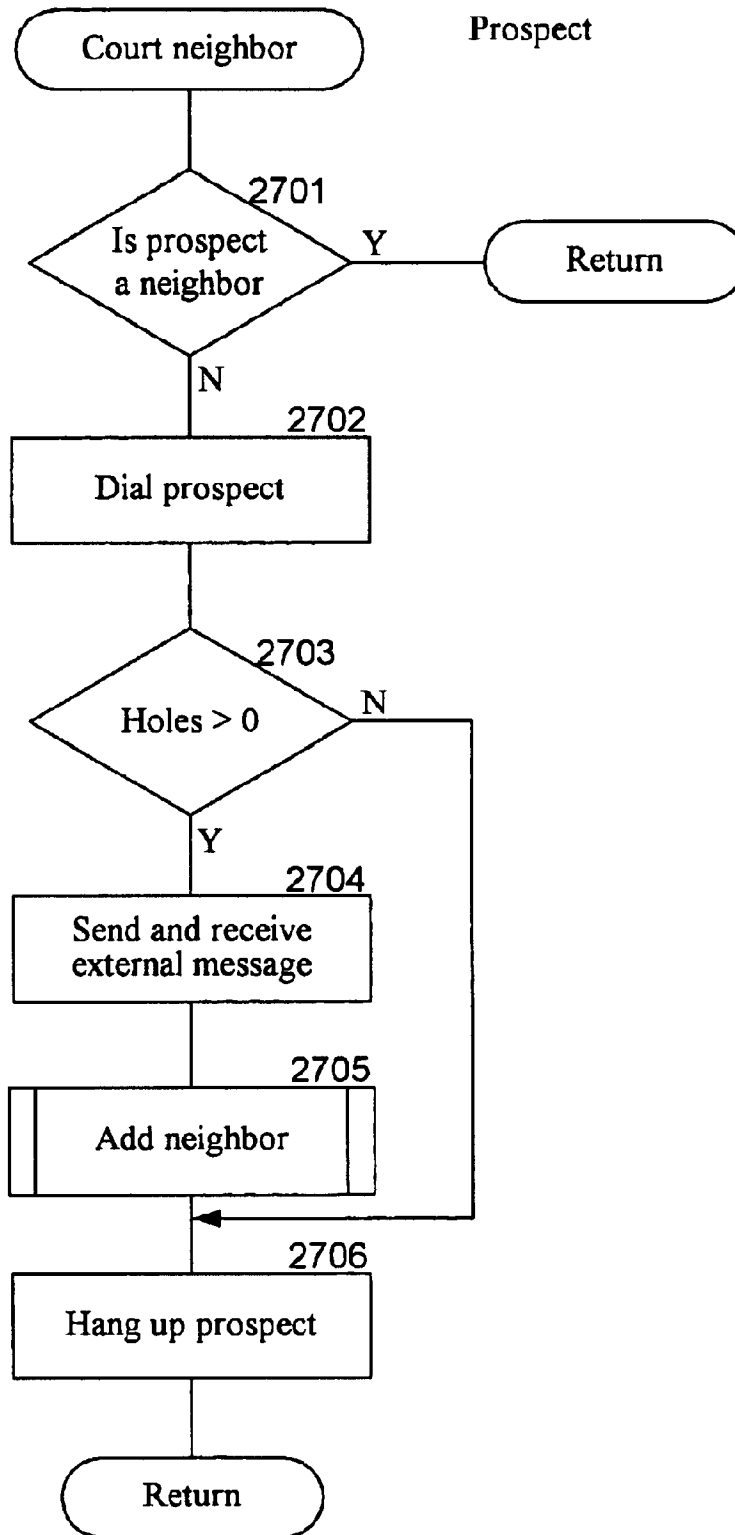


Fig. 28

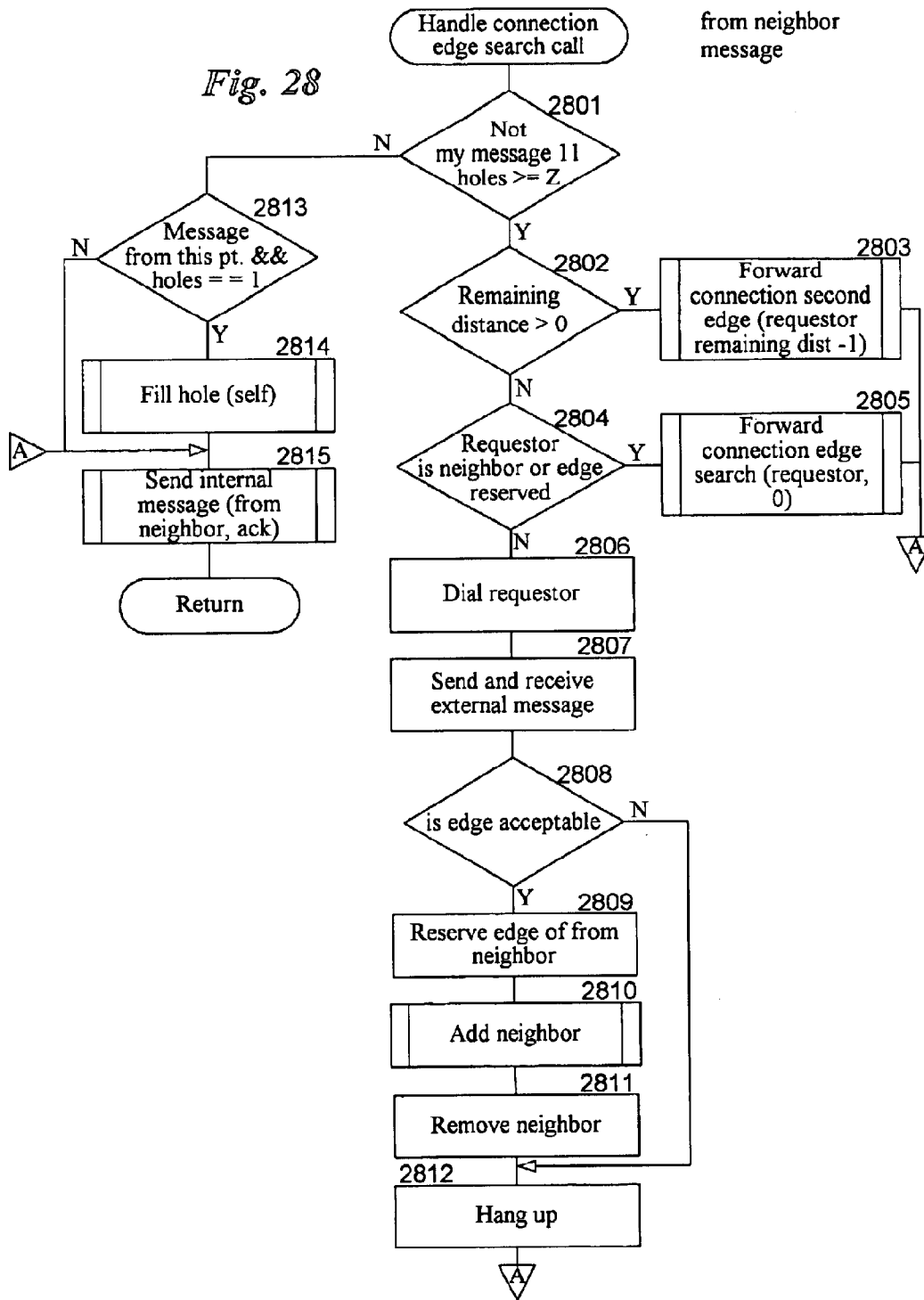


Fig. 29

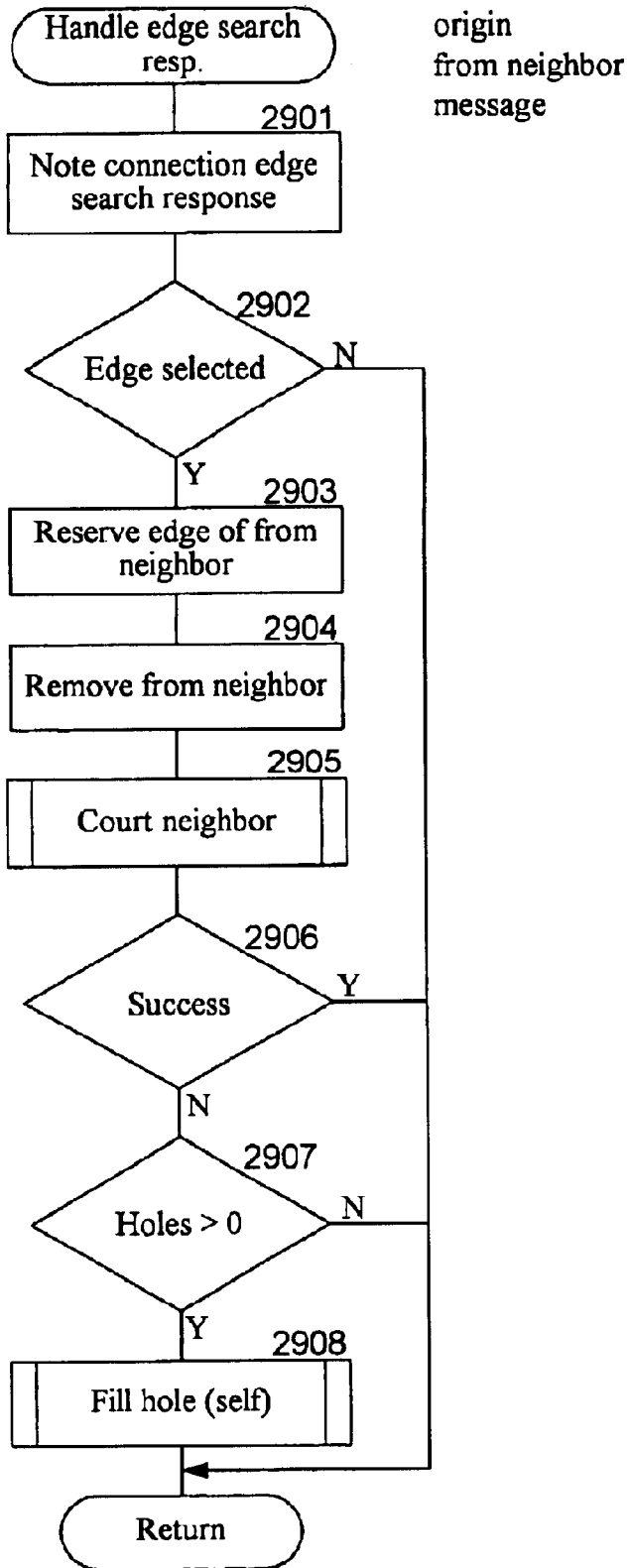


Fig. 30

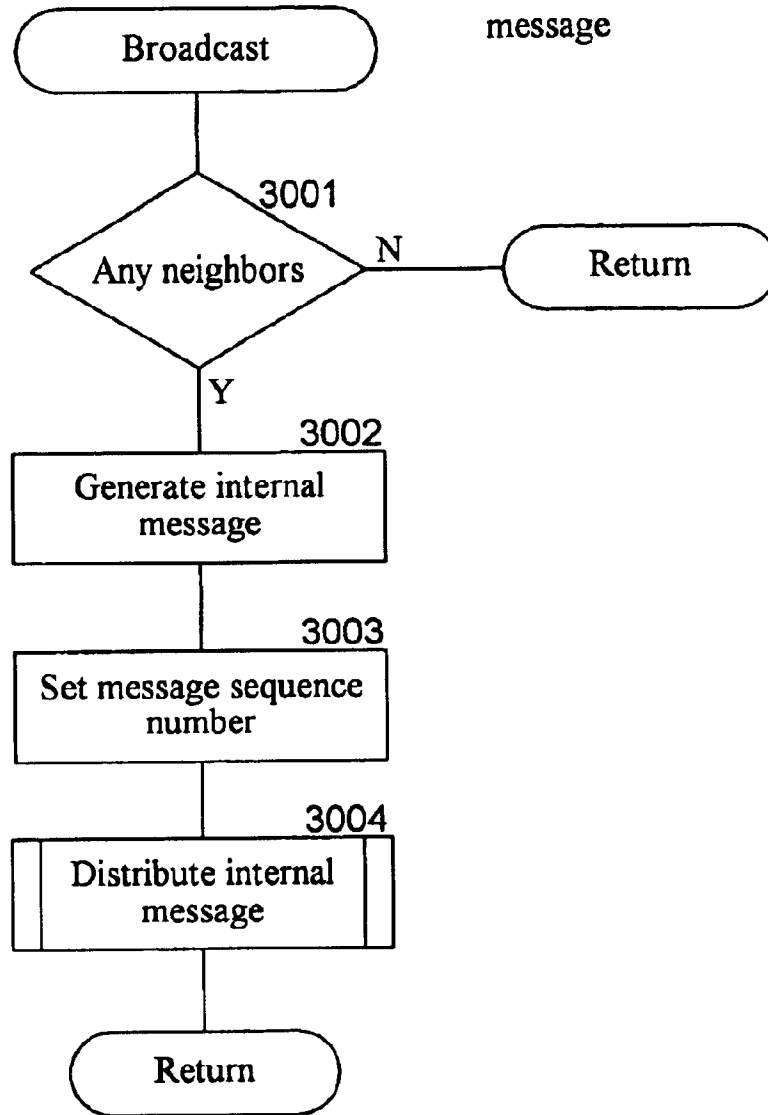


Fig. 31

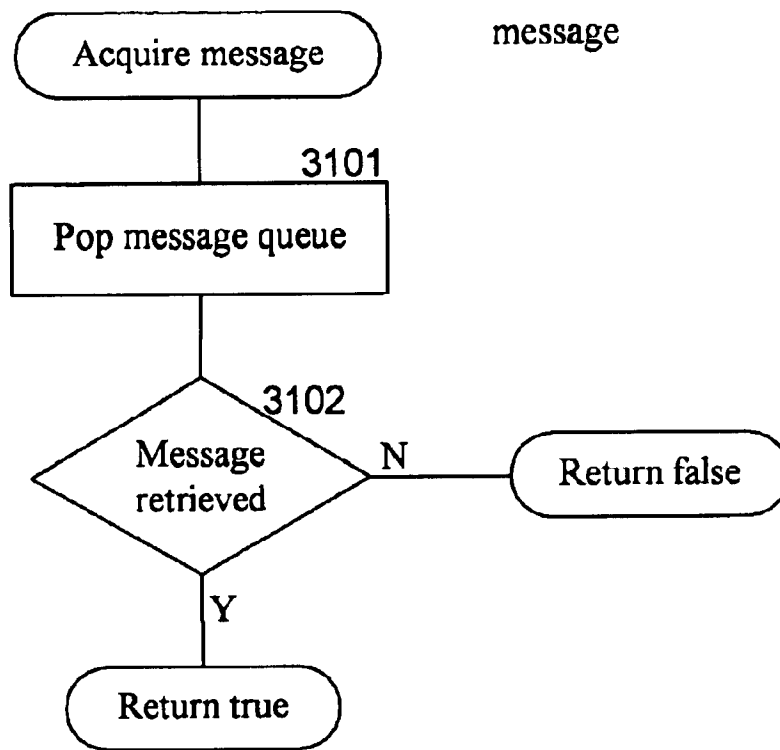


Fig. 32

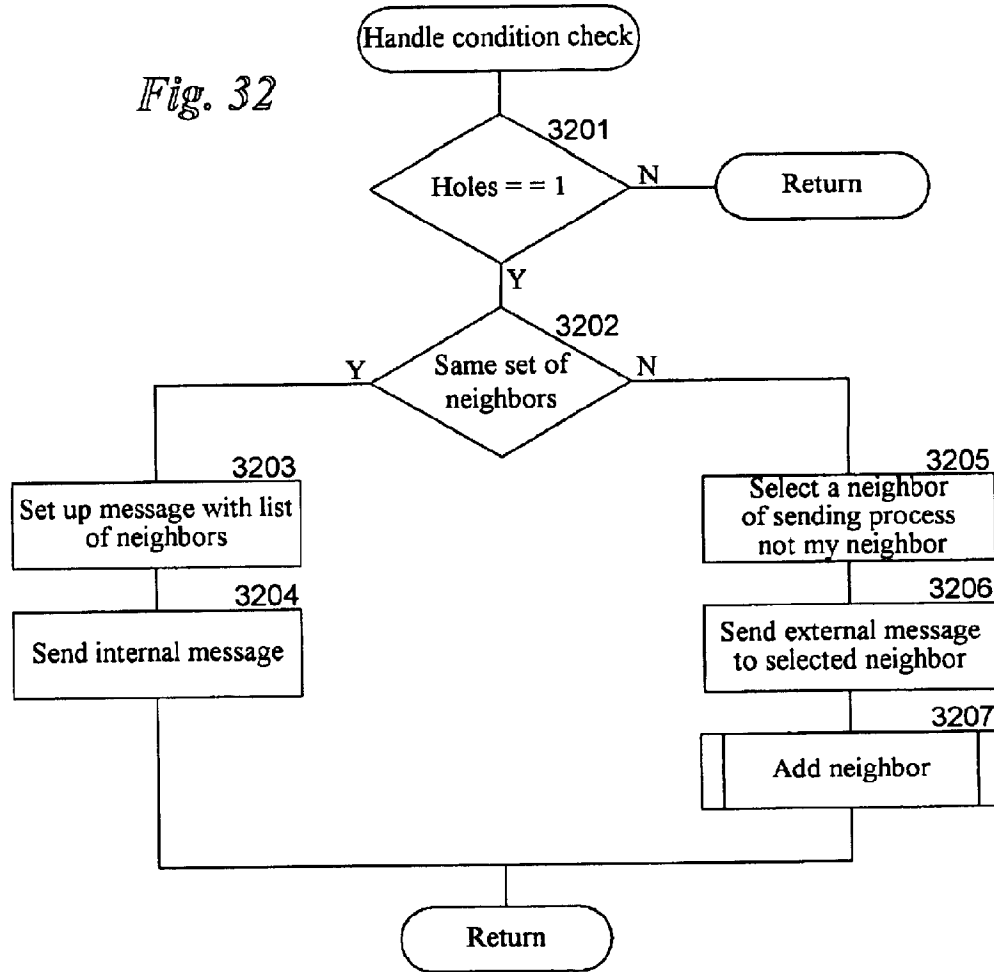


Fig. 33

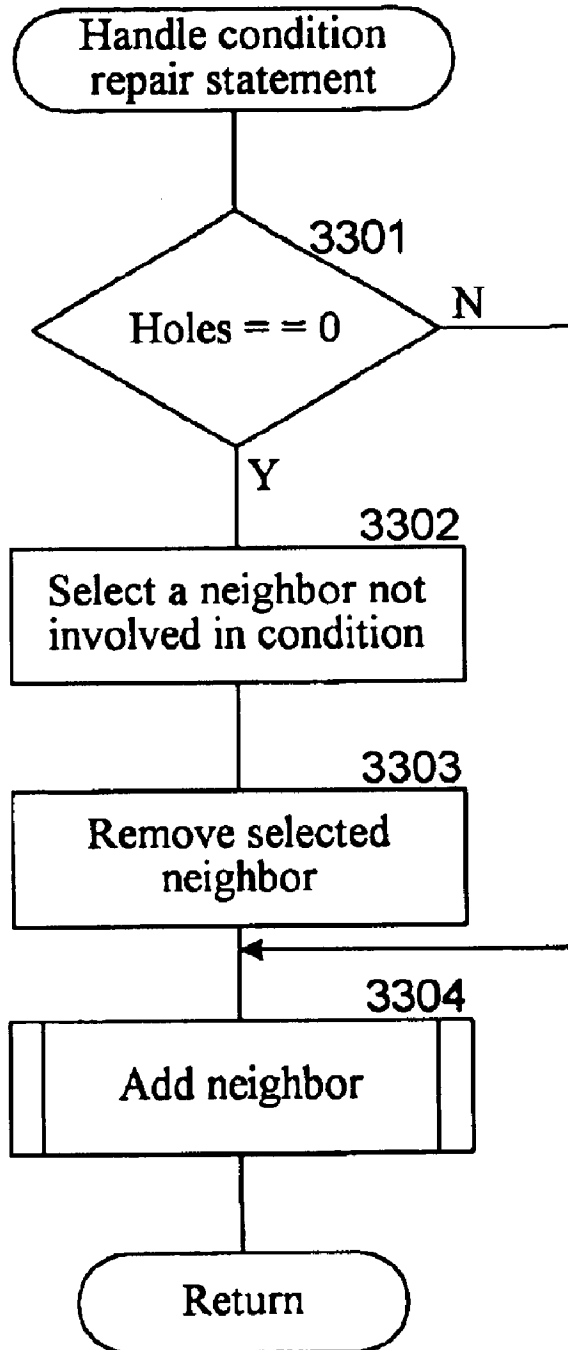
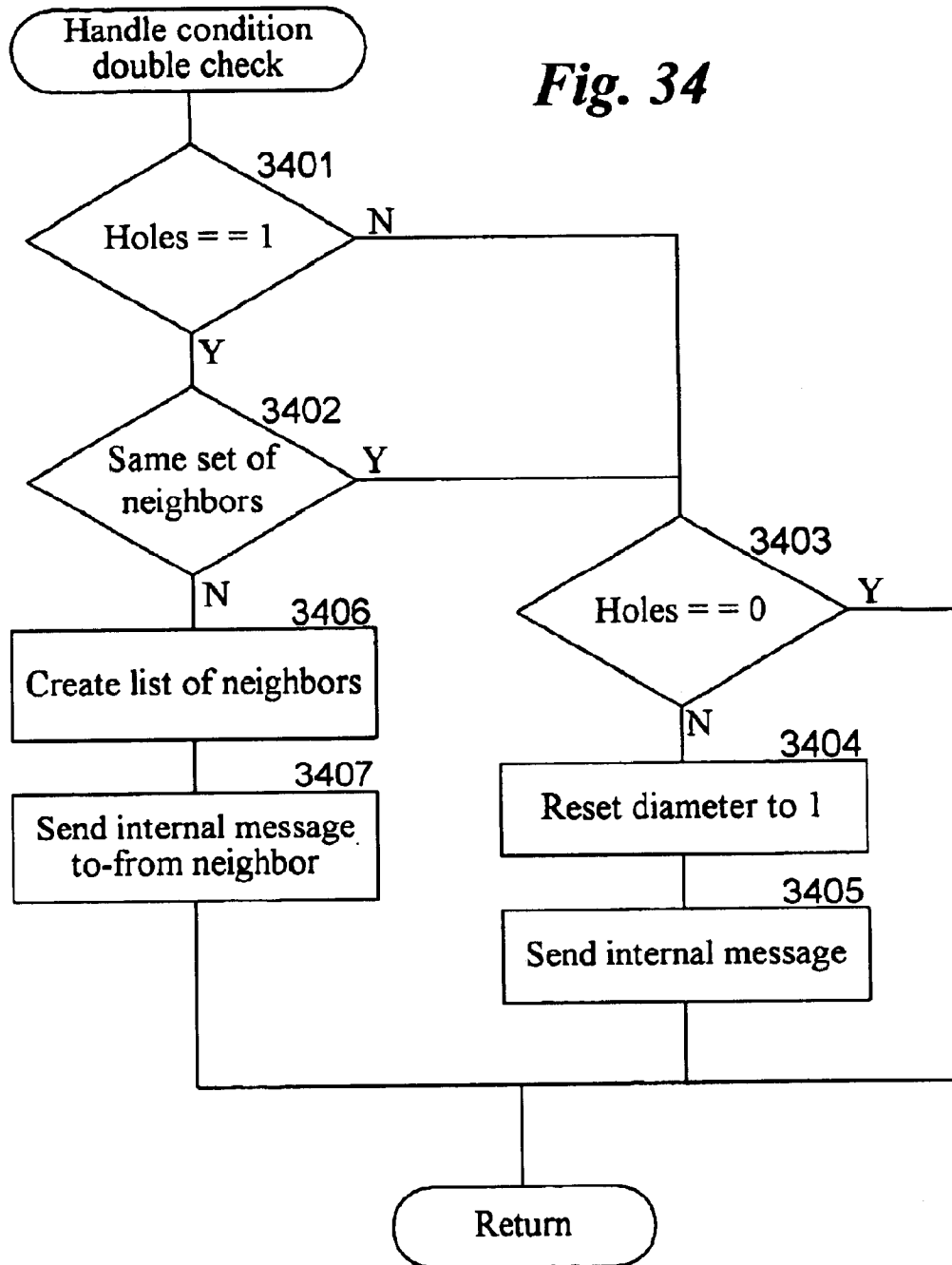


Fig. 34



US 6,701,344 B1

1

DISTRIBUTED GAME ENVIRONMENT**CROSS-REFERENCE TO RELATED APPLICATIONS**

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

TECHNICAL FIELD

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

BACKGROUND

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

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

The client/server middleware systems provide a server that coordinates the communications between the various clients who are sharing the information. The server functions as a central authority for controlling access to shared resources. Examples of client/server middleware systems include remote procedure calls ("RPC"), database servers, and the common object request broker architecture ("CORBA"). Client/server middleware systems are not particularly well suited to sharing of information among many participants. In particular, when a client stores information

2

to be shared at the server, each other client would need to poll the server to determine that new information is being shared. Such polling places a very high overhead on the communications network. Alternatively, each client may register a callback with the server, which the server then invokes when new information is available to be shared. Such a callback technique presents a performance bottleneck because a single server needs to call back to each client whenever new information is to be shared. In addition, the reliability of the entire sharing of information depends upon the reliability of the single server. Thus, a failure at a single computer (i.e., the server) would prevent communications between any of the clients.

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

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

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

BRIEF DESCRIPTION OF THE DRAWINGS

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

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

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

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

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

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

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

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

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

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

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

US 6,701,344 B1

3

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4

DETAILED DESCRIPTION

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

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

FIG. 1 illustrates a graph that is 4-regular and 4-connected which represents the broadcast channel. Each of the nine nodes A-I represents a computer that is connected to the broadcast channel, and each of the edges represents an "edge" connection between two computers of the broadcast channel. The time it takes to broadcast a message to each computer on the broadcast channel depends on the speed of the connections between the computers and the number of connections between the originating computer and each other computer on the broadcast channel. The minimum number of connections that a message would need to traverse between each pair of computers is the "distance" between the computers (i.e., the shortest path between the two nodes of the graph). For example, the distance between computers A and F is one because computer A is directly connected to computer F. The distance between computers A and B is two because there is no direct connection between computers A and B, but computer F is directly connected to computer B. Thus, a message originating at computer A would be sent directly to computer F, and then sent from

US 6,701,344 B1

5

computer F to computer B. The maximum of the distances between the computers is the “diameter” of broadcast channel. The diameter of the broadcast channel represented by FIG. 1 is two. That is, a message sent by any computer would traverse no more than two connections to reach every other computer. FIG. 2 illustrates a graph representing 20 computers connected to a broadcast channel. The diameter of this broadcast channel is 4. In particular, the shortest path between computers 1 and 3 contains four connections (1-12, 12-15, 15-18, and 18-3).

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

Composing the Graph

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

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

6

nected. The pairs of computers B and E and computers C and D are the two pairs that are identified as the neighbors for the new computer Z. The connections between each of these pairs is broken, and a connection between computer Z and each of computers B, C, D, and E is established as indicated by FIG. 3B. The process of breaking the connection between two neighbors and reconnecting each of the former neighbors to another computer is referred to as “edge pinning” as the edge between two nodes may be considered to be stretched and pinned to a new node.

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

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

A seeking computer could connect to the broadcast channel by connecting to computers either directly connected to the found portal computer or directly connected to one of its neighbors. A possible problem with such a scheme for

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

Broadcasting Through the Graph

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

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

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

message may have to travel a distance of four to reach the receiving computer. The second message only has to travel a distance of one. Thus, it is possible for the second message to reach the receiving computer before the first message.

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

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

US 6,701,344 B1

9

Decomposing the Graph

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

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

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

10

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

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

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

US 6,701,344 B1

11

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

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

Port Selection

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

12

and only selects each port number once. In addition, every time the algorithm is executed on any computer for a given channel type and channel instance, it generates the same port ordering. As described below, it is possible for a computer to be connected to multiple broadcast channels that are uniquely identified by channel type and channel instance. The algorithm may be "seeded" with channel type and channel instance in order to generate a unique ordering of port numbers for each broadcast channel. Thus, a seeking computer will dial the ports of a portal computer in the same order as the portal computer used when allocating its call-in port.

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

Locating a Portal Computer

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

When a seeking computer locates a portal computer that is itself not fully connected, the two computers do not

US 6,701,344 B1

13

connect when they first locate each other because the broadcast channel may already be established and accessible through a higher-ordered port number on another portal computer. If the two seeking computers were to connect to each other, then two disjoint broadcast channels would be formed. Each seeking computer can share its experience in trying to locate a portal computer with the other seeking computer. In particular, if one seeking computer has searched all the portal computers to a depth of eight, then the one seeking computer can share that it has searched to a depth of eight with another seeking computer. If that other seeking computer has searched to a depth of, for example, only four, it can skip searching through depths five through eight and that other seeking computer can advance its searching to a depth of nine.

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

Identifying Neighbors for a Seeking Computer

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

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

In one embodiment, the distance that the edge connection request message travels is established by the portal computer to be approximately twice the estimated diameter of the broadcast channel. The message includes an indication of the distance that it is to travel. Each receiving computer decrements that distance to travel before sending the message on. The computer that receives a message with a distance to travel that is zero is considered to be the randomly selected computer. If that randomly selected computer cannot connect to the seeking computer (e.g., because it is already connected to it), then that randomly selected computer forwards the edge connection request to one of its

14

neighbors with a new distance to travel. In one embodiment, the forwarding computer toggles the new distance to travel between zero and one to help prevent two computers from sending the message back and forth between each other.

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

External Data Representation

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

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

M-Regular

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

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

Components

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

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

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

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

A Distributed Game Environment

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

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

The game environment may provide a game web site through which players can view the state of current games and register new games. The game web server would include a mapping between each game and the broadcast channel on which the game is to be played. When joining a game, the user would download the broadcaster component and the game application program from the web server. The user would also download the description of the game, which may include the graphics for the game. The web server would also provide the channel type and channel instance associated with the game and the identification of the portal

computers for the game. The game environment may also have a game monitor computer that connects to each game, monitors the activity of the game, and reports the activity to the web server. With this activity information, the web server can provide information on the current state (e.g., number of players) of each game.

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

The following tables list messages sent by the broadcaster components.

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

INTERNAL MESSAGES	
Message Type	Description
broadcast_stmt	Indicates a message that is being broadcast through the broadcast channel for the application programs
connection_port_search_stmt	Indicates that the designated process is looking for a port through which it can connect to the broadcast channel
connection_edge_search_call	Indicates that the requesting process is looking for an edge through which it can connect to the broadcast channel
connection edge search resp	Indicates whether the edge between this process and the sending neighbor has been accepted by the requesting party
diameter_estimate_stmt	Indicates an estimated diameter of the broadcast channel
diameter_reset_stmt	Indicates to reset the estimated diameter to indicated diameter
disconnect_stmt	Indicates that the sending neighbor is disconnecting from the broadcast channel
condition_check_stmt	Indicates that neighbors with empty port condition have been detected
condition_double_check_stmt	Indicates that the neighbors with empty ports have the same set of neighbors
shutdown_stmt	Indicates that the broadcast channel is being shutdown

Flow Diagrams

FIGS. 8-34 are flow diagrams illustrating the processing of the broadcaster component in one embodiment. FIG. 8 is

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

FIG. 9 is a flow diagram illustrating the processing of the seek portal computer routine in one embodiment. This routine is passed the channel type and channel instance of the broadcast channel to which this process wishes to connect. This routine, for each search depth (e.g., port number), checks the portal computers at that search depth. If a portal computer is located at that search depth with a process that is fully connected to the broadcast channel, then the routine returns an indication of success. In blocks 902-911, the routine loops selecting each search depth until a process is located. In block 902, the routine selects the next

US 6,701,344 B1

19

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

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

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

20

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

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

US 6,701,344 B1

21

necting to the broadcast channel. For example, a fellow seeking process may become the first process fully connected to the broadcast channel.

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

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

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

22

is not fully connected, then it is attempting to connect to the broadcast channel. In block 1505, the routine sends the external message response (i.e., `seeking_connection_resp`) to the seeking process and then returns.

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

FIG. 17 is a flow diagram illustrating the processing of the add neighbor routine in one embodiment. This routine adds the process calling on the external port as a neighbor to this process. In block 1701, the routine identifies the calling process on the external port. In block 1702, the routine sets a flag to indicate that the neighbor has not yet received the broadcast messages from this process. This flag is used to

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

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

recently sent connection edge search call has not yet been acknowledged and indicates that the edge to this neighbor is reserved if the remaining forwarding distance is less than or equal to one. It is reserved because the selected neighbor may offer this edge to the requesting process for edge pinning. The routine then returns.

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

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

FIG. 21 is a flow diagram illustrating the processing of the fill hole routine in one embodiment. This routine is passed an indication of the requesting process. If this process is requesting to fill a hole, then this routine sends an internal

US 6,701,344 B1

25

message to other processes. If another process is requesting to fill a hole, then this routine invokes the routine to handle a connection port search request. In block 2101, the routine initializes a connection port search statement internal message (i.e., `connection_port_search_stmt`). In decision block 2102, if this process is the requesting process, then the routine continues at block 2103, else the routine continues at block 2104. In block 2103, the routine distributes the message to the neighbors of this process through the internal ports and then returns. In block 2104, the routine invokes the handle connection port search routine and then returns.

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

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

26

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

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

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

FIG. 28 is a flow diagram illustrating the processing of the handle connection edge search call routine in one embodiment. This routine is passed a indication of the neighbor who sent the message and the message itself. This routine either forwards the message to a neighbor or proposes the edge between this process and the sending neighbor to the requesting process for edge pinning. In decision block 2801, if this process is not the requesting process or the number of holes of the requesting process is still greater than or equal to two, then the routine continues at block 2802, else the routine continues at block 2813. In decision block 2802, if the forwarding distance is greater than zero, then the random walk is not complete and the routine continues at block 2803, else the routine continues at block 2804. In block 2803, the routine invokes the forward connection edge search routine passing the identification of the requesting

process and the decremented forwarding distance. The routine then continues at block 2815. In decision block 2804, if the requesting process is a neighbor or the edge between this process and the sending neighbor is reserved because it has already been offered to a process, then the routine continues at block 2805, else the routine continues at block 2806. In block 2805, the routine invokes the forward connection edge search routine passing an indication of the requesting party and a toggle indicator that alternatively indicates to continue the random walk for one or two more computers. The routine then continues at block 2815. In block 2806, the routine dials the requesting process via the call-in port. In block 2807, the routine sends an edge proposal call external message (i.e., edge_proposal_call) and receives the response (i.e., edge_proposal_resp). Assuming that the response is successfully received, the routine continues at block 2808. In decision block 2808, if the response indicates that the edge is acceptable to the requesting process, then the routine continues at block 2809, else the routine continues at block 2812. In block 2809, the routine reserves the edge between this process and the sending neighbor. In block 2810, the routine adds the requesting process as a neighbor by invoking the add_neighbor routine. In block 2811, the routine removes the sending neighbor as a neighbor. In block 2812, the routine hangs up the external port and continues at block 2815. In decision block 2813, if this process is the requesting process and the number of holes of this process equals one, then the routine continues at block 2814, else the routine continues at block 2815. In block 2814, the routine invokes the fill hole routine. In block 2815, the routine sends an connection edge search response message (i.e., connection_edge_search_response) to the sending neighbor indicating acknowledgement and then returns. The graphs are sensitive to parity. That is, all possible paths starting from a node and ending at that node will have an even length unless the graph has a cycle whose length is odd. The broadcaster component uses a toggle indicator to vary the random walk distance between even and odd distances.

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

FIG. 30 is a flow diagram illustrating the processing of the broadcast routine in one embodiment. This routine is invoked by the application program to broadcast a message on the broadcast channel. This routine is passed the message to be broadcast. In decision block 3001, if this process has at least one neighbor, then the routine continues at block

3002, else the routine returns since it is the only process connected to be broadcast channel. In block 3002, the routine generates an internal message of the broadcast statement type (i.e., broadcast_stmt). In block 3003, the routine sets the sequence number of the message. In block 3004, the routine invokes the distribute internal message routine to broadcast the message on the broadcast channel. The routine returns.

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

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

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

FIG. 34 is a flow diagram illustrating the processing of the handle condition double check routine. This routine determines whether the neighbors with empty ports condition really is a problem or whether the broadcast channel is in the small regime. In decision block 3401, if this process has one hole, then the routine continues at block 3402, else the routine continues at block 3403. If this process does not have one hole, then the set of neighbors of this process is not the same as the set of neighbors of the sending process. In decision block 3402, if this process and the sending process have the same set of neighbors, then the broadcast channel is not in the small regime and the routine continues at block

3403, else the routine continues at block 3406. In decision block 3403, if this process has no holes, then the routine returns, else the routine continues at block 3404. In block 3404, the routine sets the estimated diameter for this process to one. In block 3405, the routine broadcasts a diameter reset internal message (i.e., diameter_reset) indicating that the estimated diameter is one and then returns. In block 3406, the routine creates a list of neighbors of this process. In block 3407, the routine sends the condition check message (i.e., condition_check_stmt) with the list of neighbors to the neighbor who sent the condition double check message and then returns.

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

What is claimed is:

- 1. A computer network for providing a game environment for a plurality of participants, each participant having connections to at least three neighbor participants, wherein an originating participant sends data to the other participants by sending the data through each of its connections to its neighbor participants and wherein each participant sends data that it receives from a neighbor participant to its other neighbor participants, further wherein the network is m-regular, where m is the exact number of neighbor participants of each participant and further wherein the number of participants is at least two greater than m thus resulting in a non-complete graph.
- 2. The computer network of claim 1 wherein each participant is connected to 4 other participants.
- 3. The computer network of claim 1 wherein each participant is connected to an even number of other participants.
- 4. The computer network of claim 1 wherein the network is m-connected, where m is the number of neighbor participants of each participant.
- 5. The computer network of claim 1 wherein the network is m-regular and m-connected, where m is the number of neighbor participants of each participant.
- 6. The computer network of claim 1 wherein all the participants are peers.
- 7. The computer network of claim 1 wherein the connections are peer-to-peer connections.
- 8. The computer network of claim 1 wherein the connections are TCP/IP connections.
- 9. The computer network of claim 1 wherein each participant is a process executing on a computer.
- 10. The computer network of claim 1 wherein a computer hosts more than one participant.

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

12. The computer network of claim 1 wherein the inter-connections of participants form a broadcast channel for a game of interest.

13. A distributed game system comprising:

a plurality of broadcast channels, each broadcast channel for playing a game, each of the broadcast channels for providing game information related to said game to a plurality of participants, each participant having connections to at least three neighbor participants, wherein an originating participant sends data to the other participants by sending the data through each of its connections to its neighbor participants and wherein each participant sends data that it receives from a neighbor participant to its neighbor participants, further wherein the network is m-regular, where m is the exact number of neighbor participants of each participant and further wherein the number of participants is at least two greater than m thus resulting in a non-complete graph; means for identifying a broadcast channel for a game of interest; and

means for connecting to the identified broadcast channel.

14. The distributed game system of claim 13 wherein means for identifying a game of interest includes accessing a web server that maps games to corresponding broadcast channel.

15. The distributed game system of claim 13 wherein a broadcast channel is formed by player computers that are each interconnected to at least three other computers.

16. A computer network for providing a game environment for a plurality of participants, each participant having connections to exactly four neighbor participants, wherein an originating participant sends data to the other participants by sending the data through each of its connections to its neighbor participants and wherein each participant sends data that it receives from a neighbor participant to its other neighbor participants, further wherein the network is in a stable 4-regular state and wherein there are at least six participants to result in a non-complete graph.

17. The computer network of claim 16 wherein a computer hosts more than one participant.

18. A computer network for providing a game environment for a plurality of participants, each participant having connections to at least three neighbor participants, wherein an originating participant sends data to the other participants by sending the data through each of its connections to its neighbor participants and wherein each participant sends data that it receives from a neighbor participant to its other neighbor participants, further wherein the network is m-regular and the network forms an incomplete graph.

19. The computer network of claim 18 wherein a computer hosts more than one participant.

* * * * *

(12) **INTER PARTES REVIEW CERTIFICATE** (1152nd)

United States Patent
Bourassa et al.

(10) **Number:** US **6,701,344 K1**
(45) **Certificate Issued:** Apr. 11, 2019

(54) **DISTRIBUTED GAME ENVIRONMENT**

(75) Inventors: **Virgil E. Bourassa; Fred B. Holt**

(73) Assignee: **ACCELERATION BAY, LLC**

Trial Numbers:

IPR2015-01970 filed Sep. 25, 2015

IPR2016-00933 filed Apr. 22, 2016

IPR2015-01972 filed Sep. 25, 2015

IPR2016-00934 filed Apr. 22, 2016

Inter Partes Review Certificate for:

Patent No.: **6,701,344**

Issued: **Mar. 2, 2004**

Appl. No.: **09/629,042**

Filed: **Jul. 31, 2000**

The results of IPR2015-01970 joined with IPR2016-00933; IPR2015-01972 joined with IPR2016-00934 are reflected in this inter partes review certificate under 35 U.S.C. 318(b).

INTER PARTES REVIEW CERTIFICATE

U.S. Patent 6,701,344 K1

Trial No. IPR2015-01970

Certificate Issued Apr. 11, 2019

1

AS A RESULT OF THE INTER PARTES
REVIEW PROCEEDING, IT HAS BEEN
DETERMINED THAT:

Claim 12 is found patentable.

Claims 1-11 and 16-19 are cancelled.

21. (substitute for claim 7) *A computer network for providing a game environment for a plurality of gaming participants, each gaming participant having connections to at least three neighbor gaming participants, wherein an originating gaming participant sends gaming data to the other gaming participants by sending the gaming data through each of its connections to its neighbor gaming participants and wherein each gaming participant sends gaming data that it receives from a neighbor gaming participant to its other neighbor gaming participants,*

2

*further wherein the network is m-regular, where m is the exact number of neighbor gaming participants of each gaming participant,
further wherein the number of gaming participants is at least two greater than m thus resulting in a non-complete graph,
further wherein the connections between the gaming participants are peer-to-peer connections,
further wherein the network is formed through a broadcast channel that overlays an underlying network,
further wherein the game environment is provided by at least one game application program executing on each computer of the computer network that interacts with the broadcast channel, and
further wherein gaming participants can join and leave the network using the broadcast channel.*

* * * * *

EXHIBIT 5

**THIS EXHIBIT HAS BEEN
REDACTED IN ITS ENTIRETY**

EXHIBIT 6

**THIS EXHIBIT HAS BEEN
REDACTED IN ITS ENTIRETY**

EXHIBIT 7

**THIS EXHIBIT HAS BEEN
REDACTED IN ITS ENTIRETY**

EXHIBIT 8

TERMS OF SERVICE

REVISED: OCTOBER 1, 2013

TERMS & CONDITIONS

This document constitutes an agreement (the "Agreement") between you and the United States company Rockstar Games, Inc., its parents, subsidiaries, and affiliates, (the "Company," "we," "us," and "our") that governs the relationship between you and the Company with respect to your use of the Online Services. The Company provides access to the Online Services and any related services subject to your compliance with this Agreement. Thus, it is important that you carefully read and understand this Agreement.

This Internet website is the property of Rockstar Games, a wholly owned subsidiary of Take Two Interactive (collectively, the "Company").

The terms and conditions herein are in addition to and supplement the End User License Agreement at www.rockstargames.com/eula that governs the use of all software and services distributed by the Company.

[Description Of Online Services](#)

[Trademark And Copyright Information](#)

[Submissions](#)

[Code Of Conduct](#)

[Limited License By The Company](#)

[License To The Company](#)

[Making Purchases](#)

[Ringtones, Wallpapers, And Other Mobile Device Services & Products](#)

[Virtual Currency And Virtual Goods](#)

[Warranty Disclaimer](#)

[Void Where Prohibited](#)

[Indemnification](#)

[Litigation Issues](#)

[Termination](#)

[Miscellaneous](#)

[Designated Agent Under the Digital Millenium Copyright Act](#)

Repeat Infringer Policy

DESCRIPTION OF ONLINE SERVICES

Subject to full compliance with this Agreement, the Company may offer to provide certain products, services, and websites accessed through internet-capable hardware platforms including gaming consoles, personal computers, mobile computers, or smart phones, or in-game applications or software platforms including third-party hosts (collectively the "Online Services"). Online Services shall include, but not be limited to, any service or content the Company provides to you, including any materials displayed or performed. The Company may change, suspend or discontinue the Online Services for any reason, at any time, including the availability of any feature or content. The Company may also impose limits on certain features and services or restrict your access to parts or all of the Online Services without notice or liability.

TRADEMARK AND COPYRIGHT INFORMATION

All Online Services material, including, but not limited to, text, data, graphics, logos, button icons, images, audio clips, video clips, links, digital downloads, data compilations, and software is owned, controlled by, licensed to, or used with permission by the Company and is protected by copyright, trademark, and other intellectual property rights. The Online Services material is made available solely for your personal, non-commercial use and may not be copied, reproduced, republished, modified, uploaded, posted, transmitted, or distributed in any way, including by email or other electronic means, without the express prior written consent of the Company in each instance. You may download material intentionally made available for downloading through the Online Services for your personal, non-commercial use only, provided that you keep intact any and all copyright and other proprietary notices that may appear on such materials.

SUBMISSIONS

PLEASE NOTE: The Company welcomes input from the gaming community. However, any submissions to the Company of any nature whatsoever, whether through a posting on a Company website, email to the Company, mail, or any other means, becomes the sole and exclusive property of the Company, which shall have full right, title and interest thereto, including all copyrights(including the right to create a derivative work and the right of the original author in the exploitation of a derivative work), in all mediums now existing or hereafter created, and without any obligation to account, credit, or make any payment to the submitter for any use thereof. No purported reservation of rights incorporated in or accompanying any submission shall have any force or effect. By making a submission of any kind to the Company, you hereby agree to all of the foregoing.

CODE OF CONDUCT

The following rules, policies, and disclaimers shall govern and/or apply to your use of the Online Services.

You agree, by using the Online Services, that: (1) you will only use the Online Services for lawful purposes, in compliance with applicable laws, for your own personal, non-commercial use; (2) you will not restrict or inhibit any other user from using and enjoying the Online Services (for example, by means of harassment, hacking or defacement); (3) you will not use the Online Services to create, upload, or post any material that is knowingly false and/or defamatory, inaccurate, abusive, vulgar, obscene, profane, hateful, harassing, sexually oriented, threatening, invasive of one's privacy, in violation of any law, or is inconsistent with community standards; (4) you will not post, upload, or create any copyrighted material using the Online Services unless you own the copyright in and to such material; (5) you will not post, upload, or transmit any information or software that contains a virus, worm, timebomb, cancelbot, trojan horse or other harmful, disruptive, or deleterious component; (6) you will not post, upload, create, or transmit materials in violation of another party's copyright or other intellectual property rights; (7) you will not cheat or utilize any unauthorized robot, spider, or other program in connection with the Online Services; and (8) you will not impersonate any other individual or entity in connection with your use of the Online Services. All determinations will be made by the Company in its sole discretion.

When we provide Online Services involving user-created content ("UGC"), we do not review every piece of UGC, nor do we confirm the accuracy, validity, or originality of the UGC posted. We do not actively monitor the contents of the postings, nor are we responsible for the content of any postings. We do not vouch for, nor do we warrant the validity, accuracy, completeness, or usefulness of any UGC. The contents of the postings do not represent the views of the Company, its subsidiaries, or any person or property associated with the Company, the Online Services, or any website in the Company's family of websites. If you feel that any posting is objectionable, we encourage you to use associated report functions or contact us by visiting www.rockstargames.com/support. We will remove objectionable content if we deem removal to be warranted. Please understand that removal or editing of any content is a manual process and might not occur immediately. We reserve the right to remove (or not) any UGC or content for any (or no) reason whatsoever. You remain solely responsible for your UGC, and you agree to indemnify and hold harmless the Company and its agents with respect to any claim based upon the transmission of your UGC. Posting of advertisements, chain letters, pyramid schemes, solicitations, and the like, are inappropriate and forbidden on the Online Services (including bulletin boards and chat rooms).

We reserve the right to reveal your identity (including whatever information we know about you) without notice to you in certain circumstances set forth in our Privacy Policy. Please visit www.rockstargames.com/privacy for more details.

LIMITED LICENSE BY THE COMPANY

The Company grants you a limited, non-sublicensable license to access and use the Online Services. Such license is subject to this Agreement and, as applicable, the software EULA located at www.rockstargames.com/eula, and specifically conditioned upon the following: (i) you may only view, copy and print portions of the Online Services for your own informational, personal and non-commercial use; (ii) you may not modify or otherwise make derivative uses of the Online Services, or any portion thereof; (iii) you may not remove or modify any copyright, trademark, or other proprietary notices that have been placed in the Online Services; (iv) you may not use any data mining, robots or similar data gathering or extraction

methods; (v) you may not use the Online Services other than for their intended purpose; (vi) you may not reproduce, prepare derivative works from, distribute or display the Online Services, except as provided herein; and (vii) you must not violate the Code of Conduct set forth above.

Except as expressly permitted above, any use of any portion of the Online Services without the prior written permission of the Company is strictly prohibited and will terminate the license granted herein. Any such unauthorized use may also violate applicable laws, including without limitation copyright and trademark laws and applicable communications regulations and statutes. Unless explicitly stated herein, nothing in this Agreement may be construed as conferring any license to intellectual property rights, whether by estoppel, implication or otherwise. This license is revocable at any time.

You represent and warrant that your use of the Online Services will be consistent with this license, the EULA, and any other applicable agreements or policies, and will not infringe or violate the rights of any other party or breach any contract or legal duty to any other parties, or violate any applicable law. You expressly agree to indemnify the Company against any liability to any person arising out of your use of Online Services not in accordance with this Agreement. To request permission for uses of the Online Services not included in the foregoing license, you may write to the Company at webmaster@take2games.com.

LICENSE TO THE COMPANY

By creating UGC, posting messages, uploading files, creating files, inputting data, or engaging in any form of communication with or through the Online Services, you are granting the Company a royalty-free, perpetual, non-exclusive, unrestricted, worldwide license to: (1) use, copy, sublicense, adapt, transmit, publicly perform, or display any such material; and (2) sublicense to third parties the unrestricted right to exercise any of the foregoing rights granted with respect to the material. The foregoing grants shall include the right to exploit any proprietary rights in such material, including but not limited to rights under copyright, trademark, service mark, or patent laws under any relevant jurisdiction. Please consult the EULA at www.rockstargames.com/eula for additional license terms related to our software.

MAKING PURCHASES

If you wish to purchase products or services described in the Online Services, you may be asked to supply certain information including credit card or other payment information. You agree that all information that you provide will be accurate, complete, and current. You agree to pay all charges, including shipping and handling charges, incurred by users of your credit card or other payment mechanism at the prices in effect when such charges are incurred. You will also be responsible for paying any applicable taxes relating to your purchases. Please review the the Company's privacy policy at www.rockstargames.com/privacy before submitting such information.

RINGTONES, WALLPAPERS, AND OTHER MOBILE DEVICE SERVICES & PRODUCTS

Certain mobile phone handsets and carriers offer services that enable consumers to select and purchase directly through their mobile devices various digital mobile products. The Online Services may also offer the ability to select and purchase various digital mobile products that will be delivered to your mobile device. These digital mobile products offerings and products may enable the consumer to customize their mobile device or mobile device service (for example with ringtones or wallpaper), or allow the consumer to select certain video or audio files that can be viewed or listened to whenever the consumer chooses. All or some of the digital mobile products offerings may not be available on, transmissible to, or compatible with all mobile devices. As a result, consumers may not be able to access, purchase or make use of all the services or offerings. Any attempt to purchase these products or services may result in mobile carrier charges being separately billed to your mobile device account for SMS messaging or other communications. In addition, the consumer may be separately billed by the mobile carrier for the actual product, service or offering selected. In the event the consumer has a call waiting and an incoming call is received while accessing or ordering any mobile product or service, such product, service or other offering may be interrupted or may not completely download. You can unsubscribe from any subscription service by following the instructions in the message or on the website related to the product. Please see <http://www.rockstargames.com/support> for support information regarding our other products.

VIRTUAL CURRENCY AND VIRTUAL GOODS

The Online Services, including software, may offer the ability to purchase and/or earn via gameplay a limited license to use virtual currency and/or virtual goods exclusively within applicable software and services provided by the Company. Such license is subject to and specifically conditioned upon your acceptance of, and compliance with, the EULA, this Agreement and any other applicable policies or agreements. All in-game Virtual Currency and/or Virtual Goods may be consumed or lost by players in the course of gameplay according to the game's rules applicable to currency and goods, which may vary. See the EULA at www.rockstargames.com/eula for more details.

WARRANTY DISCLAIMER

THE COMPANY MAY PROVIDE LINKS AND POINTERS TO INTERNET WEBSITES MAINTAINED BY THIRD PARTIES ("THIRD-PARTY SITES") AND MAY, FROM TIME TO TIME, PROVIDE THIRD-PARTY MATERIALS ON ITS WEBSITES. NEITHER THE COMPANY, ITS PARENT OR SUBSIDIARY COMPANIES, NOR THEIR AFFILIATES, ENDORSE, TAKE RESPONSIBILITY FOR, OPERATE OR CONTROL IN ANY RESPECT ANY INFORMATION, PRODUCTS, OR SERVICES ON THESE THIRD-PARTY SITES. THE MATERIALS ON COMPANY WEBSITES AND THE THIRD-PARTY SITES ARE PROVIDED "AS IS" AND "AS AVAILABLE" WITH NO REPRESENTATIONS OR WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT. YOU ASSUME TOTAL RESPONSIBILITY AND RISK FOR YOUR USE OF COMPANY WEBSITES AND THE ONLINE SERVICES.

VOID WHERE PROHIBITED

Although Company Online Services are accessible worldwide, not all products or services are available to all persons or in all geographic locations. The Company reserves the right to limit, in its sole discretion, the provision and quantity of any product or service to any person or geographic area it so desires. Any offer for any product or service made is void where prohibited.

INDEMNIFICATION

You agree to indemnify, defend, and hold harmless the Company and its affiliated companies, officers, directors, employees, agents, licensors, and suppliers from and against all losses, expenses, damages, and costs, including reasonable attorneys' fees, resulting from any violation by you of this Agreement. The Company reserves the right to assume the exclusive defense and control of any matter subject to indemnification by you.

LITIGATION ISSUES

This Agreement is entered into in the State of New York and shall be governed by, and construed in accordance with, the laws of the State of New York, exclusive of its choice of law rules. You and the Company agree to submit to the exclusive jurisdiction of the state and federal courts sitting in the Borough of Manhattan in the City of New York in the State of New York, and waive any jurisdictional, venue, or inconvenient forum objections to such courts. You and the Company further agree as follows: (i) any claim brought to enforce this Agreement must be commenced within two (2) years of the cause of action accruing; (ii) no recovery may be sought or received for damages other than out-of-pocket expenses, except that the prevailing party will be entitled to costs and attorneys' fees; and (iii) any claim must be brought individually and not consolidated as part of a group or class action complaint.

TERMINATION

The Company may terminate or suspend any and all Online Services and any registered account immediately, without prior notice or liability, for any reason, including if you breach any terms and conditions of this Agreement. Upon termination of your account, your right to use the Online Services will immediately cease. If you wish to terminate your account, you may simply discontinue using the Online Services. All provisions of this Agreement which by their nature should survive termination shall survive termination, including, without limitation, ownership provisions, warranty disclaimers, indemnity and limitations of liability.

MISCELLANEOUS

In the event that any of the provisions of this Agreement are held by a court or other tribunal of competent jurisdiction to be unenforceable, such provisions shall be limited or eliminated to the minimum extent necessary so that this Agreement shall otherwise remain in full force and effect. This Agreement, along with the Privacy Policy located at www.rockstargames.com/privacy and the EULA, constitutes the entire agreement between you and the Company pertaining to the subject matter hereof, and any and all written or oral agreements heretofore existing between you and the Company with respect to the subject matter of this Agreement are expressly canceled. The Company may modify the terms of this Agreement at any time in its sole discretion by posting a revised Agreement or, in the case of a material modification, by posting notice of such modification on the website page entitled "Legal Notices" or "Legal Information" (or similar title) before the modification takes effect.

DESIGNATED AGENT UNDER THE DIGITAL MILLENNIUM COPYRIGHT ACT

The Digital Millennium Copyright Act ("DMCA") provides a mechanism for notifying service providers of claims of unauthorized use of copyrighted materials. Under the DMCA, a claim must be sent to the service provider's designated agent. If you believe in good faith that the Company should be notified of a possible online copyright infringement involving any Online Service, please notify the Company's designated agent:

Service Provider: Take-Two Interactive Software, Inc.

Address of Designated Agent:

Take-Two Interactive Software, Inc.

622 Broadway

New York, New York 10012

Attention: General Counsel

Telephone Number of Designated Agent: 646-536-2842

Facsimile Number of Designated Agent: 646-941-3566

Email Address of Designated Agent: copyright@take2games.com

Please be aware that, in order to be effective, your notice of claim must comply with the detailed requirements set forth in the DMCA. You are encouraged to review them (see 17 U.S.C. Sec. 512(c)(3)) before sending your notice of claim.

To meet the notice requirements under the DMCA, the notification must be a written communication that includes the following: To meet the notice requirements under the DMCA, the notification must be a written communication that includes the following: (1) A physical or electronic signature of a person authorized to act on behalf of the owner of an exclusive right that is allegedly infringed; (2) Identification of the copyrighted work claimed to have been infringed, or, if multiple copyrighted works at a single online site are covered by a single notification, a representative list of such works at that site; (3) Identification of the

material that is claimed to be infringing or to be the subject of infringing activity and that is to be removed or access to which is to be disabled, and information reasonably sufficient to permit us to locate the material; (4) Information reasonably sufficient to permit us to contact the complaining party, such as an address, telephone number and, if available, an electronic mail address at which the complaining party may be contacted; (5) A statement that the complaining party has a good-faith belief that use of the material in the manner complained of is not authorized by the copyright owner, its agent or the law; and (6) A statement that the information in the notification is accurate, and under penalty of perjury, that the complaining party is authorized to act on behalf of the owner of an exclusive right that is allegedly infringed.

REPEAT INFRINGER POLICY

In accordance with the DMCA and other applicable law, the Company has adopted a policy of terminating, in appropriate circumstances and at Company's sole discretion, registered accounts deemed to be repeat infringers. The Company may also at its sole discretion limit access to the Online Services and/or terminate the account of anyone who infringes any intellectual property rights of others, whether or not there is any repeat infringement.

[Back to top](#)

EXHIBIT 9



Home News Characters **Video** Images Guides Cheats

TERMS OF USE

You agree to not use **GTA-5** in order to:

- upload, post, or otherwise transmit any Content that is unlawful, harmful, threatening, abusive, harassing, tortious, defamatory, vulgar, obscene, libelous, invasive of another's privacy, hateful, or racially, ethnically or otherwise objectionable;
- harm minors in any way;
- impersonate any person or entity, including, but not limited to, a GTA-5 official, forum leader, guide or host, or falsely state or otherwise misrepresent your affiliation with a person or entity;
- forge headers or otherwise manipulate identifiers in order to disguise the origin of any Content transmitted through the Service or develop restricted or password-only access pages, or hidden pages or images (those not linked to from another accessible page);
- upload, post, or otherwise transmit any Content that you do not have a right to transmit under any law or under contractual or fiduciary relationships (such as inside information, proprietary and confidential information learned or disclosed as part of employment relationships or under nondisclosure agreements);
- upload, post, or otherwise transmit any Content that infringes any patent, trademark, trade secret, copyright, or other proprietary rights of any party;
- upload, post, or otherwise transmit any unsolicited or unauthorized advertising, promotional materials, "junk mail," "spam," "chain letters," "pyramid schemes," or any other form of solicitation, except in those areas of the Service that are designated for such purpose;
- upload, post, or otherwise transmit any material that contains software viruses or any other computer code, files or programs designed to interrupt, destroy or limit the functionality of any computer software or hardware or telecommunications equipment;
- interfere with or disrupt the servers or networks connected to the GTA-5 site, or disobey any requirements, procedures, policies or regulations of networks connected to the site;
- intentionally or unintentionally violate any applicable local, state, national or international law, including, but not limited to, regulations promulgated by the U.S. Securities and Exchange Commission, any rules of any national or other securities exchange, including, without limitation, the New York Stock Exchange, the American Stock Exchange or the NASDAQ, and any regulations having the force of law;
- "stalk" or otherwise harass another;
- collect or store personal data about other users;
- promote or provide instructional information about illegal activities, promote physical harm or injury against any group or individual, or promote any act of cruelty to animals;
- use your home page (or directory) as storage for remote loading or as a door or signpost to another home page; or
- engage in commercial activities without the express written consent of GTA-5. This includes, but is not limited to, the following activities:
 - offering for sale any products or services;
 - soliciting for advertisers or sponsors;
 - conducting raffles or contests that require any type of entry fee;
 - displaying a sponsorship banner of any kind, including those that are generated by banner or link exchange services

Back to Top ↑

Search...

Go →

Follow Us On Social Media



Advertisement

Categories

- Characters
- Images
- Cheats
- News
- Guides
- Videos

Advertisement

- displaying banners for services that provide cash or cash-equivalent prizes to users in exchange for hyperlinks to their websites.

Rights

You acknowledge that we acquire all Rights to use any posted materials as described above so that we do not violate any Rights you may have in materials you post. By submitting content to or through our sites, you grant us the non-exclusive right to reproduce, modify, and distribute it as we see fit in any medium and for any purpose in any form, media, or technology now known or later developed. You also permit any other user to access, display, and print such content for personal use. You vouch that any material you submit does not violate, plagiarize, or infringe upon the right of any third party, including copyright, trademark, or proprietary rights. If non-original content is included in your posting, you must obtain permission from the content owner and attribute it.

Copyrights and Copyright Agent

GTA-5 respects the rights of all copyright holders. If you believe that your work has been copied and used on one of our Sites in a way that constitutes copyright infringement, please provide GTA-5's Copyright Agent the following information required by Section 512 of the Digital Millennium Copyright Act:

Advertisement

- A physical or electronic signature of a person authorized to act on behalf of the owner of an exclusive right that is allegedly infringed;
- Identification of the copyright work claimed to have been infringed, or, if multiple copyrighted works at a single on-line site are covered by a single notification, a representative list of such works at that site;
- Identification of the material that is claimed to be infringing or to be the subject of infringing activity and that is to be removed or access to which is to be disabled, and information reasonably sufficient to permit us to locate the material;
- Information reasonably sufficient to permit GTA-5 to contact the complaining party;
- A statement that the complaining party has a good-faith belief that use of the material in the manner complained of is not authorized by the copyright owner, its agent, or the law; and
- A statement that the information in the notification is accurate, and under penalty of perjury, that the complaining party is authorized to act on behalf of the owner of an exclusive right that is allegedly infringed.

We appreciate the opportunity to serve you!

If you have any questions regarding this privacy policy statement or if you feel that this site has not followed its stated information policy, feel free to contact us.

Additionally, you may contact your state or local consumer protection office, The Direct Marketing Association's Committee on Ethical Business Practices, the **Better Business Bureau**, or The Federal Trade Commission by phone at (202) FTC-HELP (202/382-4357) and online at www.ftc.gov.

Note: This privacy policy applies to all of GTA-5's owned and operated websites and to GTA-5's other information gathering activities.

GTA-5 complies with all the stringent standards of ethical conduct set forth by the Direct Marketing Association.

– See more at: <http://www.gta-5.com/terms-and-condition/>

Leave a Reply

You must be logged in to post a comment.

2017 PulseFire Media

[Contact](#) | [Privacy Policy](#) | [Terms and Conditions](#)

[Back to Top](#) ↑

EXHIBIT 10





TABLE OF CONTENTS

- 2 PRODUCT SUPPORT
- 3 GAME CONTROLS
- 4 CONTROLS
- 4 BASIC OFFENSE
- 4 BASIC DEFENSE
- 5 ADVANCED OFFENSE
- 6 ADVANCED DEFENSE
- 7 PRO STICK™: SHOOTING
- 8 PRO STICK™: DRIBBLING
- 9 POST MOVES
- 10 DEFENSIVE CONTROLS
- 11 KINECT VOICE COMMANDS
- 13 NBA 2K16 GAME CREDITS
- 22 LIMITED SOFTWARE WARRANTY, LICENSE AGREEMENT & INFORMATION USE DISCLOSURES

WARNING Before playing this game, read the Xbox One™ system, and accessory manuals for important safety and health information. www.xbox.com/support.

Important Health Warning: Photosensitive Seizures

A very small percentage of people may experience a seizure when exposed to certain visual images, including flashing lights or patterns that may appear in video games. Even people with no history of seizures or epilepsy may have an undiagnosed condition that can cause "photosensitive epileptic seizures" while watching video games. Symptoms can include light-headedness, altered vision, eye or face twitching, jerking or shaking of arms or legs, disorientation, confusion, momentary loss of awareness, and loss of consciousness or convulsions that can lead to injury from falling down or striking nearby objects. **Immediately stop playing and consult a doctor if you experience any of these symptoms.** Parents, watch for or ask children about these symptoms—children and teenagers are more likely to experience these seizures. The risk may be reduced by being farther from the screen; using a smaller screen; playing in a well-lit room; and not playing when drowsy or fatigued. If you or any relatives have a history of seizures or epilepsy, consult a doctor before playing.



Product Support:
<http://support.2k.com>

Please note that NBA 2K16 online features are scheduled to be available until **December 31, 2017** though we reserve the right to modify or discontinue online features on 30-days' notice. Visit www.nba2k.com/status for more information.

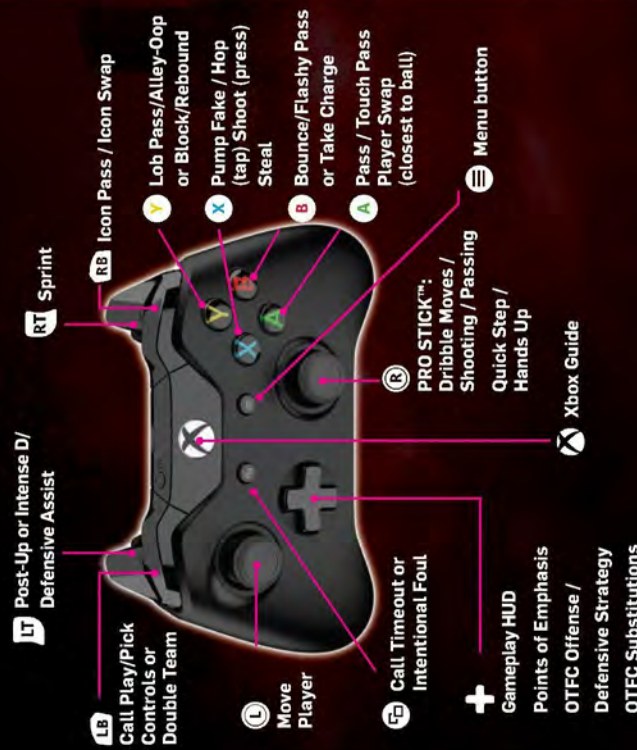
Some features will be temporarily unavailable during initial game installation.

XBOX ONE CONTROLLER

Basic Offense	Control	Basic Defense
Move Player	O	Move Player
PRO STICK™: Dribble Moves / Shooting / Passing	O	Quick Step / Hands Up
Post-Up	Y	Intense D / Defensive Assist
Sprint	R1	Sprint
Call Play / Pick Controls	A	Double Team
Icon Pass	B	Icon Swap
Pass / Touch Pass	A	Player Swap (closest to ball)
Bounce Pass (tap), Flashy Pass (double tap)	B	Take Charge
Shoot (press) Pump Fake / Hop (tap) Spin Gather (double tap)	X	Steal (press) Intentional Foul (hold)
Lob Pass (tap), Alley-Oop (double tap)	Y	Block / Rebound
2K Smart Play	+	Gameplay HUD
Points of Emphasis	↔	Points of Emphasis
OTFC Quick Plays / Offense Strategy	↔	OTFC Defensive Sets
OTFC Substitutions	↔	OTFC Substitutions

Game Controls

Xbox One Wireless Controller



ADVANCED OFFENSE

Action	Input
Positional Playcall	Tap A , tap desired teammate's player icon, choose play from menu
2K Smart Play	C
Pick Control	Press and hold A . Use D to choose Roll vs. Fade and A to choose pick side
Bounce Pass	Tap B
Overhead/Lob Pass	Tap V
Flashy Pass	Double-tap B
Alley-Oop	Double-tap V
Dribble Pitch/Handoff	Press and hold B to bring the selected teammate to the ball, wait for him to get into handoff range or release B to force the pass early
Lead to Basket Pass	Press and hold V to force the selected teammate to make a basket cut, wait for him to get in range or release V to force the pass early
Fake Pass	V + B (while standing or driving)
Give & Go	Press and hold A to retain control of passer, release A to pass the ball back to him
Putback Dunk/Layup Finish Alley-Oop (when controlling receiver)	Hold X
PRO STICK™ Pass	A + C
Call Timeout	View button

ADVANCED DEFENSE

Action	Input
Move	C
Quick Step Movement	Tap C in the direction you want the defender to step
Fast Shuffle	A + V + C
Steal	Tap X
Block	V
Rebound	V (ball in air)
Take Charge	B
Flop	Double-tap B
Crowd Dribbler	Hold V
Dynamic Defensive Assist	Press and hold V (press harder for more assistance)
Hands Up	Hold C
Deny Hands Out	Hold C (while playing offball defense)
Double Team	Hold A
Icon Double Team	Tap A , then press and hold desired double teamer's action button

PRO STICK™

The PRO STICK™ gives you more control over your offensive arsenal than ever before.

PRO STICK™ : SHOOTING

Action	Input
Jump Shot	Hold ○ in any direction (toward hoop for bank shot)
Pump Fake	Start a jump shot, then quickly release ○
Runner / Floater (driving mid-range)	Hold ○ away from hoop
Stepback Jumper (driving lateral)	Hold ○ away from hoop
Hop Gather	Tap ⓧ while standing or driving (○ determines direction of hop)
Spin Gather	Double tap ⓧ while standing or driving
Normal Layup (driving to hoop)	Hold ○ left, right, or toward hoop while driving (○ direction determines finish hand)
Euro Step Layup (driving to hoop)	Hold ○ away left/right
Reverse Layup (driving along baseline)	Hold ○ toward baseline
2-Hand Dunks (driving to hoop)	ⓧ + Hold ○ toward hoop
Dominant/Off-Hand Dunk (driving to hoop)	ⓧ + Hold ○ left or right to dunk with that hand
Flashy Dunks (driving to hoop)	ⓧ + Hold ○ away from hoop
Mid-Air Change Shot	Start dunk/layup, ○ any direction while in air
Step Through	Pump fake, then hold ○ again before pump take ends

PRO STICK™ : DRIBBLING

Action	Input	Context
Triple Threat Jab Step	Tap ○ Left/Right/Forward	Triple Threat
Triple Threat Spinout	Rotate ○ then quickly return to neutral	Triple Threat
Triple Threat Stepback	ⓧ + Tap ○ away from hoop	Triple Threat
Signature Sizeup	ⓧ + Tap ○ in any direction	Dribbling
Hesitation (quick)	Tap ○ toward ball hand	Dribbling
In and Out	Tap ○ toward hoop	Dribbling
Crossover (front)	Tap ○ toward off hand	Dribbling
Crossover (between legs)	Tap ○ between off hand and player's back	Dribbling
Behind the Back	Tap ○ away from hoop	Dribbling
Spin	Rotate ○ from ball hand around player's back, then quickly return to neutral	Dribbling
Half-Spin	Rotate ○ in a quarter-circle from ball hand to hoop, then quickly return to neutral	Dribbling
Stepback	ⓧ + Tap ○ away from hoop	Dribbling

POST MOVES (PRESS **Y** TO POST UP)

Action	Input
Post Movement	Hold O
Drive to Key	Y + O toward key then quickly release Y
Drive to Baseline	Y + O toward baseline then quickly release Y
Quick Spin	Rotate O to outside shoulder
Hook Drive	Rotate O to inside shoulder
Fakes	Tap O in any direction but away from hoop
Post Hop	Hold O to the left or right away from hoop, then tap X
Post Stepback	Hold O away from hoop, then tap X
Dropstep	Hold O to the left or right toward hoop, then tap X

POST SHOTS

Action	Input
Post Hook (close range)	O toward hoop (with O neutral)
Post Fade (beyond close range)	O left or right away from hoop
Step Through Layout	O toward hoop (while holding O toward hoop)
Shimmy Fade	Hold Y then move O left or right away from hoop
Pump Fake	Start a shot listed above then move O to neutral
Up & Under / Step Through	Pump fake, then O again before pump fake ends

DEFENSIVE CONTROLS

Action	Input	Context
Move	O	Any
Fast Shuffle	Y + Y + O	Any
Steal	Tap X	Any
Block	Y	Any
Rebound	Y (ball in air)	Any
Take Charge	O	Any
Flop	Double-tap O	Onball Defense
Intense Defense	Y	Onball Defense
Crowd Dribbler	Hold Y	Onball Defense
Hands Up	Hold O	Onball Defense
Deny Ball	Hold Y	Offball Defense
Double Team	O	Any

KINECT VOICE COMMANDS

You can use Kinect Voice Commands to implement a variety of actions while playing.

Voice Command	Action
Always Active	
"Time Out" "Call Time Out"	Call a time out
"Switch Camera"	Switches to next camera position
Offense	
"Isolation" "Post Play" "Pick and Roll" "Three Point"	Playtypes
"Quick Isolation" "Quick Iso" "Clear Out" "Pick and Roll" "Quick Post Up" "Quick Spot Up Three"	Quick play control
"Set a Screen for me" "Set a Pick for me"	Quick screen

Defense	
"Double Team"	Call for AI double team
"Help Me"	Call for help from team
"Intentional Foul"	Call for intentional foul
"Pick up ball"	Call for nearest AI player to switch to the ball handler if he doesn't have him already
"Bring in -- Bench Player Last Name / Full Name"	Initiate a substitution with a specific player
"Man to Man" "Zone 2-3" "Zone 3-2" "Fullcourt Press"	Call for defensive set
MyCAREER	
"Pass me the ball" "Pass the ball to me"	Call for the ball
"Alley oop" "Throw the alley"	Call for an alley oop pass
"Shoot the Ball!" "Shoot that Shot" "Take that Shot" "Shoot that!" "Shoot it"	Call for AI shot

NBA 2K16 GAME CREDITS

VISUAL CONCEPTS ENTERTAINMENT, INC.

LEAD ENGINEER
Andrew Morrison

ART DIRECTOR
Joseph Clark

ENGINEERING

AI ENGINEERS
Shawn Lee
Gordon Reed
Eddie Park
Ben Hester

TECH GROUP
Kamrhik Krishnamurthy
Johanna Tang
Mark Harsley
Chris Larson
Nick Jones
Mark Roberts
Evan Hargraves
Tim Schroeder
Steven Fuller
David Copalowski
Brian Tommaso
Harden Young
Paul Hale
Brod Jones
Kenny LeVergne
Court Keeler
Anthony Lundquist
Jeff Boudara
Nathan DeGrand
Scott Kohn
Srikanth Jagannathan
Katherine Hayton
Wen Chi Gu
David Yu
Eletherios "Lefkos" Anastoglou
Shua "Belle" Liu
Jingyi "Jingyi" Liu
Yu Bai
Bo Liang
Arvind Gopalakrishnan

DIRECTOR OF TECHNOLOGY
Tim Walter

LEAD LIBRARY ENGINEER
Iver Olsen

LEAD TOOLS

LIBRARY ENGINEER
Boris Kazanski
Zhe Peng
Brian Ramagli

SENIOR TOOLS
Scholar Boesler
Romanik Rousseau

PRODUCTION

EXECUTIVE PRODUCER
Jeff Thomas

SENIOR PRODUCERS
Asif Chaudhry
Erick Boenisch
Falcia Steinhilber
Ben Bishop
Rob Jones

GAMEPLAY DIRECTOR
Mike Wang

PRODUCTION & DESIGN
Jared Heston
Jason Sapida
Dion Peete
Jay Iwahashi

CHARACTER ART DIRECTOR
Heather Marshall

CHARACTER ARTIST
Winnie Hsieh
Tyler Bronis
Tim Auer
Yuki Yamamura
Yoon Kim
Gmar Sanchisbal
Jeongcheol Shin
Evan Abilham
David Dame
Pascal Hang

TECHNICAL ART
Lore Gaudin
Lance Cook
Ream Stewart
Graf

ENVIRONMENT LEAD
John Lee

ENVIRONMENT ARTIST
Tim Doonan
Tim Loucks
Ray Wong

ANIMATION DIRECTOR
Roy Tse

Jason Souza
Dan Indra
Joe Levesque
Abe Navarro
Jon Cort
John Lee
Eric Dillard
Nino Samuel
Dan Bickley
Jesse Bean
Matt Lippincott
Robert Nelson
Kyle Lai-Fatt
Kurtis Hon
Matt Lippincott
Matt Lippincott
Mike O'Sullivan
Scott O'Sullivan
Charles Williams
Josh Morrison
Ben Horne
Lance Cook
Shawif Fattah
Brett Hawkins

ART TEAM

ANIMATION
Santiago Nunez
Technicolor

ADDITIONAL FACIAL PROCESSING
Technicolor

UI ART DIRECTOR
Heriman Fok

UI ART LEAD
Justin Cook
Ian Colino

UI VISUAL DESIGN
Anthony Yau
Zhen Tan

USER INTERFACE
Carrie Michelle Dinitz Parneki
David Lee
Andy Medler
Cathy Chargin
Jeffrey Davis
Spencer Kopach
Rob Simmons

STUDIO ART DIRECTOR
Anton Dawson

ART PRODUCER
Karen Huang

FACE CAPTURE
Pixelgun Studio

ANIMATION PRODUCER
Stephanie Gene Morgan

LEAD GAMEPLAY ANIMATOR
Elias Figueroa

GAMEPLAY TECHNICAL LEAD
Jamie Wicks

PERFORMANCE CREATIVE LEAD
Mike Dricko

PERFORMANCE TECHNICAL LEAD
Derek Kurimato

ANIMATOR
Ben Harrison
Joel Flory
Jonathan Lyons
Eric Perrier
Wister Phung

ADDITIONAL ANIMATION
Alvin Gano

ADDITIONAL FACIAL PROCESSING
Technicolor

UI ART DIRECTOR
Heriman Fok

UI ART LEAD
Justin Cook
Ian Colino

UI VISUAL DESIGN
Anthony Yau
Zhen Tan

USER INTERFACE
Carrie Michelle Dinitz Parneki
David Lee
Andy Medler
Cathy Chargin
Jeffrey Davis
Spencer Kopach
Rob Simmons

STUDIO ART DIRECTOR
Anton Dawson

ART PRODUCER
Karen Huang

FACE CAPTURE
Pixelgun Studio

SPECIAL THANKS
Matt Chaiwall
Heath Digital
Edge Art
Lemon Sky
Studio of Rock
Virtuos
Hydro74

STUDIO HOST
Ernie Johnson

STUDIO ANALYST
Shazelle O'Neal
Kenny Smith

PA ANNOUNCER
Peter Baro

PROMO ANNOUNCER
Jay Styne

OUTDOOR ANNOUNCER
CJ Norcia

SPANISH ANNOUNCERS
Siro Miguel Serrano

ANTONI DAMIEL
Jorge Quiroga

2KTV CAST

HOST & PRODUCER
Rachel A. DeMila

LEAD CAMERA & EDITOR
Alan Palmer

GAME EXPERT & PRODUCER
Jonathan Smith

EDITOR & CAMERA
Rodney Johnson
David Park

CAMERA & EDITOR
Bryan Pasco

LEAD GRAPHICS
Jolan Wood

AUDIO MIX
Brian Buel

ADDITIONAL CAMERA
Ian Levesque

MAKE UP ARTIST
Jeanne San Diego

ALEX BARIATE
Marissa Vossen

ANIMATION PRODUCER
Stephanie Gene Morgan

LEAD GAMEPLAY ANIMATOR
Elias Figueroa

GAMEPLAY TECHNICAL LEAD
Jamie Wicks

PERFORMANCE CREATIVE LEAD
Mike Dricko

PERFORMANCE TECHNICAL LEAD
Derek Kurimato

ANIMATOR
Ben Harrison
Joel Flory
Jonathan Lyons
Eric Perrier
Wister Phung

ADDITIONAL ANIMATION
Alvin Gano

ADDITIONAL FACIAL PROCESSING
Technicolor

UI ART DIRECTOR
Heriman Fok

UI ART LEAD
Justin Cook
Ian Colino

UI VISUAL DESIGN
Anthony Yau
Zhen Tan

USER INTERFACE
Carrie Michelle Dinitz Parneki
David Lee
Andy Medler
Cathy Chargin
Jeffrey Davis
Spencer Kopach
Rob Simmons

STUDIO ART DIRECTOR
Anton Dawson

ART PRODUCER
Karen Huang

FACE CAPTURE
Pixelgun Studio

ANIMATION PRODUCER
Stephanie Gene Morgan

LEAD GAMEPLAY ANIMATOR
Elias Figueroa

GAMEPLAY TECHNICAL LEAD
Jamie Wicks

PERFORMANCE CREATIVE LEAD
Mike Dricko

PERFORMANCE TECHNICAL LEAD
Derek Kurimato

ANIMATOR
Ben Harrison
Joel Flory
Jonathan Lyons
Eric Perrier
Wister Phung

ADDITIONAL ANIMATION
Alvin Gano

ADDITIONAL FACIAL PROCESSING
Technicolor

UI ART DIRECTOR
Heriman Fok

UI ART LEAD
Justin Cook
Ian Colino

UI VISUAL DESIGN
Anthony Yau
Zhen Tan

USER INTERFACE
Carrie Michelle Dinitz Parneki
David Lee
Andy Medler
Cathy Chargin
Jeffrey Davis
Spencer Kopach
Rob Simmons

STUDIO ART DIRECTOR
Anton Dawson

ART PRODUCER
Karen Huang

FACE CAPTURE
Pixelgun Studio

SPECIAL THANKS
Matt Chaiwall
Heath Digital
Edge Art
Lemon Sky
Studio of Rock
Virtuos
Hydro74

STUDIO HOST
Ernie Johnson

STUDIO ANALYST
Shazelle O'Neal
Kenny Smith

PA ANNOUNCER
Peter Baro

PROMO ANNOUNCER
Jay Styne

OUTDOOR ANNOUNCER
CJ Norcia

SPANISH ANNOUNCERS
Siro Miguel Serrano

ANTONI DAMIEL
Jorge Quiroga

2KTV CAST

HOST & PRODUCER
Rachel A. DeMila

LEAD CAMERA & EDITOR
Alan Palmer

GAME EXPERT & PRODUCER
Jonathan Smith

EDITOR & CAMERA
Rodney Johnson
David Park

CAMERA & EDITOR
Bryan Pasco

LEAD GRAPHICS
Jolan Wood

AUDIO MIX
Brian Buel

ADDITIONAL CAMERA
Ian Levesque

MAKE UP ARTIST
Jeanne San Diego

ALEX BARIATE
Marissa Vossen

LIVIN' DA DREAM CAST

FREO
Saranus J. Jakeson

CEE-CEE
Michelle Michener

MS. MARTHA
Gina Breedlove

MR. PETE
Arthur Richardson

VIC VAN LIER
Wade Wilson

YVETTE MENDENHALL
Anya Engel-Adams

TEAM OWNER
Paul Ghiringhelli

DOM PANGOTTI
Al Palagonia

OFFICER VASQUEZ
Benicki Hing Hernandez

RECRUITER
Doug Boyd
Chris Marsal
Walter McLean
Blair Leatherswood
Bobby August Jr.
Salvatore Calanni Jr.
George Philip Parrais
Vince Watson
Teddy Spence

GAFFER
Sereene Lee
Geoffrey Nola

LIVIN' DA DREAM PRODUCTION

DIRECTOR
Spike Lee

DIRECTOR OF PHOTOGRAPHY
Kevin DeVonish
Cliff Charles

1ST ASSISTANT CAMERA
Pam Laa
Nick Schwyer
Casting
Kim Coleman

LINE PRODUCER
Jason Sokoloff

PRODUCTION COORDINATOR
Ives Holier

MOTION CAPTURE DEPARTMENT

SCRIPT SUPERVISOR
Virginia McCarthy

CAMERA PRODUCTION ASSISTANT
Jess Dela Mercad

SUPERVISING SOUND EDITOR
Phil Stockton

SOUND ENGINEER
Patti Psur

POST PRODUCTION

SOUND COORDINATOR
Chris Fielder

MUSIC EDITOR
Martin Morris

SCORE COMPOSER
Bruce Hornsby

COSTUME DESIGN
Ruth Carter

DOLLY GRIP
Carlos Lopez
Rhamat Norwood

GRIP
Jay Coakley
Rick Edmondson
Walter McLean
Blair Leatherswood
Bobby August Jr.
Salvatore Calanni Jr.
George Philip Parrais
Vince Watson
Teddy Spence

GAFFER
Sereene Lee
Geoffrey Nola

SCRIPT SUPERVISOR

PRODUCTION ASSISTANT
Colin Donly

STAGE MANAGER
Anthony Tomalia

STAGE TECHNICIAN II
Joe Antonio
Emma Castles
Jeremy Sobichal

STAGE TECHNICIAN I
Stacy Adams
Christophara Burton

PRODUCTION MANAGER
Charles Ghislandi

SPECIALIST II
Jose Gonzalez
Gil Espanto
Ryan Girard

SPECIALIST I
Michelle Hill
Jeremy Wagues

TECHNICAL MANAGER
Steve Park

PIPELINE ENGINEER II
Charles Harris

MEDIA SUPERVISOR
Nate Baker

AUDIO ASSISTANT I
Andrew Hanson

CAMERA OPERATORS
Alan Ricardoz
Michael Ngontoya
Stephanie Sanchez

2K SPORTS THEME MUSIC

"THE CONTEST" AND "NETWORK SPUBS TONIGHT"
Written, Engineered, and Produced by Bill Kole

"THE COMEBACK", "THE RIVALRY", "THE COME DOW"
Written by Joel Simmons, Engineered and Produced by Bill Kole

2K THEMES PERFORMED BY
Cosmosquad

ARENA ORGAN, BEATS, MUSIC, & ADDITIONAL IN-GAME MUSIC

Casey Cameron
Hiras Ohira
Danielle Strickland
Joshua Gervanese
Reinard Coloma
Christopher Nichols
Belinda Erickman
Daniak Stafford
Leslie Peacock

MY PARK LOADING MUSIC & STUDIO MUSIC
Cody Mills

NATIONAL ANTHEM VOCALIST
Natalie Major

ADDITIONAL CAPTURE SUPPORT
Christopher Jones

SPECIAL THANKS
Tim Anderson
Phil Stockton
Fresno State Bulldog Marching Band
Greg Ortiz
Craig Reimer
California Aggie Marching Band

PLAYER CHATTER
Drew Johnson Jr.
Shane Meaton
Matt Pymn
Nick Powers
Michey L'Esquad
Will Dapino
Michael Turner
Spencer Douglas
Cody Burdman
Sean Pucher
Brian Shute
Eric White

2K CREATIVE DEVELOPMENT

VP CREATIVE DEVELOPMENT
Josh Atkins

CREATIVE DIRECTOR
Eric Simonich

DIRECTOR OF CREATIVE PRODUCTION
Jack Seibel

MANAGER OF CREATIVE PRODUCTION
Josh Ortelana

CREATIVE PRODUCTION COORDINATOR
Kaitlin Bleiler

CREATIVE PRODUCTION ASSISTANTS
William Gale
Marilyn Kelly
Megan Rohr

DIRECTOR OF RESEARCH AND PLANNING
Mike Salmon

SR. MARKET RESEARCHER
David Rees

USABILITY RESEARCHER
Jordan Linder

USER TESTING ASSISTANT
Jonathan Bonillas

2K MARKETING TEAM

SVP. MARKETING
Sarah Anderson

VP OF INTERNATIONAL MARKETING
Matthew Weiner

VP OF MARKETING
Alfie Brody

DIRECTOR OF MARKETING
Mike Rheinhardt

BRAND MANAGERS
Andrew Blumberg
William Inglis

VP OF COMMUNICATIONS, THE AMERICAS
Ryan Jones

SR. COMMUNICATIONS MANAGER
Ryan Peters

SR. DIRECTOR, MARKETING PRODUCTION
Jackie Truong

MANAGER
Haim Nguyen

MARKETING PRODUCTION ASSISTANT
Nelson Chao

SR. GRAPHIC DESIGNER
Christopher Mats

PROJECT MANAGER
Heidi Ois

VIDEO PRODUCTION MANAGER
Kenny Grubbe

VIDEO EDITOR / MOTION GRAPHIC DESIGNERS
Michael Rogteem

GRAPHIC DESIGNERS
Eric Neff

VIDEO EDITOR
Peter Koppman

ASSOCIATE VIDEO EDITORS
Doug Tyler
Nick Pyvalainen

ART DIRECTOR, WEB
Gabre Abarcar

WEB DIRECTOR
Nate Schlaumberg

INTERNATIONAL SOCIAL MEDIA & CONTENT EXECUTIVE
 Ibrahim Bhatti

2K INTERNATIONAL PRODUCT DEVELOPMENT
INTERNATIONAL PRODUCER
 Mark Ward

LOCALIZATION MANAGER
 Nathalie Adnawes

LOCALIZATION MANAGER ASSISTANT
 Emma Lepout

EXTERNAL LOCALIZATION TEAMS
 Leticia Rodriguez
 Synthesia International S.r.l.
 Synthesia Iberia
 Robert Book
 Local Heroes
 Keywada's International

DESIGN TEAM
 Tom Baker
 James Guilman

2K INTERNATIONAL TEAM
 Adam Merritt
 Agnes Rosique
 Alan Moore
 Alan Parker
 Balinda Crowe
 Ben Seccombe
 Bernardo Hermoso
 Carlo Volz
 Chris Jernigan
 Chris Jernigan
 Chris White
 Dan Cooke
 Daniel Hill
 David Galloway
 Dean Stanton
 Diana Freitag
 Jan Sturm
 Jean-Paul Hardy
 John Jernigan
 Lisa Mandel
 Maria Martinez
 Oliver Keller
 Sandra Melero
 Simon Turner
 Tim Smith
 Warner Guinée

TAKE TWO INTERNATIONAL OPERATIONS
 Anthony Dodd
 Nisha Verma
 Phil Anderson
 Richard Kelly

SR. DIRECTOR AND COUNSEL, 2K BUSINESS AFFAIRS
 Jerry Wang

COUNSEL
 Justyn Sanderford

VP OF BUSINESS DEVELOPMENT
 Steve Lux

DIRECTOR OF OPERATIONS
 Derian Redfield

LICENSING/OPERATIONS SPECIALIST
 Xenia Mut

OPERATIONS MANAGER
 Ben Kvalb

OPERATIONS COORDINATOR
 Peter Driscoll

2K IT
DIRECTOR, 2K IT
 Robi Roudabush

IT MANAGER
 Bob Jones

SR. NETWORK/SYSTEMS ENGINEER
 Russel Mauns

SYSTEMS ENGINEERS
 Jon Fygiak
 Lee Ryan

SYSTEMS ADMINISTRATOR
 Fernando Ramirez

SR. SYSTEMS ADMINISTRATORS
 Tareq Abbassi
 Scott Alexander
 Davis Kriegerhoff

IT ANALYST
 Michael Crescia

2K INTERNATIONAL GENERAL MANAGER
 Neil Ralley

INTERNATIONAL PRODUCT MANAGER
 Ediz Bassol

INTERNATIONAL PR MANAGER
 Wouter van Vugt

INTERNATIONAL SOCIAL & COMMUNITY EXECUTIVE
 Catherine Vandier

WEB DESIGNER
 Keith Edavartia

WEB DEVELOPER
 Alex Beuschler

WEB PRODUCER
 Tiffany Nelson

CHANNEL MARKETING MANAGERS
 Anna Agoyev
 Marek Gandy

DIGITAL MARKETING COORDINATOR
 Kelsie Lathi

SR. DIRECTOR OF EVENTS
 Lesley Zinn Abatear

EVENTS MANAGER
 David Iskra

SR. MANAGER, COMMUNITY AND SOCIAL MEDIA
 Ronnie Singh

COMMUNITY AND SOCIAL MEDIA COORDINATOR
 Chris Manning

DIRECTOR, CUSTOMER SERVICE
 Lina Somers

CUSTOMER SERVICE MANAGER
 David Eggers

KNOWLEDGE BASE COORDINATOR
 Mike Thompson

SR. MANAGER OF PARTNERSHIPS & LICENSING
 Jessica Hopp

PARTNER MARKETING MANAGER
 Dawn Eerp

DIGITAL MARKETING COORDINATOR
 Ashley Lantry

MARKETING ASSISTANT
 Jes-sica Perez

ADMINISTRATIVE ASSISTANT
 Dino Sulprizio

2K OPERATIONS
 VP, STUDIO OPERATIONS
 Kane Kellogg

SVP, SR. COUNSEL
 Peter Welch

2K ASIA
ASIA SR. PUBLISHING DIRECTOR
 Jason Wong

ASIA SR. MARKETING MANAGER
 Diana Yan

ASIA MARKETING MANAGER
 Daniel Tan

JAPAN MARKETING MANAGER
 Maho Sawashima

PRODUCT EXECUTIVE
 Rohan Ishwardal

PRODUCT EXECUTIVE
 Sharon Lim

LOCALIZATION MANAGER
 Yosuke Tano

LOCALIZATION COORDINATOR
 Pierre Gujjarro

LOCALIZATION ASSISTANT
 Yasutaka Ahta

TAKE-TWO ASIA OPERATIONS
 Eileen Chong
 Veronika Khuan
 Chenmei Tan
 Blacko Davis
 Ryoko Higashi

TAKE-TWO ASIA BUSINESS DEVELOPMENT
 Erik Ford
 Tanya Galloway
 Eileen Hsu
 Kelvin Ahn
 Paul Adebisi
 Fumiko Okura
 Fredrick Tan
 Fredrick Tan
 Julius Chen
 Ken Tikaratna
 Albert Hoolsema

2K QUALITY ASSURANCE
SR. VICE PRESIDENT OF QUALITY
 Alan Pichonowski

QUALITY ASSURANCE TEST MANAGER
 Jeremy Ford

QUALITY ASSURANCE TEST, MANAGER - SUPPORT TEAMS
 Scott Stribrod

PROJECT LEAD
 Shane Giffin

LEAD TESTER
 Chris Adams
 Nathan Bell

ASSOCIATE LEAD TESTERS
 Jorge Corpeho
 Phyllis Fletcher
 Adam Gonzalez
 Leticia Rodriguez
 Dawayne Roberto Wilbert, Jr.

SENIOR TESTERS
 David Drake
 Zack Garner
 Ann Garza
 Robert Klampner
 Philip Lui

QUALITY ASSURANCE TESTERS
 Travis Allen
 Adharae Anderson
 Tanya Galloway
 Eduardo Artuz
 Steven Barding
 Christopher Bludista
 Kyle Bolas
 Michael Bond
 Corey Bradley
 Sampson Brer
 Matt Casarillo-Ureno
 Kyle Cobos
 Nathan Craig
 Joshua Collins
 Casey Costa-Lok Hoyt
 Bryce Fernandez
 Jan Flugum
 Zach Giffin
 Henry Handley
 John Hanifzai

2K INTERNATIONAL QUALITY ASSURANCE
LOCALIZATION QA MANAGER
 Jose Medina

MASTERING ENGINEER
 Wayne Boyce

MASTERING TECHNICIAN
 Alan Vincent

LOCALIZATION QA SENIOR LEAD
 Oscar Peniza

PAUL HEADERS ON
 Daniel Lim
 Greg Jefferson
 Chris Johnson
 Jemal Jordan-Butler
 James Kautz
 Cissie Kautz
 Johnathon Luk
 David Longolo
 Jason Malleman
 Jason Malina
 Cesar Marquez
 Quincy McGee
 Lin Mei
 Enrique Meza
 Jose Miras
 Eda Nalawski
 Philip Owens
 Josh Roy
 Brian Reiss
 Chris Rippey
 Adam Roberts
 Max Rubin
 Gabbi Ronquillo
 Robert San Agustin
 Daniel Smyth
 John Spill
 John Spillora
 Allan Thomas
 Washington Thompson III
 Dominic Willis
 Justin Wright
 Justy Wright
 Alexis White
 Anthony Zaragoza

SPECIAL THANKS
 Lele Calton
 Alex Baik
 Louis Napolitano
 Joe Boris
 David Bernscale
 John Berman
 Richard Holowski
 Chris Jones
 Kris Jolly
 Adam Corral
 Eric Hays
 Todd Ingram

LOCALIZATION QA PROJECT LEAD

Fabrizia Mariani
 LOCALIZATION QA LEADS
 Karim Chertif
 Florian Gambon
 ASSOCIATE LOCALIZATION QA LEAD
 Cristiana La Mira

SENIOR LOCALIZATION QA TECHNICIANS

Alba Loureiro
 Christopher Funke
 Emilio Riera
 Fernando Robles
 Jihye Kim
 Johanna Cohen
 Jose Olivares
 Pierre Tissot

LOCALIZATION QA TECHNICIANS

Christiane Molin
 David Swan
 Dimitri Gerard
 Galina Ushakov
 Galina Ushakov
 Giuliano Castaford
 Iris Loison
 Javier Vizil
 Julie White
 Julia Calle
 Luca Magini
 Manuel Aguayo
 Martin Schlocker
 Nicolas Bedi
 Nicolas Bedi
 Norma Hernandez
 Pablo Menendez
 Roland Hebersack
 Rudiger Kolb
 Sergio Acosta
 Simon Hess
 Sergio Acosta
 Shawn Williams-Brown
 Sharif Mahdy Farag
 Timothy Cooper

2K CHINA QUALITY ASSURANCE

QA DIRECTOR
 Zhang Xi Kun
 QA SUPERVISOR
 Steve Manners
 QA LEADS
 Gao You Ming
 Huang Cheng

QA SENIOR TESTERS

Wang Yi Min
 Shao Bang Zhu
 QA TESTERS
 Bai Bai Long
 Cai Kuang Yu
 Cao Kuli
 Cheng Feng
 Deng Chun Chao
 Deng Jian
 Deng Yang
 Hu Die
 Huang Chang
 Jiang Xiang
 Jiang Xiao Yu
 Kong Wei Yu
 Lai Yi Peng
 Li Gang
 Li Hong

QA TESTERS

Zhou Qian Yu
 Hu Meng Meng
 Zhang Li
 Zhao Liang
 Mao Jing Jie
 Yan Yan
 Yu Le
 JUNIOR QA TESTERS
 Zhou Qian Yu
 Hu Meng Meng
 Zhang Li
 Zhao Liang
 Mao Jing Jie
 Yan Yan
 Yu Le

IT ENGINEERS

Zhao Hong Wei
 Hu Xiang
 FOX STUDIOS
 Rick Fox
 Michael Weber
 Tom Schmidt
 Keith Fox
 Dustin Smith
 Joe Schmidt
 NATIONAL BASKETBALL ASSOCIATION
 PRESIDENT, GLOBAL OPERATIONS & MERCHANDISING
 Salvatore Larocca
 EXECUTIVE VICE PRESIDENT, GLOBAL MARKETING PARTNERSHIPS
 Emilio Collins
 VICE PRESIDENT, GLOBAL MARKETING PARTNERSHIPS
 Andrew Kelly
 VICE PRESIDENT, LEGAL & BUSINESS AFFAIRS
 Irishi Karthikeyan
 VICE PRESIDENT, LICENSING
 Matt Holt

SENIOR QA TESTERS

Zhu Jun Dan
 QA TESTERS
 Yan Liu Yang
 Kan Liang
 Ning Xu
 Cho Hyunmin

JUNIOR QA TESTERS

Yu Le
 Zhou Qian Yu
 Hu Meng Meng
 Zhang Li
 Zhao Liang
 Mao Jing Jie
 Yan Yan
 Yu Le

IT ENGINEERS

Zhao Hong Wei
 Hu Xiang
 FOX STUDIOS
 Rick Fox
 Michael Weber
 Tom Schmidt
 Keith Fox
 Dustin Smith
 Joe Schmidt
 NATIONAL BASKETBALL ASSOCIATION
 PRESIDENT, GLOBAL OPERATIONS & MERCHANDISING
 Salvatore Larocca
 EXECUTIVE VICE PRESIDENT, GLOBAL MARKETING PARTNERSHIPS
 Emilio Collins
 VICE PRESIDENT, GLOBAL MARKETING PARTNERSHIPS
 Andrew Kelly
 VICE PRESIDENT, LEGAL & BUSINESS AFFAIRS
 Irishi Karthikeyan
 VICE PRESIDENT, LICENSING
 Matt Holt

SENIOR MANAGER, LEGAL & BUSINESS AFFAIRS

Vince Kearney
 SENIOR COORDINATOR, LICENSING
 Greg Brownstein
 SPECIALIST, LICENSING
 Winnie Song
 SENIOR ACCOUNT EXECUTIVE, GLOBAL MEDIA
 Arlie Outrone
 COORDINATOR, GLOBAL MARKETING PARTNERSHIPS
 Jen Murphy

MOTION CAPTURE TALENT

NBA TALENT
 Harrison Barnes
 Kent Bazemore
 Trey Burke
 Will Clyburn
 Stephen Curry
 Brandon Davies
 Dennis Exum
 PJM Hollins
 Will Ojeda
 Ben McLemore
 James Nunnally
 Austin Rivers
 Lance Stephenson
 Dion Williams
 BASKETBALL TALENT
 Antonio Biglow
 Jake Bobjan
 Myre "Kremix" Bowden
 Michael Bowens, Jr.
 Justin Brown
 Collin Cateagat
 Collin Cateagat
 Joell Crawford
 Roy Giles
 Dominique Grant
 Jim Harris
 Jim Harris
 P.J. Sten Howard
 Allen Huddleston
 Tony Johnson
 John Jordan
 Mike McChristian
 Corey McKinney
 Mike McKinney
 Xavier McNally
 Karam Nizoz
 Michael Nunnally

SENIOR MANAGER, LEGAL & BUSINESS AFFAIRS

Vince Kearney
 SENIOR COORDINATOR, LICENSING
 Greg Brownstein
 SPECIALIST, LICENSING
 Winnie Song
 SENIOR ACCOUNT EXECUTIVE, GLOBAL MEDIA
 Arlie Outrone
 COORDINATOR, GLOBAL MARKETING PARTNERSHIPS
 Jen Murphy

MOTION CAPTURE TALENT

NBA TALENT
 Harrison Barnes
 Kent Bazemore
 Trey Burke
 Will Clyburn
 Stephen Curry
 Brandon Davies
 Dennis Exum
 PJM Hollins
 Will Ojeda
 Ben McLemore
 James Nunnally
 Austin Rivers
 Lance Stephenson
 Dion Williams
 BASKETBALL TALENT
 Antonio Biglow
 Jake Bobjan
 Myre "Kremix" Bowden
 Michael Bowens, Jr.
 Justin Brown
 Collin Cateagat
 Collin Cateagat
 Joell Crawford
 Roy Giles
 Dominique Grant
 Jim Harris
 Jim Harris
 P.J. Sten Howard
 Allen Huddleston
 Tony Johnson
 John Jordan
 Mike McChristian
 Corey McKinney
 Mike McKinney
 Xavier McNally
 Karam Nizoz
 Michael Nunnally

SENIOR MANAGER, LEGAL & BUSINESS AFFAIRS

Vince Kearney
 SENIOR COORDINATOR, LICENSING
 Greg Brownstein
 SPECIALIST, LICENSING
 Winnie Song
 SENIOR ACCOUNT EXECUTIVE, GLOBAL MEDIA
 Arlie Outrone
 COORDINATOR, GLOBAL MARKETING PARTNERSHIPS
 Jen Murphy

MOTION CAPTURE TALENT

NBA TALENT
 Harrison Barnes
 Kent Bazemore
 Trey Burke
 Will Clyburn
 Stephen Curry
 Brandon Davies
 Dennis Exum
 PJM Hollins
 Will Ojeda
 Ben McLemore
 James Nunnally
 Austin Rivers
 Lance Stephenson
 Dion Williams
 BASKETBALL TALENT
 Antonio Biglow
 Jake Bobjan
 Myre "Kremix" Bowden
 Michael Bowens, Jr.
 Justin Brown
 Collin Cateagat
 Collin Cateagat
 Joell Crawford
 Roy Giles
 Dominique Grant
 Jim Harris
 Jim Harris
 P.J. Sten Howard
 Allen Huddleston
 Tony Johnson
 John Jordan
 Mike McChristian
 Corey McKinney
 Mike McKinney
 Xavier McNally
 Karam Nizoz
 Michael Nunnally

SENIOR MANAGER, LEGAL & BUSINESS AFFAIRS

Vince Kearney
 SENIOR COORDINATOR, LICENSING
 Greg Brownstein
 SPECIALIST, LICENSING
 Winnie Song
 SENIOR ACCOUNT EXECUTIVE, GLOBAL MEDIA
 Arlie Outrone
 COORDINATOR, GLOBAL MARKETING PARTNERSHIPS
 Jen Murphy

MOTION CAPTURE TALENT

NBA TALENT
 Harrison Barnes
 Kent Bazemore
 Trey Burke
 Will Clyburn
 Stephen Curry
 Brandon Davies
 Dennis Exum
 PJM Hollins
 Will Ojeda
 Ben McLemore
 James Nunnally
 Austin Rivers
 Lance Stephenson
 Dion Williams
 BASKETBALL TALENT
 Antonio Biglow
 Jake Bobjan
 Myre "Kremix" Bowden
 Michael Bowens, Jr.
 Justin Brown
 Collin Cateagat
 Collin Cateagat
 Joell Crawford
 Roy Giles
 Dominique Grant
 Jim Harris
 Jim Harris
 P.J. Sten Howard
 Allen Huddleston
 Tony Johnson
 John Jordan
 Mike McChristian
 Corey McKinney
 Mike McKinney
 Xavier McNally
 Karam Nizoz
 Michael Nunnally

Davidson College
Arizona State University
University of Kansas
UCLA
University of Louisville
University of Maryland
University of Minnesota
University of Connecticut
University of Michigan
Villanova University
University of Wisconsin

LIMITED SOFTWARE WARRANTY AND LICENSE AGREEMENT

This limited software warranty and license agreement ("Agreement") is made between you ("User") and the software provider ("Provider"). The Agreement is subject to the terms and conditions of the software license agreement ("License Agreement") and the software provider's standard terms and conditions of sale ("Standard Terms and Conditions"). The Agreement is made in consideration of the User's purchase of the software and the Provider's provision of the software to the User. The Agreement is made in full and final settlement of all claims and disputes between the User and the Provider regarding the software and the Provider's standard terms and conditions of sale.

WARRANTY. The Provider warrants that the software will perform substantially in accordance with the software license agreement and the software provider's standard terms and conditions of sale for the period of time specified in the software license agreement and the software provider's standard terms and conditions of sale. The Provider does not warrant that the software will be error-free or that it will be compatible with all hardware configurations. The Provider does not warrant that the software will be suitable for any particular purpose or that it will be free from viruses or other malicious code. The Provider does not warrant that the software will be updated or that it will be supported by the Provider after the end of the warranty period.

LICENSE CONDITIONS. The software is provided to the User under a limited software license. The User agrees to use the software in accordance with the software license agreement and the software provider's standard terms and conditions of sale. The User agrees not to copy, modify, or distribute the software or any part thereof. The User agrees not to use the software for any purpose that is prohibited by applicable law. The User agrees to indemnify the Provider from and hold the Provider harmless from any claims and damages arising out of the User's use of the software.

TERMINATION. The Provider reserves the right to terminate this Agreement if the User breaches any of the terms and conditions of the software license agreement or the software provider's standard terms and conditions of sale. Upon termination, the User agrees to stop using the software and to delete all copies of the software from the User's system. The Provider does not warrant that the software will be available for use after the termination of this Agreement.

FORCE MAJEURE. If the performance of this Agreement is prevented, hindered, or delayed by a cause beyond the control of the Provider, the Provider shall not be liable for any delay or non-performance in such circumstances. The Provider shall use its best efforts to overcome the cause of the delay and to resume performance as soon as possible.

ASSIGNMENT. The User may not assign, transfer, or otherwise dispose of the software or any part thereof. The Provider may assign, transfer, or otherwise dispose of the software or any part thereof. The Provider may assign, transfer, or otherwise dispose of the software or any part thereof to its affiliates, subsidiaries, or successors in interest.

ENTIRE AGREEMENT. This Agreement, together with the software license agreement and the software provider's standard terms and conditions of sale, constitute the entire agreement between the User and the Provider regarding the software and the Provider's standard terms and conditions of sale. No oral or written agreement, understanding, or course of dealing shall be construed to modify, supplement, or otherwise affect the terms and conditions of this Agreement.

GOVERNING LAW. This Agreement shall be governed by the laws of the state of California. The parties agree to submit to the jurisdiction of the courts of the state of California and to the arbitration of the American Arbitration Association. The arbitration shall be held in San Francisco, California.

NOTICES. All notices under this Agreement shall be in writing and shall be sent to the Provider at the address set forth in the software license agreement and the software provider's standard terms and conditions of sale. Notices to the User shall be sent to the email address set forth in the software license agreement and the software provider's standard terms and conditions of sale.

SEVERABILITY. If any provision of this Agreement is held to be unenforceable or invalid, the remaining provisions of this Agreement shall remain in full force and effect. The parties agree that the provisions of this Agreement are severable and that the invalidity of one or more provisions shall not affect the validity of the remaining provisions.

FORCE MAJEURE. If the performance of this Agreement is prevented, hindered, or delayed by a cause beyond the control of the Provider, the Provider shall not be liable for any delay or non-performance in such circumstances. The Provider shall use its best efforts to overcome the cause of the delay and to resume performance as soon as possible.

ASSIGNMENT. The User may not assign, transfer, or otherwise dispose of the software or any part thereof. The Provider may assign, transfer, or otherwise dispose of the software or any part thereof. The Provider may assign, transfer, or otherwise dispose of the software or any part thereof to its affiliates, subsidiaries, or successors in interest.

ENTIRE AGREEMENT. This Agreement, together with the software license agreement and the software provider's standard terms and conditions of sale, constitute the entire agreement between the User and the Provider regarding the software and the Provider's standard terms and conditions of sale. No oral or written agreement, understanding, or course of dealing shall be construed to modify, supplement, or otherwise affect the terms and conditions of this Agreement.

GOVERNING LAW. This Agreement shall be governed by the laws of the state of California. The parties agree to submit to the jurisdiction of the courts of the state of California and to the arbitration of the American Arbitration Association. The arbitration shall be held in San Francisco, California.

NOTICES. All notices under this Agreement shall be in writing and shall be sent to the Provider at the address set forth in the software license agreement and the software provider's standard terms and conditions of sale. Notices to the User shall be sent to the email address set forth in the software license agreement and the software provider's standard terms and conditions of sale.

SEVERABILITY. If any provision of this Agreement is held to be unenforceable or invalid, the remaining provisions of this Agreement shall remain in full force and effect. The parties agree that the provisions of this Agreement are severable and that the invalidity of one or more provisions shall not affect the validity of the remaining provisions.

EXHIBIT 11

**THIS EXHIBIT HAS BEEN
REDACTED IN ITS ENTIRETY**

EXHIBIT 12



News Devotionals Blogs Backgrounds Images Videos Kids Games

Search

Tell us what you think



3 Easy Steps

1. Click "Start"
2. Add Extension
3. Start Converting



report this ad

Home >> News >> NBA 2K15 Update: 2K Released Major Patch 4 for Xbox One, PS4, PC; NBA 2K16 October Release Confirmed?

NBA 2K15 Update: 2K Released Major Patch 4 for Xbox One, PS4, PC; NBA 2K16 October Release Confirmed?

By Shane



Email Share Print Comment

Text size: Small Medium Large



report this ad

2K Sports has recently released NBA 2K15 patch 4 and now available for the PC, PS4 and Xbox One versions of the game. The fourth patch update for the basketball sim fixes a lot of issues and bugs.

The complete list of notes for NBA 2K15 patch 4 are as follows:

- (XB1) Online games will no longer disconnect during gameplay when one or more users had minimized the title and returned while sitting on the Team Select screen prior to playing.
- Made a number of improvements to the player progression/regression systems that drive player ratings in MyGM/MyLEAGUE/MyCAREER.
- Fixed a rare case where the user team would trade for the upcoming pick mid-draft (MyGM/MyLEAGUE), but the team that traded away the pick would retain the rights to the player.
- Fixed an issue where some users were unable to access specific Online Leagues.
- Removed "auto-hesitation" dribble animations that played when driving near defenders. Smoothed out the sprint to jog transition in the back court.
- Tuned shot percentages for a more obvious distinction between open and covered shots. Slight increase to contact dunk frequency for high level dunkers.
- Corrected an issue in MyTEAM where improved player badges were not appearing in the correct color when viewing a head-to-head game. The correct badge level with appropriate gameplay adjustments are now being made.
- The 'Bruiser' and 'Hardened' badges will now unlock properly in MyCAREER when the appropriate conditions are met.
- Fixed a rare case where some users would occasionally see the title hang during the series of legal screens at the start of the game.
- (XB1) Corrected an issue where Park Chat through the My NBA 2K15 app would not work for some users.
- (XB1) Fixed an issue where a user's gamertag would not update in the Roster Creator interface when the gamertag was changed outside of the title.
- NOTE: All patch fixes will work in your existing game mode saves.

In related news, the servers for NBA 2K14 are officially shut off effective April 1. "Players will no longer be able to play ranked games/online leagues," 2K stated.

Meanwhile, as we have [reported previously](#), the all new NBA 2K16 is anticipated to release in October and set to bring something new from the previous game installment including a full body scan feature according to Latino Post.

Right Now



Dr. Michael Youssef calls for Uncompromising Faith in the wake of London Terror Attacks



The Year of the Reformation Officially started in Russia



'Six weeks' medicine and two months' food for displaced Iraqi Christians'



Bus gets window smashed, graffitied in broad daylight... for saying boys and girls are different

The said report noted that rumors have speculated Harden to replace Kevin Durant on the cover of the basketball gaming title but that is yet to be confirmed.

With the full body scan of the NBA players, Good Game Bro cited that 2K Sports is dwelling more on improved body types for players in this upcoming release expected to offer better graphic improvements than its predecessors by giving a life-like possible gaming experience to the players.

NBA 2K15 patch 4 update won't be available for players who own the PS3 or Xbox 360 version of the game. 2K said, "At this time, there are no updates to announce for NBA 2K15 on PlayStation 3 and Xbox 360."

Tags nba 2k15, nba 2k15 patch 4, nba 2k15 patch 4 release, nba 2k15 update



SHOP DECOR

report this ad

ADVERTISEMENT

Trending Now

Ads by Revcontent



Md: Do This Immediately If You Have Diabetes (Video)

Your Online Insider



All Natural Pain Treatment for Your Pets

Innovet



Born Before 1985? Gov't Will Pay Up to \$355/month off Your Mortgage

SavvyFinance.org



Oncologists Are Flipping Out After True Cause of Cancer is Released

Natural Health Solutions



Toenail Fungus Gone if You Do This (Before Bed)

Life Naturals



This 1 Tip Has Helped 1,000s of Homeowners Save

Homeowner News



20+ Places on Earth You Need to See Once in Your Life

Buzzworthy



20+ Real People with Superhuman Abilities

Buzzworthy



22 Daring Women That Will Inspire Your Standards Of Beauty

Buzzworthy

report this ad

Recommended



Health care for all isn't optional, Vatican tells UN



LGBTQ ratings flop: Americans keep rejecting 'gay' programming



Beauty And The Beast director: 'I wish I could say I... rip pages out of the Bible'



Supreme Court Judge: Marriage, Religious Liberty under attack

report this ad

Resources



365 Moments Of Peace In God--Seek You



I need Jesus



Consecrated Feet



Crown of Beauty

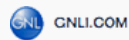
CROSSMAP Copyright © 2000-2018 Crossmap.com. All rights reserved.

[News](#) [Devotionals](#) [Blogs](#) [Backgrounds](#) [Images](#) [Videos](#) [Kids](#) [Games](#) [Advertising](#)

[About Us](#) [Statement of Faith](#) [Privacy Policy](#) [Terms and Conditions](#) [Contact Us](#)

MEDIA PARTNERS

THE CHRISTIAN POST



BREATHE east

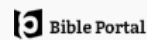


EXHIBIT 13

NBA 2K16 Official Patches

This is a **list of official patches** (also referred to as **title updates**) that have been released for [NBA 2K16](#), along with their release notes. The most recent patch is V5, released on January 25th, 2016 for PC, PlayStation 4, and Xbox One.

Contents

NBA 2K16 Title Update V6

- Release Dates
- Patch Notes

NBA 2K16 Title Update V5

- Release Dates
- Patch Notes

NBA 2K16 Title Update V4

- Release Dates
- Patch Notes

NBA 2K16 Title Update V3

- Release Dates
- Patch Notes

NBA 2K16 Title Update V2

- Release Dates
- Patch Notes (PlayStation 4 & Xbox One)
- Patch Notes (PC)

NBA 2K16 PC Hotfixes

Links

NBA 2K16 Title Update V6

Release Dates

- **PC:** May 19th
- **PlayStation 4:** May 2nd
- **Xbox One:** May 5th

Patch Notes

- Updated player portraits for a number of players.
- Tuned the effectiveness of long outlet passes for players with the Break Starter badge.
- Delay branching from Kobe Bryant Back size-up.
- Re-worked size-up dribble launches to prevent unnaturally explosive first steps.
- Tune miss chances for deep downcourt and cherry picking.
- Dock vertical and rebounding when you go behind the basket. Don't reset 3-in-the-key offense timer in baseline behind paint for human controlled blacktop players.

NBA 2K16 Title Update V5

Release Dates

- **PC:** January 25th
- **PlayStation 4:** January 25th
- **Xbox One:** January 25th

Patch Notes

- Fixed an issue where users were able to equip mascot costumes despite not having the appropriate rep level in MyPARK.
- Addressed a game disconnect that would sometimes occur during scrimmage games in MyCOURT when there are spectators.
- Fixed an issue that was causing unwanted ankle breaker reactions on defenders who were standing in good defensive position.
- Resolved a rare case where a Pro-Am Arena game would stall out on the loading screen.
- (PS4) Adjusted the 'How Much Ya Got?' requirements such that a maximum salary offer will unlock the Trophy (regardless of how many years the contract offer is for).
- Added 'Player of the Week' and 'Player of the Month' panels to the MyPARK leaderboards. Take a look!

NBA 2K16 Title Update V4

Release Dates

- **PC:** December 15th
- **PlayStation 4:** December 15th

- **Xbox One:** December 15th

Patch Notes

GENERAL

- Addressed a hang that can occur when editing/applying tattoos in Edit Player.
- Resolved a bug where one or more users in a co-op game would lose control of their player at some point during the game.
- Fixed a disconnect issue that some users were seeing during online games.
- (PC) Fixed a corruption issue where some users were experiencing hangs, missing art assets, and infinite loads. Please ensure that your virus scanning software (e.g. Avast) whitelists your Steam folder/directory. This is required for new content to properly download and install.
- Closed a loophole where the pause timer would stop counting down when sitting in the 'Report Image' flow.
- Improved the user interface when applying contracts to players in MyTEAM by removing the display of how many contracts the player would have IF the current card was applied.
- Fixed a case where the user accepting an online invite would sometimes skip the position selection screen in the way into matchmaking.
- Headwear should now properly unequip when attempting to do so from within the MyPARK Apparel section of the store.
- The player portrait will now update when a new user is signed in on the main menu via the account picker.
- A number of other optimizations and enhancements have been made to improve the user experience.

GAMEPLAY

- Teammate free throw celebration/interaction behaviors have been corrected.
- Slightly tuned down the frequency of ball collisions.
- Fixed an issue that led to a temporary loss of user control on possession changes while in the 2K Cam.
- Tuned down the frequency of missed alley-oop and putback dunks when open.
- Fixed an issue that allowed high rated post players to shoot post fadeaway 3's at a high percentage.

MyCAREER/MyPARK/MyCOURT

- Fixed a rare issue where the game would hang when Doris Burke interviews your MyPLAYER at the end of the first half and/or at the end of the game.
- The correct records and rep levels will now be shown for the players on the MyPARK loading screen.
- Corrected the wall and floor art for the San Antonio Spurs theme in MyCOURT.
- Fixed a case where some purchased shoes were not properly appearing on the shoe racks in MyCOURT.

MyGM/MyLEAGUE

- Resolved a hang that could occur many, many years into MyGM when entering the offseason.
- Generated players will now wear the correct color socks for away games.
- Fixed an issue where generated players were getting their free throw animations randomized every offseason.
- The Strengths/Weaknesses screen in Team Comparison should now be functioning properly once again.

NOTE: All patch fixes will work in your existing game mode saves (unless specifically noted otherwise).

NBA 2K16 Title Update V3

Release Dates

- **PC:** November 20th
- **PlayStation 4:** November 13th
- **Xbox One:** November 25th

Patch Notes

GENERAL

- (PC) Fixed an issue where commentary would cut out for some users.
- (PC) Added a "Full Court" option to the "Broadcast Generic" camera; intended for use with 21:9 or wider screens.
- (PC) Corrected a case where instant replays were too short for users running faster than 60 FPS.
- (PC) Fixed a case where the game was unable to launch for some users.
- (PC) Resolved a hardware-specific audio driver hang that some users were encountering.
- Added support to view the uniform your opponent selects prior to starting a Play Now Online game.
- Flip Saunders patch added to Minnesota Timberwolves uniforms.
- A number of accuracy-based improvements have been made to classic team uniforms (07-08 Celtics, 99-00 Raptors, 05-06 Heat, etc.)
- A number of improvements have been made with the 2KTV viewing experience. Make sure and watch the latest episode for your chance to earn some easy VC!
- Improved the results of face scanning.

- Primary Jersey and Primary Shorts colors are now used to determine both accessory and sock colors when using custom-built uniforms.
- Certain materials were not properly appearing on user-edited shoes; this should be resolved.
- Fixed an issue where users would not join the Locker Room when accepting an invite while playing in a MyCAREER game.
- Corrected a disconnect issue that could happen when pausing and unpausing the game during online play.
- Fixed an issue in MyTEAM where players/items from the 99-00 Blazers were returning when searching for the Timberwolves in the Auction House.

GAMEPLAY

- Resolved an issue where too much stamina would recover during timeouts (giving the effect that players never tired).
- Finally resolved an issue where players would not sweat appropriately in longer quarter lengths.
- Fixed a rare case where you would be unable to dismiss the quick sub menu when attempting to make a substitution.
- Found a case where the camera would be pointing towards the crowd at the tip-off of an overtime game.
- Corrected the Free Throw Difficulty slider, which was flipped. Make sure to update your custom sliders if you accounted for this on your end already.
- Outdoor sound effects should no longer play in indoor environments after having previously played a MyTEAM Gauntlet game.
- Players who have not yet had their in-game injury diagnosed will have their playable status updated once the diagnosis is complete.
- Players who have not yet had their in-game injury diagnosed will correctly suffer from the effects of the actual injury.
- Players who pick up short term injuries (lasting only minutes long) should now have them properly removed when the remaining duration reaches zero during a game.
- Tuned the 'hold ball too long' event in the teammate grade logic to be a little more forgiving.
- Fixed an issue where you could be awarded 'good foul' in your teammate grade when fouling a three-point shooter.
- We no longer give an 'allow offensive rebound' teammate grade penalty after blocking an opponent's shot.
- Addressed a movement issue that would cause players to twitch while in Deny states.
- Referees should no longer twitch during free throws.
- Pass receivers are now less likely to veer away from the basket to catch a pass.
- Improved CPU logic for when they should and should not use alley-oop passes.
- Reduced the defensive impact of defenders who are on their way down after a block attempt.
- Fixed a very rare case where a free throw shot would launch toward the wrong basket.
- Tuned body-up system to prevent sub-par defenders from being overly effective at stopping the ball handler.
- Fixed a frequent hitch in the dribble system seen when branching into certain dribble collisions.
- Tightened up 'crowded' dribble launches to prevent unwanted hesitations when attempting to drive to the basket.
- Fixed post-up double dribble and travel (pivot) issues.
- Tuned down excessive flopping being performed by AI players.
- Addressed the issue of 'camping' in MyPARK games by: 1) Disabling the set screen button within 6 feet of the basket, 2) Allowing users to initiate off-ball contact with screen setters near the rim, and 3) By adding offensive 3 seconds rule to half court games.
- Fixed a case that was causing passers to stop to throw a pass in the backcourt.
- Tuned the dribble system in order to tone down the effectiveness of behind-the-back dribble moves and 'zig-zagging' back and forth to get open.
- Improvements made to alley-oop defense.
- Fixed an issue that would prevent some contact layups from respecting the user's desired finish direction.
- Reduced the frequency of unwanted contested shot fadeaways.
- Tweaks to errant pass frequency when passing out of double teams and contact shots.
- Made dive for loose balls more accurate and more likely to acquire the ball.
- Swapped out 'Normal 13' jump shot pullups per community feedback.
- Made low-rated shot blockers less accurate and capped their jumping verticals.
- Made decisions by the CPU to steal a pass more reasonable through tuning the frequencies in certain situations.
- Improvements to double team logic.
- Increased the difficulty in stealing the ball from post players.
- Improvements to logic system for when 'ankle breakers' should play and when they should not play.
- Fixed an issue in which points would not count for shots that go in after a flagrant foul call (i.e. And-1 on flagrants).
- Fixed a MyTEAM post move score exploit by better defining post shot scenarios.
- Added jam ups to spin layups to prevent the offense from barreling through heavy traffic.
- Improved CPU logic for when to hustle back in transition defense.
- Added a pop-up overlay to let the user know when the Adaptive Coaching Engine is changing their freelance offense.
- Adjusted freeland motion cut times and fixed an issue with swing freelance starting positions.
- Adjusted stationary hop jumper shooting percentages to make them a more viable option on offense.

- Tuned back the frequency of missed dunks in traffic.
- Fixed an issue where if someone called for an icon pick and got interrupted by a stealing defender, the ball handler was previously unable to pass the ball by any means for the remainder of the possession.
- Improvements to help defense and rotation logic.

MyPARK/2K PRO-AM

- You now have the ability to design an 'Away' uniform for your Pro-Am team. The appropriate uniform will be worn depending on if your team is Home or Away for the current game.
- The correct overall rating for your MyPLAYER will now be shown on the Pro-Am loading screen.
- Improved voice chat support in both MyPARK and 2K Pro-Am.
- Accolades will now unlock based on your performances in custom arena Pro-Am games.
- A number of fixes have been applied to improve lineup building during the Pro-Am.
- Made CPU players that take over for quitting users less effective in Pro-Am.
- A Pro-Am game will end immediately with a win if all opposing players have quit out of the game.
- 'Quit' deterrents have been added to deter users from quitting out of Pro-Am games. Too many quits will result in the short term inability to play the mode.
- Removed the 10-second delay that would occur before the start of a game to 21 in MyPARK.
- Fixed an issue with the squad countdown timer displaying incorrectly when traveling between parks.
- Users can no longer celebrate when their opponent scores in 21.
- Camera flashes should no longer be seen in empty custom-arena matches. Progress further in your MyCAREER to draw more people to your Pro-Am games!
- Fixed a very rare issue where some users were reporting a hang when exiting their MyPARK.
- The ball should no longer get stuck to your MyPLAYER when celebrating in the park.
- Corrected an issue where some users were not receiving the correct amount of VC after winning a MyPARK Ante-Up game.
- Fixed an issue where stats would sometimes show as incorrect in the Game Recap following a Pro-Am game.
- Fixed an issue where certain players were unable to travel back to their home parks at the end of Rival Days due to travel restrictions.
- A large number of other MyPARK performance fixes and enhancements have been made to improve the overall experience.

MyCAREER/MyCOURT

- Fixed an issue where certain store rewards (from Connections) weren't saving upon unlocking. Previously unlocked items should again show up as owned items.
- Added support for crowd members to hold up 'big head' posters of your MyPLAYER.
- Users should now start seeing 'photobombs' again once they reach their second NBA season in MyCAREER.
- Custom code added to prevent very similar shades of white/grey/black uniforms from being matched up automatically as part of the 'alternate uniform night' feature. This logic also applies to the MyGM/MyLEAGUE modes.
- Fixed a MyCAREER issue where your contact could inadvertently change after being traded between two teams.
- Improved PlayVision to better help you understand what you should be doing on offense in MyCAREER.
- Reduced the max amount of scheduled minutes your MyPLAYER can get to 38 (pro-rated) minutes. Actual playtime can vary based on game conditions, of course.
- MyPLAYERS who have retired from their NBA careers can now access their MyCOURTs.
- When not participating in a MyCOURT scrimmage, users can now freely walk around. Have fun.
- The game will no longer hang when attempting to equip the skateboard from the 2K Sports Store while in your MyCOURT.
- Sock length is now customizable for your MyPLAYER via the 2K Sports store.

MyGM/MyLEAGUE

- Tuned the fair market value for players; teams should now be signing players for the appropriate amount of money.
- Improved negotiation options when negotiating with Staff.
- Generated prospects should now have their own unique jumpshot animations, free throw animations, and signature size-ups. Note that this particular change will only work on GMs/LEAGUEs started post-patch.
- Improved support of the NBA's new tie-breaker rules for Playoff seeding.
- Teams are now less likely to trade away newly signed players, specifically those signed to longer contracts.
- The League Realignment option will now appear during every offseason in MyLEAGUE, even if a team was not relocated during the current season.
- Location/Team-specific crowd chants have now been disabled for relocated teams (as they no longer apply).
- Support has been added for users who wish to use position lock with multiple controllers.
- Added playcalling support during Scrimmages.
- If a user has relocated or rebranded a team with new uniforms, those uniforms will be used for Summer League/Scrimmage/Freestyle rather than the team's current practice uniforms.
- Players with the 'Force 6th Man' option turned on will no longer complain about wanting to be starters (and their expectation are adjusted accordingly).'
- Fixed an issue where changing your uniform for a Summer League game would result in the game being played in a non-Summer League environment.
- Injuries are now turned off during the Hoops Summit game in the offseason.
- Fixed a hang that could occur when jumping out of a game and into SimCast.

- Taking an invalid gameplan into a SimCast game will no longer cause the game to hang.
- Fixed an issue where hot spot player tendencies would get overwritten after changing them and leaving the Edit Player menu.
- Corrected an issue when using Player Lock where users would sometimes control all players, rather than just their own.
- Copying/Pasting/Deleting of gameplans should now function properly in MyLEAGUE Online.
- Trade Notifications in MyLEAGUE Online will now properly be deleted after being handled by the user.
- (PS4) Sony's invite system can now be used to invite users to your MyLEAGUE Online.
- Corrected an issue that would cause the game to stutter during a MyLEAGUE Online fantasy draft when the timer would run out with auto-draft enabled.
- The calendar now displays scheduled user games when one falls on the same day as the Player Re-signing deadline.
- Fixed an issue where 'Standing Shot Close' would sometimes be errantly displayed multiple times in the Training Camp Results overlay.
- Added 'Normalize Played to Sim Minutes' and 'Normalize Played to Sim Stats' to MyGM settings. Give them a try.

NOTE: All patch fixes will work in your existing game mode saves (unless specifically noted otherwise).

NBA 2K16 Title Update V2

Release Dates

PC: November 2nd PlayStation 4: October 14th Xbox One: October 21st

Patch Notes (PlayStation 4 & Xbox One)

GENERAL

- Resolved a rare case where the game would hang when entering/exiting the Tattoo editor when customizing the tattoos on your MyPLAYER.
- Addressed an issue on the main menu where the title would freeze for a period of time before proceeding to the selected action (e.g. loading a MyCAREER save).
- Tuned player sweat so that its effects are much more visible when playing longer quarter lengths.
- Players will no longer slide/skate when pressing the block button during an inbound state. It was funny while it lasted.
- Tuned crowd excitement and intensity on made buzzer beaters.
- Resolved a case where certain players (e.g. Kristaps Porzingis) would have their hairstyle change slightly when zooming in the camera very close in replay.

MyPARK/2K PRO-AM

- Pro-Am Team Play has been added to the game! You can now play as a full team in your custom-built arenas.
- Addressed a number of matchmaking issues that inhibited users from playing in Pro-Am Walk-On games.
- Fixed an issue where your player would sometimes appear with the incorrect height (if you had more than one MyCAREER save file) when entering MyPARK/Pro-Am from within MyCAREER.
- Fixed a rare case where the ball would get stuck in the High Roller courts, thus preventing a game from concluding.
- Improved voice chat support in both MyPARK and 2K Pro-Am.
- Improved the animation of the skateboard in MyPARK. If you don't have this, get it. You can move between courts faster than those that are on-foot!
- 2K Beats will now continue to play after finishing a game in MyPARK.

MyCAREER/MyCOURT

- Fixed an issue where some users were unable to join their friends' MyCOURTs.
- Fixed an issue where the host user was unable to send MyCOURT invites after leaving MyPARK/Pro-Am with a squad.
- Addressed a number of issues with Accolades not unlocking at the appropriate time within MyCAREER.
- Pressing the back button will no longer cause the game to hang from the MyCAREER NEXT menu after being eliminated from the playoffs.

MyGM/MyLEAGUE

- Numerous tuning and code enhancements to improve simulated player stats.
- Playoff games (within a series) can no longer be played out of order in MyLEAGUE Online.
- Found and fixed a case where visiting certain submenus with no head coach would cause the game to hang.
- Fans of relocated/rebranded teams will no longer hold up signs referencing the previous identity of the team.
- Fixed a rare hang that could occur when the ticker referenced a previous game result where all of the players on the team have since been traded.
- Selecting to load a custom draft class will no longer cause a game hang in the MyGM offseason if you had previously been fired during that same offseason.

NOTE: All patch fixes will work in your existing game mode saves.

Patch Notes (PC)

GENERAL

- Support added for custom keyboard configurations.

- Optimization improvements that increase performance across all configurations.
- Addressed an issue where some users would notice commentary cutting out during gameplay.
- Resolved frame rate stuttering at start of games/between quarters on moderate to high-end systems.
- Addressed an issue where some users would see a black screen in the Edit Arena and Edit Uniform menus when 'Special Effects' was turned off.
- New Video Settings option 'Media People Detail Level' added; reducing this will help performance on low-end and mid-range graphics cards.
- Corrected missing/incorrect player icons when using a non-Xbox 360 controller.
- Fixed a rare hang that could occur when attempting to fast-forward 2KTV.
- Fixed a rare driver crash on DirectX 10.0 and DirectX 10.1 graphics cards.
- Help text added to Video Settings menu.
- Additional detail setting now available for Shader Detail in the Video Settings menu.
- Fixed super sampling not working when Letterbox (formerly called Aspect Ratio Correction) was disabled.
- The following Video Settings options should now provide the expected results when modified: Color Correction, Depth of Field, Screen Space Ambient Occlusion, Bloom, and Volumetric Effects.
- Resolved a rare case where the game would hang when entering/exiting the Tattoo editor when customizing the tattoos on your MyPLAYER.
- Addressed an issue on the main menu where the title would freeze for a period of time before proceeding to the selected action (e.g. loading a MyCAREER save).
- Players will no longer slide/skate when pressing the block button during an inbound state. It was funny while it lasted.
- Tuned crowd excitement and intensity on made buzzer beaters.
- Resolved a case where certain players (e.g. Kristaps Porzingis) would have their hairstyle change slightly when zooming in the camera very close in replay.

MyPARK/2K PRO-AM

- Pro-Am Team Play has been added to the game! You can now play as a full team in your custom-built arenas.
- Addressed a number of matchmaking issues that inhibited users from playing in Pro-Am Walk-On games.
- Fixed an issue where your player would sometimes appear with the incorrect height (if you had more than one MyCAREER save file) when entering MyPARK/Pro-Am from within MyCAREER.
- Fixed a rare case where the ball would get stuck in the High Roller courts, thus preventing a game from concluding.
- Improved voice chat support in both MyPARK and 2K Pro-Am.
- Improved the animation of the skateboard in MyPARK. If you don't have this, get it. You can move between courts faster than those that are on-foot!
- 2K Beats will now continue to play after finishing a game in MyPARK.

MyCAREER/MyCOURT

- Fixed an issue where some users were unable to join their friends' MyCOURTs.
- Fixed an issue where the host user was unable to send MyCOURT invites after leaving MyPARK/Pro-Am with a squad.
- Addressed a number of issues with Accolades not unlocking at the appropriate time within MyCAREER.
- Pressing the back button will no longer cause the game to hang from the MyCAREER NEXT menu after being eliminated from the playoffs.

MyGM/MyLEAGUE

- Numerous tuning and code enhancements to improve simulated player stats.
- Playoff games (within a series) can no longer be played out of order in MyLEAGUE Online.
- Found and fixed a case where visiting certain submenus with no head coach would cause the game to hang.
- Fans of relocated/rebranded teams will no longer hold up signs referencing the previous identity of the team.
- Fixed a rare hang that could occur when the ticker referenced a previous game result where all of the players on the team have since been traded.
- Selecting to load a custom draft class will no longer cause a game hang in the MyGM offseason if you had previously been fired during that same offseason.

NOTE: All patch fixes will work in your existing game mode saves.

NBA 2K16 PC Hotfixes

A couple of hotfixes have been pushed through for the PC version of NBA 2K16, most recently on October 9th. Patch notes are as follows:

- Added support for 21:9 UltraWide monitors, multiple monitor displays, and other non-16:9 aspect ratios.
- Fixed an issue where frame rate could stutter (even on higher end graphics cards) in either or both of gameplay and MyPARK.
- Fixed a bug where playcalling icons were not appearing for some users.
- Resolved a driver crash on below min spec DirectX 10.0 cards, such as the NVidia 9800 GT.
- Optimized crowd rendering, resulting in slightly improved performance on many configurations.
- Fixed 'black rectangle' artifact.
- Removed super sample configuration option due to driver instability.

Links

- [NBA 2K16](#)

Retrieved from "https://www.nba-live.com/nbalivewiki/index.php?title=NBA_2K16_Official_Patches&oldid=6960"



This page was last edited on 10 October 2016, at 09:15.

EXHIBIT 14



GTAV Title Update 1.28 Notes (PS4/Xbox One/PS3/Xbox 360/PC)

Created July 7, 2015 • Updated October 19, 2017 • [Print](#) • [Share](#) ↗

PATCH NOTES: ILL-GOTTEN GAINS PART 2

New Content – All Platforms

- Five new vehicles have been added to Legendary Motorsport for Story Mode and GTA Online: Coil Brawler, Vapid Chino, Invetero Coquette BlackFin, Progen T20, Dinka Vindicator
- One new vehicle has been added to Docktease for Story Mode and GTA Online: Lampadati Toro
- All new vehicles are available to purchase immediately in Story Mode on Xbox One, PS4 and PC and in GTA Online for all platforms, and are deposited straight into the player's relevant vehicle storage properties in Story Mode for Xbox 360 and PS3.
- Three new horns have been added to Los Santos Customs for Story Mode and GTA Online: Classical Horn 8, Classical Horn Loop 1, and Classical Horn Loop 2
- Two new weapons have been added to Ammu-Nation for Story Mode and GTA Online: Knuckle Duster and Marksman Pistol. Both weapons unlock in Story Mode as the player progresses through Story Mode on Xbox One, PS4 and PC. They are available immediately in Story Mode for Xbox 360, PS3, and are available to purchase in GTA Online for all platforms.
- 15 new tattoos have been added to the Tattoo Parlors for both male and female characters in GTA Online.
- Hundreds of new clothing items, including new luxury outfits and accessories, have been added to the Clothes Stores for male and female characters in GTA Online.

- Eight casual glasses have been added for females that were previously only available for males in GTA Online.
- The Lab radio station from PC has been added to all consoles for Story Mode and GTA Online.

New Content – PS4, Xbox One, and PC Only

- Nine unique Knuckle Duster variants are available from Ammu-Nation for Story Mode and GTA Online: The Pimp, The Ballas, The Hustler, The Rock, The Hater, The Lover, The Player, The King, and The Vagos

New Features/Updates – All Platforms

- CREATOR: Players can now create Jobs inside Los Santos International Airport and Fort Zancudo.

New Features/Updates – PS4, Xbox One, and PC Only

- Scarves and Cuffs have been added to the Accessories section in the Interaction Menu.

New Features/Updates – Xbox 360 and PS3 Only

- Mors Insurance delivery vehicle icons now flash a few times on the minimap.

New Features/Updates – PC Only

- PC ONLY: Help text has been added for players with a strict NAT type that warns them of connectivity issues they may experience and directs them to the support page <http://rsg.ms/nattytype> for further information about improving connection.

Generic/ Miscellaneous Fixes – All Platforms

- Fixed an issue in Story Mode where the player could become stuck when attempting to skip the intro cutscene for Father / Son.
- Fixed an issue where players placing a Bounty could get stuck on the call if the Bounty was placed on a player who just joined a Job.

- Fixed an issue where players were spawning in a circle close together when returning to Free Roam from a Playlist / Job.
- Improved the load times of the in-game Leaderboards screens.
- Fixed an issue where the player is unable to serve successfully after faulting a previous swing during a Tennis Match.
- Fixed an issue where players were unable to enter their personal vehicles in the garage after accepting an invite from a friend via On Call - Friends in Session.
- Fixed an issue where too many players can be placed on one team in an Adversary Mode Job that's a part of a Head to Head Playlist.
- Fixed an issue with interiors not loading properly for low end apartments.
- Fixed an issue where a player could become trapped inside another player's apartment after blacking out.
- Fixed an issue with Pegasus vehicles falling through the world.
- Fixed an issue where players could become stuck when joining GTA Online via the Random Job option from boot.
- Fixed an issue where players buying a Convertible (topless) Voltic would receive a Voltic with a roof instead.
- Fixed an issue in Story Mode where unlock feed messages were incorrectly appearing.
- Fixed an issue where a player kicked from a GTA Online session could return to Story Mode with an incorrect error message, instead of being matched to a different session.
- Fixed an issue where remote players could be seen inside the garage during the local player's garage tutorial after entering their apartment for the first time.
- Fixed an issue with the positioning of players on the starting grid of a Sea Race Job.
- Fixed an issue that caused the reflections of vehicles in GTA Online garages to appear as the incorrect color.
- **CREATOR:** Fixed an issue where players could place more than 20 dynamic props in a Race.
- Players can now use SMG class weapons during a drive-by in the same situations that they can use other two-handed weapons.
- Fixed an issue where incorrect zoom levels would be applied to Sniper Rifles being fired from helicopters.
- Fixed an issue where weapon attachments and engravings could be incorrectly added or removed when re-spawning.
- Fixed an issue where On Call players could get stuck during transition if the host quits.
- Fixed an issue where players ranked higher than 12 are getting the message 'Heist unlock at rank 12'.
- Fixed an issue with button prompts becoming incorrect if the player attempted to use the radio or TV after using the planning board to look for a Heist.
- Fixed an issue with several awards and rewards unlocking incorrectly.
- Fixed an issue which caused boat physics to behave unnaturally.
- Fixed an issue which produced incompatible clothing options.
- Other general fixes to improve game and network stability for Story Mode and GTA Online.

Generic/ Miscellaneous Fixes – PS4, Xbox One, and PC Only

- Fixed an issue in Story Mode where the game would go into a low LOD state when attempting to switch to Michael from Trevor before the Heist Setup: Architect's Plans.
- Fixed an issue in Story Mode where the trigger cutscene for Friend Request wouldn't start.
- Fixed an issue where players could become trapped inside the Luxor if it was being reclaimed by Pegasus.
- Fixed some issues with equipping / un-equipping watches and earrings while browsing the Personal Interaction Menu.
- Fixed an issue in Story Mode where the Submersible was not attached to the ship after loading a save nearby for Heist Setup: Minisub.
- Fixed an issue in Story Mode where the player continually dies when starting the mission "Repossession" in a large aircraft such as the Luxor Deluxe.
- Fixed an issue in Story Mode where the player could see the world load in when using a saved game made in a tunnel or an underground metro station.
- Fixed an issue where players were starting Contact Missions in the air when accepting an invite while airborne.
- Fixed an issue where the Next Job Vote Screen speaker icon wasn't animating when a player was speaking.
- Fixed an issue where using certain special characters in the Playlist name would prevent the player from saving the Playlist.
- Fixed an issue where the Social Club Leaderboard for Darts was showing incorrect / impossible scores.
- Fixed an issue where vehicle spawning near Pershing Square appeared aggressive when entering the session after creating a new character.
- Fixed an issue where several players could see under the world in the vehicle selection and betting screens during a Playlist.
- Fixed an issue where the Smuggler plane circled the area over Vinewood Racetrack indefinitely.
- Fixed an issue where only one player on the cargo team was warped into the Casco after shooting open the lock.
- Fixed an issue where zooming with a Sniper Rifle conflicted with using the phone while in an aircraft.
- Fixed an issue where the Hydra was not displaying for spectators in a Race Lobby.
- Fixed an issue where the bong and wine bottle could be seen floating in mid-air.
- Fixed an issue which caused newly purchased vehicles to disappear from garages.
- Fixed an issue that could have caused weapon attachments to become unequipped when dying in certain situations.
- Fixed an issue with player names not appearing in the correct location above another player's character in GTA Online.
- Fixed an issue with the apartment radio not functioning correctly.
- Fixed an issue where earrings were not transferring correctly between Xbox 360 / PS3 and Xbox One / PS4 / PC.

Generic/ Miscellaneous Fixes – PC Only

- Fixed an issue where spectators could not view Leaderboards when using a keyboard and mouse.
- Fixed an issue which caused Insurgents to be incorrectly priced.
- Fixed an issue in Story Mode where The Epsilon mission 'Chasing the Truth' does not progress after the opening cutscene on PC.
- Fixed an issue where the free haircut coupon from Herr Kutz was not reflected in game.
- Fixed an issue where menus in the Barber / Hairdresser were offset with certain display settings.
- Fixed an issue where players couldn't change the 'Remain Host after Next Job Vote Screen' option.
- Fixed an issue where the Self Radio station was not present in apartment radios.
- Fixed an issue where the free Chrome Rims that were unlocked on a previous console were no longer free after transferring to PC.
- Matchmaking for GTA Online sessions will now consider the "Population Density" setting, increasing car and pedestrian traffic for people on high-end systems and improving performance for people on low-end systems.
- Fixed an issue with Snapmatic photos created with the Meme Editor saving at an incorrect aspect ratio.
- Added a new advanced command line option for people having issues with mouse smoothing or input lag. Using the command "-FrameQueueLimit 0" will improve mouse responsiveness at low frame rates, but may reduce the maximum frame rate.
- Fixed an issue where manually loading a Story Mode save could incorrectly send the player to GTA Online.
- Fixed an issue that caused mouse sensitivity for vehicle controls to be considerably higher than what was set in the "Mouse Driving Sensitivity" option in the Pause Menu.
- Fixed an issue where players were unable to use text chat whilst on the Heist Planning Board.
- Fixed an issue that caused graphical corruption when using the "Pause Game On Focus Loss" setting and leaving the game unfocused for long periods of time.
- Fixed an issue where the "Restore Defaults" option in the Advanced Graphics menu would not reset the Frame Scaling Mode.
- Fixed an issue that prevented the game from launching on Steam if the player had a display name greater than 32 characters.
- Fixed an issue with the pitch control of speedboats when using a mouse and keyboard.
- Fixed an issue where some players were unable to sell ambient cars.
- Added the option to open the Social Club overlay via the Pause Menu.

Rockstar Editor Fixes – PC Only

- Fixed an issue where the camera could escape out of the map in the Rockstar Editor.
- Added an option to the Pause Menu which allows Action Replay clips to auto-save when the player dies.
- Fixed an issue with the Rockstar Editor that could cause graphical issues when playing back a video at ultra-high resolutions.

- Fixed an issue with the Rockstar Editor that caused graphical issues with the Video Gallery thumbnails when running at ultra-high resolutions on a multi-GPU system.
- Fixed an issue where the player was unable to place a clip at the end of the timeline in the Rockstar Editor.
- Fixed an issue with the Rockstar Editor that caused fade in transitions from black clips to render incorrectly.
- Fixed an issue where a replay clip had a small chance of freezing when fine scrubbing backwards in the Rockstar Editor.
- Fixed an issue with corrupted replay clips when moving the cursor between different tracks in the Rockstar Editor.
- Fixed an issue with the position of smashed glass on vehicles being incorrect in recorded Rockstar Editor clips.
- Fixed an issue where Crew emblems on vehicle may not render correctly in recorded Rockstar Editor clips.
- Fixed an issue with environment lighting being incorrect if the player is driving through tunnels in recorded Rockstar Editor clips.
- Fixed an issue with the Rockstar Editor where favorited clips would not be saved correctly when switching between sort types.
- Fixed an issue with the Rockstar Editor that occurred when deleting a clip used in a project, then creating a new clip with the same name.
- Fixed an issue with fine scrubbing after skipping clips in the Rockstar Editor.
- Fixed an issue with Rockstar Editor clips showing incorrect visual effects during events in GTA Online.
- Fixed an issue with a low detail shot of the map being visible for a frame at the end of a Rockstar Editor clip.
- Fixed an issue with the score bar turning purple after user deleted and re-added a clip on the Rockstar Editor timeline.
- Fixed an issue where the playhead in the Rockstar Editor would sometimes become stuck on the text bar on a project timeline.
- Fixed an issue with the music track playhead in the Rockstar Editor wrapping to the beginning of the project.
- Fixed an issue where clip screenshots in the Rockstar Editor would sometimes disappear from the timeline overview.
- Fixed an issue with the Rockstar Editor where it was not possible to place text or audio tracks at the beginning of a project when there were already multiple text or audio items on the timeline.
- Fixed an issue where the Sniper Rifle reticle would show on top of a black filter effect in the Rockstar Editor.
- Fixed an issue that could cause the game to hang when exporting video from a Rockstar Editor project.

Generic/ Miscellaneous Fixes – PlayStation Only

- **PS4 ONLY:** Fixed an issue where players could become stuck on the skycam indefinitely when trying to enter GTA Online via a Live Tile without PS Plus.

Generic/ Miscellaneous Fixes – Xbox Only

- **360 ONLY:** Fixed an issue where players could become stuck after attempting to join another player in a Job Lobby from the Xbox friends menu.
- **360 ONLY:** Fixed an issue where the game crashed when attempting to access GTA Online after performing a sign-in change while in a high-end apartment.
- **Xbox One ONLY:** Fixed an issue where the character's appearance temporarily changes after being kicked from constrained mode due to inactivity.
- **360 ONLY:** Fixed an issue where players would become stuck if they signed out and loaded a save at a certain point in Story Mode flow.

Was this article helpful?
202 of 342 thought so.

YES

NO



ENGLISH

CORPORATE

PRIVACY

LEGAL



EXHIBIT 15

Find Answers



GTAV Title Update 1.27 Notes (PS4/Xbox One/PS3/Xbox 360/PC)

Created June 10, 2015 • Updated October 19, 2017 • [Print](#) • [Share](#) ➔

New Content – All Platforms

- Four new vehicles have been added to Legendary Motorsports for Story Mode and GTA Online: Pegassi Osiris, Albany Virgo, Benefactor Stirling GT, and Enus Windsor.
- Two new vehicles have been added to Elitás Travel for Story Mode and GTA Online: the Buckingham Luxor Deluxe and the Buckingham Swift Deluxe.
- All new vehicles are available to purchase immediately in Story Mode on Xbox One, PS4 and PC and in GTA Online for all platforms and are deposited straight into the players' relevant vehicle storage properties in Story Mode for Xbox 360 and PS3.
- The Combat PDW has been added to Ammu-Nation for Story Mode and GTA Online.
- The Combat PDW unlocks in Story Mode as the player progresses through the flow on Xbox One, PS4 and PC, is available immediately in Story Mode for Xbox 360, PS3 and is available to purchase in GTA Online for all platforms.
- 15 new tattoos have been added to the Tattoo Parlors for both Male and Female characters in GTA Online.
- Hundreds of new clothing items, including new designer outfits and accessories have been added to the Clothes Stores for Male and Female characters in GTA Online.
- Alternative livery designs for the Enus Windsor are available in the Respray section in Los Santos Customs for Story Mode and GTA Online.

New Content – PS4, Xbox One, and PC Only

- Luxurious interior activities have been added in GTA Online for passengers in the Luxor Deluxe and Swift Deluxe and are available in first person. Activities available are:
 - Drinking Champagne (Both)
 - Smoking Cigars (Luxor Deluxe)
 - Browsing Internet (Swift Deluxe)
- Luxury Engravings have been added for a selection of weapons & attachments for Story Mode and GTA Online.
 - Yusuf Amir Luxury Finish is available for the following weapons: Assault Rifle, Carbine Rifle, Pistol, SMG, Marksman Rifle, and Micro SMG.
 - Gilded Gun Metal Finish is available for the following weapons: Advanced Rifle, AP Pistol, Sawed Off Shotgun.
 - Etched Wood Grip Finish is available for the Heavy Pistol and Sniper Rifle.
 - Platinum Pearl Deluxe Finish is available for the Pistol .50.

New Features/Updates - All Platforms

- All of the vehicle websites have been re-designed on all platforms. The sales page for each vehicle has also been updated for all vehicles excluding bicycles on Xbox One, PS4 and PC.
- A Friends in Session option has been added to the On Call phone menu in GTA Online. This allows a player to auto invite all friends in session to the Job they are On Call for. Accepting these invites will place their friends On Call too.
- Better feedback is now given to hosts and players in a lobby as to what is happening with auto-invites.
- The Clothes and Tattoo shop menus have been re-organized. There are now more menu options and the newest items will appear on the top in each category.
- The Karin Technical now comes with bulletproof tires when ordered from Pegasus.
- Flashing help text is now displayed for non-host players on the Job settings screen if the Criminal Mastermind challenge would reset because of the difficulty setting the host has chosen.
- The Crew Challenge playlists have been removed.
- Additional methods have been added to complete the "Bribe the Cops and cause havoc" Daily Objective.
- Vehicles with topless variants now have the option to add or remove the vehicle roof in Los Santos Customs. Sale prices of these vehicles have been rebalanced in line with this change.
- When calling in an Airstrike, a jet range indicator now appears to show how far away the jet is. Once the jet is nearby, the indicator will turn red.
- Fixed an issue where award T-shirts were not unlocking properly.
- Some weapon rebalancing tweaks have been made based on community feedback. The MG and the Combat MG have increased damage, and the Assault Shotgun is now less effective at longer ranges.
- The hook of the Cargobob can no longer lift the vehicles of players in passive mode.
- Added the ability to cycle between targets when using lock-on missiles (such as Buzzard missiles or the Homing Launcher).

New Features/Updates – PS4, Xbox One, and PC

- First person Vehicle Hood Cam (already in PC) has been added for Xbox One and PS4.
- More accessories have been added to the Interaction Menu in GTA Online. Players can now change their Watches, Chains and Earrings from the Accessories section in the Interaction Menu.
- Vanilla Unicorn dancers are now topless for private dances on Xbox One and PS4 in GTA Online.
- Improvements to controller rumble strength.
- The number of dancers at Vanilla Unicorn in GTA Online has increased to match Story Mode.

New Features/Updates – PC Only

- Team colors are now used in PC text chat during Team Deathmatches.
- The control icon that displays whether a player is using keyboard and mouse or a game pad displayed in the Job settings screen now updates live to indicate what input method the player is using at that time.
- A new sort / filter system has been introduced for Clip Management in the Rockstar Editor.
- The radio tracks available in the Rockstar Editor have been adjusted. If any clips you've created have been affected, the game will alert you when loading the clip.

Generic / Miscellaneous Fixes – All Platforms

- Fixed numerous exploits in GTA Online.
- Fixed an issue where the Heist Loyalty challenge would improperly reset when going from a 2 player to a 4 player Job.
- Fixed an issue where invites on the Job List show up as coming from the same person even though the invites are coming from different players.
- Fixed an issue where stunt jumps that had previously been completed weren't being counted towards the "Complete a Stunt Jump" Daily Objective.
- Fixed an issue where players' chains were disappearing from their necks when launching a Heist.
- Fixed several issues where players were seen falling through the apartment when accepting an invite to a Heist or during apartment cutscenes on a Heist.
- Fixed an issue where players couldn't access another player's personal vehicles even though their vehicle access setting allowed it.
- Fixed an issue where players see themselves falling through the Velum just before the Prison Break Finale cutscene.
- Fixed an issue where the hacking minigame would stall if the player launches their mobile phone at the same time as interacting with the panel.
- Fixed an issue where the Prison Break Heist didn't fail when blowing up the prison bus.

- Fixed an issue where a player could see another player's body dropping from the table into the drinking position during the final cutscene of the Heist.
- Fixed an issue where players start a Job in a vehicle would sometimes be stuck and unable to move.
- Fixed several issues with character's appearances changing after a migration.
- Fixed an issue where players could lose control of their characters when going out of bounds during a Versus Mission.
- Fixed an issue where players would not be told to lose their wanted level, even though other players were unable to progress (because they had to wait for the local player to lose their wanted level).
- Fixed an issue where players could fall through the world at the army base after committing suicide on a Job.
- Fixed an issue where personal vehicles were spawning on top of each other outside player's apartments.
- Fixed an issue where the player's currently equipped armor was lost when joining a new session or when migrating from another saved profile.
- Fixed an issue where players would get stuck on a white or black screen when finishing The Fleeca Job: Scope Out.
- Fixed an issue where one player was spawning far away from the other player and had no vehicle when launching a One on One Vehicle Deathmatch.
- Fixed an issue where robbing a store while another player is browsing snacks can cause their game to become unresponsive.
- Fixed an issue where the incorrect Marshall Country livery was being delivered.
- Fixed an issue with major traffic build up at intersection which was causing explosions.
- Fixed an issue where spectators were disconnecting at the end of a Job.
- Fixed an issue where characters imported from 360 or PS3 (which hadn't been updated since the Beach Bum Update) to Xbox One or PS4 lost all their purchased garage vehicles.
- Fixed some issues where players were being kicked from a player's apartment when trying to join a Heist Setup Mission.
- Fixed an issue where players could get temporarily stuck in their garage after quitting a lobby while in an apartment.
- Fixed an issue with blips remaining on the Trackify App.
- Fixed an issue with the Snapmatic grid remaining on screen if the camera is used after the circuit board hacking minigame.
- Fixed an issue where the player had limited functionality accessing their Inventory in the Interaction Menu after successfully hacking a terminal outside the vault door.
- Fixed an issue where players who transferred both 360 and PS3 platforms to Xbox One and/or PS4 couldn't transfer again to PC.
- Fixed an issue where the Hooded Jacket could be worn with the hood up without a mask.
- Fixed an issue where players would get stuck when there were server issues while joining a new session and the player had outstanding cross-session invites that needed displaying.
- Fixed an issue where balaclavas on GTA Online characters in Director Mode clip through the character's face.

- The change gender option has been removed from the Character Creator if the player chooses to edit their character's appearance or access the Creator following a migration.
- The police station back door is now locked in GTA Online.
- The rollercoaster no longer appears invisible when viewing it from a distance.
- Fixed an issue that can lead to unpopulated sessions and sessions with just one player.
- Fixed various issues which caused body parts to become invisible.
- Fixed an issue which caused t-shirt designs to appear over saved outfits.
- Fixed an issue which caused players to lose played Jobs from the Recently Played Jobs list.
- Fixed an issue which allowed players on the Prisoner team to buy weapons during the Prison Break Heist.
- Fixed various issues with incomplete / incompatible outfits.
- Fixed an issue which caused the aircraft carrier to incorrectly appear in GTA Online.
- Fixed issues which caused certain horns to be incorrectly priced in Los Santos Customs.
- Fixed an issue with Depth of Field in Snapmatic.
- Fixed an issue which prevented players from selling the Mesa.
- Fixed an issue which caused Parachute Bags to become Duffel Bags.
- Fixed an issue with the behavior of RPGs in water.
- Players in the Creator will no longer receive a wanted level when testing their Jobs.
- Fixed an issue which caused players to drop parachutes in the Saved Outfits menu.
- Fixed an issue which caused loss of player control when entering the garage.
- Fixed an issue which prevented players from changing license plates on some vehicles.
- Fixed an issue which caused Heist pay-out cuts to display a negative value.
- Fixed an issue which caused placed weapons to be replaced with pistols in the LTS creator.
- Fixed an issue which caused parachute smoke trails to reset.
- Fixed an issue with the Mesa's roof options.
- Fixed an issue which caused the Armored Kuruma's doors to be too vulnerable to sniper fire.
- Fixed an issue which caused players whose teammates disconnected to count as having passed a Heist.
- Fixed an issue which caused the driving and stealth stats to switch while viewing player stats in the lobby.
- Vehicles will now deform appropriately on PS4, Xbox One, and PC when smashed with bats or other melee weapons.
- Fixed an issue with Lifeinvader discounts on Warstock Cache and Carry.
- Fixed an issue which prevented Heist invites being sent following the disconnection of a player.
- Fixed an issue with radar blips in Survivals. Previously, if you died in a Survival and chose to spectate a teammate, their blip on the radar would remain wherever the previous round ended and not properly follow the player.
- Fixed an issue with the logo on the Declasse Voodoo's steering wheel.
- Fixed an issue which caused the trunk to disappear on the Coquette Classic Topless.
- Fixed an issue which caused drunkenness to persist at the start of each round of an LTS.
- Fixed an issue with dynamic props which caused the Creator to return to the main menu.
- Fixed an issue with certain vehicles where players could not destroy the windshields from the inside.

- Game invites will no longer be lost for players that transition from GTA Online to Story Mode while the invite is being sent.
- Fixed an issue where players could not enter the Prison Bus because of bus positioning.
- Fixed an issue where NPCs set on fire by an explosion would not scream or writhe when on the ground.
- Fixed an issue where incorrect audio effects would play when firing suppressed weapons from the side of a helicopter in first person view.
- Fixed an issue where the reload animation of the Flare Gun could playback incorrectly when the player is behind low cover.
- Fixed an issue with explosions triggering incorrectly when shooting a Sticky Bomb held by a friendly player.
- Fixed an issue that caused duplicate Dynasty 8 signposts to appear when owning multiple apartments within Eclipse Towers.
- Fixed an issue where an incorrect error message regarding profile permissions would sometimes appear when joining another player's created Job.
- Fixed an issue with the suspension height on low profile cars being incorrect when respawning during a Race.
- Fixed an issue that caused incorrect member information to display when browsing non-active Crews on the pause menu.
- Fixed an issue where players were incorrectly able to send session invites to party members when in a Solo Session.
- Fixed an issue with lit gas trails from the Jerry Can rendering incorrectly in GTA Online.
- Fixed issue where certain scenes in the GTA Online introduction would not render correctly.
- Fixed several issues that caused roads to appear empty in GTA Online sessions.
- Fixed an issue that caused parked cars to incorrectly spawn during Races when traffic was turned off.
- Fixed an issue where rewards from completing Daily Objectives would not appear in the player's Maze Bank transaction history.
- Fixed a crash caused by invalid names on the GTA Online character selection screen.
- Fixed a crash that had a rare chance of occurring during Golf in GTA Online.
- Fixed an issue that caused some players to be split into different sessions after completing a Heist Setup Mission.
- Fixed an issue during the Pacific Standard Finale that caused the interior of the bank to render incorrectly.
- Fixed an issue that would cause aircraft helmets to sometimes appear when the 'Auto Show Aircraft Helmet' option in the Interaction Menu was set to 'Off'.
- Fixed an issue which caused some players to get stuck on the transition screen after certain Heist Setup Missions.
- Fixed an issue that caused the reticle of the HVY Insurgent's mounted gun to be slightly misaligned in first person view.
- Fixed an issue where the 'Featured Playlist' loading option would sometimes incorrectly send the player to GTA Online instead of the playlist lobby.
- Fixed issues with animations playing incorrectly on female GTA Online characters wearing high heels.

- Fixed an issue where a lobby host was not able to invite others from “From Last Job” and “From Current Session” screens in Job lobbies.
- Fixed an issue with animations when switching weapons and blind firing at the same time whilst behind cover.
- Players can now place Points of Interest without issue in GTA Online and Story Mode.
- Fixed an issue where muting another player wouldn’t carry over between multiple GTA Online sessions.
- Fixed an issue where the player was unable to view the Friends menu correctly in the Pause Menu.
- Fixed an issue where the Invite from Friends List menu would show an incorrect prompt if the player has no friends.
- Fixed an issue where incorrect button prompts were displayed on the Social Club sign in page.
- Fixed an issue where loading screen help text would not display in the correct location.
- Fixed an issue where the depth of field effect would render incorrectly when switching between targets.
- Fixed an issue where players were not able enter the van during the Series A Finale.
- Fixed an issue with player ranks appearing incorrectly when comparing stats in a lobby menu.
- Fixed an issue where the “Expanded Radar” setting would reset to “Off” when changing sessions in GTA Online.

Generic / Miscellaneous Fixes – PC Only

- Fixed button conflicts with the pause menu and the hacking minigames on PC.
- Fixed several issues with memory leaks that may have caused the game to crash after running for long periods of time.
- Fixed a display driver crash on PC.
- Fixed several issues which led to conflicts between text chat entry and player control on PC.
- Fixed an issue on PC which caused a stored Oracle to turn into an Oracle XS.
- Fixed an issue which caused PC players to get the error “Your account has been logged in elsewhere”.
- Black suit pants are now black and no longer have a purple tinge.
- Fixed an issue where the artist and title for tracks in Self Radio did not display correctly on vehicle dashboard displays.
- Fixed an issue where certain combinations of anti-aliasing settings on CrossFire systems could cause graphical corruption.
- Fixed an issue with hairstyles and beards not rendering correctly when using certain combinations of shader and anti-aliasing settings.
- Fixed multiple issues of camera clipping in vehicles caused by multi-monitor setups.
- Fixed an issue where mouse input could not be used to select options on the Report Player screen.
- Fixed an issue where the player was unable to move the cursor with the keyboard or mouse when typing a Job description in Creator Mode.

- Fixed an issue that caused the loading spinner to disappear when the game is minimized during the initial loading screen.
- Fixed an issue that prevented the mobile phone from being displayed if the player is signed out of Social Club.
- Fixed an issue with the activation screen that could cause a hang during initial loading.
- Fixed an issue that caused excessive blurring when on GTA Online transition screens.
- Fixed an issue where the Hacking minigame would be more difficult when running at lower frame rates.
- Fixed issues with mouse input when running at lower frame rates.
- Fixed an issue with loading the game when the My Documents folder has been moved to a non-default location such as a network drive.
- Added a new command line option for advanced users to specify the number of cores used by GTAV on their system by using the command line argument “-maxthreads #”.
- Fixed a crash when updating a previously published user created Job.
- Fixed a crash that could occur on the “Change Character Appearance?” screen when entering GTA Online.
- Fixed an issue that caused certain PC configurations to display black graphical corruption.
- Fixed a crash that occasionally occurred during character switches with some advanced graphics settings set to maximum.
- Fixed issues with UI positioning when using 5:4 aspect ratio.
- Fixed an issue where the player was unable to close the game using Alt+F4 while disconnecting from a Job lobby.
- Fixed an issue caused by switching windows when the game’s resolution was set higher than the desktop resolution.
- Fixed an issue where player would sometimes lose control when sending a text message in GTA Online.
- GTAV will now warn the player if the settings file could not be saved.
- Fixed an issue where monitor refresh rate in Windows Borderless mode would not match actual monitor refresh rate.
- Putting guidelines in Golf will now render properly when using 3D monitors.
- Fixed an issue with VSync locking the framerate to a low value when switching to higher than 1920x1080 resolutions due to incorrect monitor refresh rate value.
- Fixed an issue of graphical corruption during the Tutorial Race of GTA Online with certain SLI setups.
- Fixed an issue where a player could get stuck on the “Invite Friends” screen of Social Club when starting a new Job.
- Fixed an issue that prevented the mobile phone from being displayed when accepting a GTA Online invitation whilst editing Snapmatic text.
- Fixed an issue with the Rockstar Editor caused by picking up and trimming audio tracks simultaneously.
- Fixed an issue where weapons being dropped upon death would render incorrectly in recorded Rockstar Editor clips.
- Fixed a crash that occurred on certain CrossFire configurations when exporting long Rockstar Editor videos at high frame rates.

- Fixed an issue where dirt levels on vehicles were not being recorded correctly in Rockstar Editor clips.
- Fixed an issue with water rendering incorrectly during recorded Rockstar Editor clips.
- Fixed an issue with vehicle doors not recording positions correctly in Rockstar Editor clips.
- Fixed an issue where vehicle dials would appear incorrect in recorded Rockstar Editor clips.
- Fixed an issue where vehicle Crew emblems did not appear in recorded Rockstar Editor clips.
- Fixed a crash that could occur when recording a GTA Online clip featuring grenades or other throwable projectiles.
- Fixed an issue which caused a hang when launching the Rockstar Editor after switching Social Club accounts.
- Fixed an issue that would occasionally cause explosion effects to not playback in the Rockstar Editor.
- Fixed an issue that would occasionally hinder full video and audio playback in the Rockstar Editor Video Gallery.
- Fixed several issues where story dialogue was missing or overlapping during playback in the Rockstar Editor.
- Fixed an issue where score tracks used in Rockstar Editor projects would end abruptly instead of fading out.
- Fixed issues where the score would drop out or skip during clip transitions.
- The free camera will now follow a target that is freefalling or parachuting.
- Fixed issues with camera collision in the Rockstar Editor.
- Fixed an issue with the "At Target" microphone type in the Rockstar Editor camera menu.
- Fixed an issue which caused free camera to change position when scrubbing through clips in the Rockstar Editor.
- Fixed an issue with first person camera restrictions in recorded Rockstar Editor clips.
- Fixed an issue with keyboard and mouse controls in the Rockstar Editor and Director Mode.
- Fixed an issue where the "Skip To Beat" function would be slightly off-beat after trimming a clip in the Rockstar Editor.
- Fixed an issue with environment streaming when scrubbing through clips.
- Fixed an issue with the "Skip To Beat" function which would restrict the player's ability to skip backwards through a song in the Rockstar Editor.
- Fixed an issue that would sometimes cause the game to crash when playing back an exported video in the Rockstar Editor Video Gallery.
- Fixed an issue that occasionally caused the game to crash when saving Rockstar Editor clips during a GTA Online Race Job.
- Fixed several issues caused when signing out of Social Club while loading a clip in the Rockstar Editor.
- Fixed an issue which caused the game to crash during the export of very long videos (multiple hours) in the Rockstar Editor.
- Fixed issues that could occur when loading into select locations in Director Mode.
- Fixed an issue where GTA Online Crew t-shirt colors would not show correctly in recorded Rockstar Editor clips.
- Fixed an issue with saved variations in the "Recently Used" character menu in Director Mode.

- Fixed an issue with the dialogue available for the “Emergency Services – Paramedic” model in Director Mode.
- Fixed an issue where television shows would play at an incorrect speed when Rockstar Editor videos are exported at a high frame rate.
- Fixed issues with flickering effects that could be seen on some CrossFire systems.
- Fixed an issue where GTA Online in game text chat would sometimes be disabled after using the Rockstar Editor.
- Fixed an issue where Rockstar Editor clips did not pause when opening the Social Club overlay menu.
- Fixed issues that occurred when trimming text clips in the Rockstar Editor.
- Fixed an issue with the clouds changing on clip transitions during Rockstar Editor playback.
- Fixed an issue with particle effects not appearing or disappearing as expected when scrubbing through clips in the Rockstar Editor.
- Fixed an issue with blood occasionally not appearing on player model's face during Rockstar Editor playback.
- Fixed an issue with video playback freezing when uploading to YouTube.

Was this article helpful?
474 of 729 thought so.

YES

NO



ENGLISH

CORPORATE

PRIVACY

LEGAL



EXHIBIT 16

Find Answers 



GTA V Title Update 1.36 Notes (PS4/Xbox One/PC)

Created October 4, 2016 • Updated May 23, 2017 • **Print** • **Share** ➔

[11/29/16] New Content – PS4, Xbox One & PC

- Stunt Props can now be used in the Creator for the following job types:
 - Capture
 - Deathmatch
 - LTS

[11/22/16] New Content – PS4, Xbox One & PC

- A new Adversary Mode for 2-16 players has been added to GTA Online which unlocks at Rank 1:
 - Kill Quota
 - Players are equipped with one weapon per round, once they hit their kill quota on that weapon they proceed to the next one. The first team to hit all their quotas, or with the highest count at the end of the timer, wins.
- A new vehicle has been added to GTA Online, available from the Southern San Andreas Super Autos website:
 - Bravado Youga Classic

[11/15/16] New Content – PS4, Xbox One & PC

- A new vehicle has been added to GTA Online, available from the Southern San Andreas Super Autos website:
 - Pegassi Esskey

[11/8/16] New Content – PS4, Xbox One & PC

- A new Adversary Mode has been added to GTA Online which unlocks at Rank 1:
 - **Deadline**
 - Players try to take each other out using trails coming out of the back of their bikes.
 - This mode is for 2-4 players.
- A new vehicle has been added to GTA Online, available from the Legendary Motorsport website:
 - Nagasaki Shotaro (available after completing 1 round of Deadline).
- New Deadline Outfits have been added to GTA Online for Male and Female characters:
 - These Outfits will become available after completing 1 round of Deadline.

[11/1/16] New Content – PS4, Xbox One & PC

- A new vehicle has been added to GTA Online, available from the Southern San Andreas Super Autos website:
 - Declasse Tornado Rat Rod

[10/28/16] New Content - PS4, Xbox One & PC

- One new vehicle has been added to GTA Online, available from the Southern San Andreas Super Autos website:
 - LCC Sanctus
- A new Adversary Mode has been added to GTA Online which unlocks at Rank 1:
 - **Lost Vs Damned**
 - Day and night Deathmatch between the forces of good and evil
 - This mode is for 2-10 players

[10/25/16] New Content - PS4, Xbox One & PC

- One new vehicle has been added to GTA Online, available from the Southern San Andreas Super Autos website:
 - Pegassi Vortex

[10/11/16] New Content – PS4, Xbox One & PC

- Two new vehicles have been added to GTA Online:
 - BF Raptor
 - Daemon Custom
- A sixth purchasable property slot has been added to GTA Online

New Content – PS4, Xbox One & PC Only

- Clubhouse Properties have been added to GTA Online. Players can buy a Clubhouse via the new foreclosures.maze-bank.com website after completing or skipping the GTA Online tutorial. Upon purchasing a Clubhouse, players will be given the title of Motorcycle Club (MC) President which allows them to have up to 7 players in their MC and gain exclusive access to new Club Work, Club Challenges, Member Challenges, Clubhouse Contracts, Businesses and more. Players can also customize their Clubhouse with additions such as Murals, Styles, Club Emblems, a Gun Locker and a Custom Bike Shop.
- Business properties have been added to GTA Online. Players can purchase Businesses via the new Open Road website on the laptop in their Clubhouse. Businesses include: Weed, Forgeries, Counterfeit Cash, Meth and Cocaine. Businesses can be upgraded with a choice of Security, Staff or Equipment upgrades to increase their production rate and reduce the risk of potential attacks from police raids and enemies. Players will have to set up, resupply and defend their Businesses to capitalize on their maximum potential profit.
- MCs also have access to exclusive features and benefits such as:
 - MC Presidents and their Road Captain can use the “Riding Formation” feature in the Interaction Menu. This creates a radius around the player who has set the formation which members can enter on their motorcycles and receive accelerated player and vehicle health regeneration.
 - The MC President can choose between two different riding styles, normal and relaxed, in the Manage MC section of the Interaction Menu, which applies to all gang members.

- MC Presidents can assign each member of their MC a role; players can be made either a Prospect, an Enforcer, a Sergeant at Arms, a Road Captain or a Vice President. Each role possesses its own abilities that can help both you and your team.
 - Presidents can choose from a selection of Biker outfit styles for their whole MC.
 - MC members can choose their own outfit from the style the President sets.
 - MCs have access to newly added Club Work, Club Challenges and Member Challenges.
- Club Work has been added to GTA Online. These include:
 - **Deathmatch** (2-16 players)
 - Invite Only. Challenge a rival MC or Organization in session to a Deathmatch.
 - **Joust** (2-16 players)
 - Challenge a rival MC in session to a Deathmatch on motorcycles.
 - **Caged In** (2+ players)
 - Open to all. Head to an area on a motorcycle and follow the route to the end to win. Use Biker melee to take out the other bikers on route.
 - **Stand Your Ground** (2+ players)
 - Open to all. Head to a nearby area and defend it from other players in session. Rivals win by capturing the area.
- Club Challenges have been added to GTA Online. These include:
 - **Search and Destroy** (2-8 players)
 - Compete against your MC in a Race to search an area of the map and hunt down a target.
 - **Wheelie Rider** (2-8 Players)
 - Players in the MC compete to achieve the longest wheelie in the time limit.
 - **Criminal Mischief** (2-8 Players)
 - Players compete to melee the most cars whilst riding their motorcycles.
- Member Challenges have been added to GTA Online. These include:
 - **Rippin' It Up** (2-8 players) (Enforcer ability)
 - MC Members have to cause the most damage whilst driving their motorcycles.
 - **Hit and Ride** (2-8 players) (Vice President ability)
 - Compete against your MC to drive-by kill the most enemy targets across the map within the time limit.
 - **On the Run** (2-8 players) (Sergeant at Arms ability)
 - All players in the MC are given a 5 star wanted rating. Last to survive wins.
 - **Race to Point** (2-8 players) (Road Captain ability)
 - Compete against your MC in a motorcycle Race to the Clubhouse or a random destination on the map.
- Clubhouse Contracts have been added to GTA Online. These include:
 - **Gunrunning**
 - Ambush a weapons deal and deliver the weapon cases to the drop-off.
 - **P.O.W**
 - Rescue a contact from a secure location and get them to safety.
 - **Guns for Hire**
 - Defend a convoy as it travels from one location to another.

- **By The Pound**
 - Complete a deal for some product and deliver it to the buyer.
- **Weapon of Choice**
 - Assassinate targets using a specific method of execution.
- **Nine Tenths of the Law**
 - Steal motorcycles from a secure location and deliver them the drop-off.
- **Cracked**
 - Attack a gang hideout and steal cash from their safe.
- **Jailbreak**
 - Hijack a prison bus full of prisoners and get them to the escape vehicles.
- **Fragile Goods**
 - Destroy Lost MC vans driving around Los Santos and Blaine County.
- **Outrider**
 - Collect contacts one by one on a motorcycle and transport them to safety.
- **Torched**
 - Go to gang members' warehouses and burn their munition crates.
- A new Adversary Mode has been added to GTA Online which unlocks at Rank 1:
 - **Slipstream**
 - A Race where teams of riders have to stick together to hit checkpoints, slipstreaming each other to get speed boosts. The first team across the finish line wins. This Mode is for 4-16 players.
- 13 new vehicles have been added to GTA online:
 - Western Nightblade
 - Shitzu Defiler
 - LCC Avarus
 - Western Zombie Bobber
 - Western Zombie Chopper
 - Shitzu Hakuchou Drag
 - Nagasaki Chimera
 - Western Rat Bike
 - Nagasaki Street Blazer
 - Maibatsu Manchez
 - Pegassi Faggio Mod
 - Pegassi Faggio Sport
 - Western Wolfsbane
- One existing vehicle has been added for purchase:
 - Western Bagger
- Seven new weapons have been added to GTA Online:
 - Pool Cue
 - Pipe Wrench
 - Battle Axe
 - Compact Grenade Launcher
 - Sweeper Shotgun

- Mini SMG
- Pipe Bomb
- Over 50 tattoos have been added to GTA Online.
- Over 275 male and female clothing items have been added to GTA Online.

New Features / Updates – PS4, Xbox One and PC Only

- Players can now melee attack while seated on a motorcycle. This can be performed while having a one-handed weapon equipped.
- New Crew Emblem pieces have been added to the Social Club Emblem Creator.
- The “Non-Contact” option can now be used in Stunt Races.
- Players are now able to copy the Rank of their existing character into the second character slot when creating a new character. The Rank has been capped at 120.
- Icons have been added next to players’ platform names on the D-Pad down players list to indicate whether a player is a CEO, VIP or an MC President.
- The Rank requirement for applying a Crew Emblem to your clothing has been removed.
- New Adversary Series triggers have been added to GTA Online Freemode.
- Changes have been made to the Freemode triggers that were added with Cunning Stunts.
- Stunt Series triggers have been reduced to 8 locations.
- Adversary Mode triggers have been reduced to 3 locations.
- Destroyed and impounded vehicles are now available in Races.
- The “Fast Zoom” option is now available in all Creators.
- The wait timer in Stunt Series lobbies has been changed to 10 seconds when the lobby is full.
- Players are now able to sell free vehicles. Players will not receive any money for the vehicle itself but will be given 50% of the value of any mods applied.
- New driver idle and fidget animations have been added for motorcycles and will play when the vehicle is stationary.

Rockstar Creator Fixes – PS4, Xbox One and PC Only

- Fixed an issue that resulted in the starting grid for any created Stunt Race needing to be placed on the ground as opposed to being able to be placed on a raised platform.
- Fixed an issue that resulted in the option “Lobby Radio” no longer being present in the Creator.
- Fixed an issue that caused different colored props to flicker when placed inside each other.
- Fixed an issue that allowed 150 fireworks to be placed near each other which impacted players’ frame rates.
- Fixed an issue that resulted in the “drop” functionality in the Creator not working as intended.

- Fixed a crash that had a chance to occur when testing a multi-route Race in the Stunt Race Creator.
- Fixed an issue that resulted in players getting stuck in the sky after exiting a Heist Mission, entering the Creator, creating a Mission then leaving the Creator.
- Fixed an issue that resulted in players getting their camera stuck after spamming the use of triggered snapping and zooming in.
- Fixed an issue that resulted in players being warped to the city upon attempting to place a Stunt tube prop that was intersecting with the ground.
- Fixed an issue that caused the “Cycle Items” option to not work when Stunt Props from the dynamic menu were selected at the time.
- Fixed syncing issues with the Bowling Pin and Bowling Ball props in Stunt Races.
- Fixed an issue with the Slow Down Prop in Stunt Races that caused vehicles to remain slowed when no longer in contact with the prop.
- Fixed an issue that resulted in players being unable to place any further props with the error “Maximum (15) speed boost tubes placed”.
- Fixed an issue that caused the GTA and Rally Race option to not be present for all eligible Race types (Land, Sea, Air) in the Race Creator.
- Fixed an issue that resulted in players getting stuck on a black screen after attempting to test a Stunt Race or a Capture in the Creator.
- Fixed an issue that resulted in players being unable to change their created Race type.
- Fixed an issue that caused players to be sent into a loading screen after cycling through the Actor Relationships menu.
- Fixed an issue that caused the Bullpup Shotgun & Hammer to not work as Forced Weapons in Creator.
- Fixed an issue that caused Capture objects to no longer be highlighted red & blue.
- Fixed an issue that caused snapping to stop working correctly if the player rotates the prop with a different rotation type.
- Fixed an issue that caused UI to flicker when snapping some gate props in certain positions.
- Fixed an issue that caused the button prompt to delete an object to remain despite choosing to lock it from deletion.
- Fixed an issue that caused the camera to stop moving if the player placed snapping props down together quickly.
- Fixed an issue that occurred when players placed a prop close to another prop and turned on snapping (with chain snapping). The prop would then connect to the other side of the placed prop.
- Fixed an issue that occurred when players used triggered snapping for two props which would then snap those two props together, instead of snapping to the already placed prop.
- Fixed an issue that resulted in the chain snapping camera moving in the opposite direction to where the prop is being placed.
- Fixed an issue that caused template color / tint to change after testing a Race.
- Fixed an issue that caused the “Use Stunt Camera Trigger” option in Creator to have no effect on published Jobs.
- Fixed an issue that caused props to snap to the wrong angle.
- Fixed frame rate issues that occurred when using wall ride props.

- Fixed an issue that resulted in Stunt Props rotating erratically after turning Prop Snapping on and off.
- Fixed an issue that resulted in players' cameras becoming stuck on props when using the map warp.
- Fixed an issue that meant players were unable to start the alarm preview for the Public Address System.
- Fixed an issue that resulted in the wrong Race categories remaining locked in the Creator.
- Fixed an issue that resulted in the button prompt for "Snap (Hold)" being missing.
- Fixed an issue that meant loading a published creation in Stunt Race Creator resulted in Race Details > "Race Title" to change to "No translation." rather than the expected Race name.
- Fixed an issue that caused various props to have the incorrect color in the Creator.
- Fixed an issue that caused pedestrian vehicles to stop at a certain distance from speed boost props, leading to blocked roads.
- Fixed an issue that resulted in the ticker feed message for the Hatchet being blank.
- Fixed an issue that caused override rotation to reset the X and Y values to 0.
- Fixed an issue with the MTL Dune that resulted in no liveries being applied during Creator tests.

General / Miscellaneous – PS4, Xbox One & PC Only

- Fixes have been implemented to improve game stability, matchmaking and network performance for GTA Online.
- Fixed an issue that resulted in players experiencing frame drops when cycling through items in the accessories section of the Interaction Menu.
- Fixed an issue that resulted in players being unable to trade in the garage at Unit 124 Popular St.
- Fixed an issue that resulted in players being unable to see graphics on their T-shirts after a Heist.
- Fixed an issue that caused some Stunt Race motorcycle behaviors to persist after returning to Freemode.
- Fixed an issue that resulted in players dying too easily upon impact when landing a jump in a created Stunt Race.
- Fixed an issue that caused players' spawn location to change to their new Office despite the player having not bought the accommodation option.
- Fixed an issue that resulted in all players being unable to take off in the Buzzard during the Vehicle Deathmatch – Buzz Kill.
- Fixed an issue that resulted in players being spawned at the start of the Vehicle Deathmatch - Buzz Kill without Buzzards.
- Fixed an issue that resulted in players being placed into an assisted aim session in a free aim state after joining then leaving a free aim session.

- Fixed an issue that caused players to become stuck on a black loading screen after their CEO disconnects from session and they've been chosen to pilot their CEO's helicopter.
- Fixed an issue that resulted in players being unable to start Special Cargo Missions from their Warehouse laptop.
- Fixed an issue that resulted in players from different Organizations that owned different Warehouses to end up in the same Warehouse upon completing a Buy Special Cargo Mission.
- Fixed an issue that resulted in players having no drop-offs after getting into the second Cuban plane during Sell Special Cargo Missions.
- Fixed an issue that caused pedestrians to spawn without props during Buy Special Cargo Missions that involved assassinations.
- Fixed an issue that caused players to not receive a call from their PA saying that a Special Cargo Item is available if the player is active on a Buy Special Cargo Mission.
- Fixed an issue in Buy Special Cargo Missions where players were unable to pick up crates.
- Fixed an issue in Buy Special Cargo Missions where CEOs wouldn't retain their wanted level after dying and respawning.
- Fixed an issue in the Stunt Race – Plummet where players were able to pass through a checkpoint without collecting it in the Annis RE-7B.
- Fixed an issue in the Stunt Race – Plummet where players would be unable to see the next checkpoint if they had respawned just as they had reached the previous one.
- Fixed an issue in the Stunt Race - Plummet where no Race UI would be present when starting a Race with 15 players and a spectator.
- Fixed an issue that caused players to display as "***invalid**" when they replied to other players' in-game text messages.
- Fixed an issue that resulted in players' scores to decrement after being killed during an impromptu Vehicle Deathmatch.
- Fixed an issue that caused the accessories, cash and ammo sections of the Interaction Menu to become inaccessible after buying a large amount of accessories.
- Fixed an issue that caused players to not be offered to widen matchmaking for any Job type if they had previously been On-Call for Stunt Series.
- Fixed an issue that resulted in players having no Quick Job functionality after creating a new Character.
- Fixed an issue that resulted in players losing the Rollercoaster UI after riding it.
- Fixed an issue that resulted in players getting stuck in the sky after accepting an invite to the Stunt Race - City Air while looking for Quick Job > Stunt Series.
- Fixed an issue that meant the Declasse Drift Tampa was selectable in the Stunt Race - City Air.
- Fixed an issue with the Declasse Drift Tampa where the Crew Emblem would temporarily disappear until the player either left session or entered the mod shop.
- Fixed an issue that resulted in Crew Emblems not being applied to the Declasse Drift Tampa.
- Fixed an issue that resulted in players experiencing a prolonged wait time when using Quick Job > Stunt Series.
- Fixed an issue that caused players to get stuck after attempting to join GTA Online if their spawn location is set to Eclipse Towers Apt. 3 Garage.

- Fixed an issue that resulted in Stunt Series lobbies not automatically launching when the countdown timer expires.
- Fixed an issue that caused players to be incorrectly placed into a solo session after entering a Stunt Series trigger.
- Fixed an issue that resulted in players being stuck looking at a Race trigger after being kicked from a lobby that was joined via the Quick Job > On Call option.
- Fixed an issue that caused players' motorcycles to be stuck floating in mid-air.
- Fixed an issue that caused players to be invisible after joining a Stunt Series Race via the On-Call option.
- Fixed an issue that caused queuing to be instantly cancelled when attempting to go On Call to a Stunt Series via the Freemode trigger while in an Invite-Only session.
- Fixed issues with trading in Offices.
- Fixed an issue that resulted in players getting stuck when attempting to enter Los Santos Customs during the Online Tutorial.
- Fixed several map escapes in GTA Online.
- Fixed an issue that resulted in players being unable to get into any Job in GTA Online.
- Fixed an issue in the Heist Humane Labs – Key Codes where players would be shown the alert "You can't access an apartment whilst you have a Mission objective" when trying to deliver the briefcase.
- Fixed an issue that resulted in players being shown 2 crates in their Warehouse after having only delivered 1.
- Fixed an issue that resulted in the Annis RE-7B stock spoiler being listed as "None" in the mod shop instead of "Stock".
- Fixed an issue with the brake calipers of the Annis RE-7B that caused clipping when using custom wheel rim modifications.
- Fixed an issue with the tires of the Annis RE-7B that rendered them with a low-resolution texture when the vehicle was dirty.
- Fixed an issue that resulted in players experiencing constant frame rate drops when going On-Call for Stunt Races.
- Fixed an issue with the Southern San Andreas Super Autos website that resulted in the "Sort by Price" option working incorrectly.
- Fixed an issue with the Southern San Andreas Super Autos website where motorcycles from Cunning Stunts were not showing up in the "Motorcycles" category filter.
- Fixed transaction issues when purchasing a tracker for vehicles.
- Fixed an issue in Team GTA Races where players were unable to enter their teammates' vehicles if their teammate left during a Race.
- Fixed an issue that resulted in players' saved racing outfits' helmets to be overridden by the bike helmet option in the Interaction Menu.
- Fixed an issue that resulted in players' outfits in Stunt Races to be changed to their Freemode outfit on the celebration screen.
- Fixed an issue that caused betting to be suppressed in Jobs with multiple rounds.
- Fixed an issue that caused players in the highest DNF position to receive less RP than players who had finished after them in Premium Races.

- Fixed issue that occurred in the Adversary Mode - Entourage where the target's regeneration circle did not follow the target pad for spectating player.
- Fixed an issue that resulted in players being unable to spectate players in the Adversary Mode - Entourage.
- Fixed an issue that caused SecuroServ options to be missing from the Interaction Menu in the Adversary Mode - Entourage.
- Fixed an issue that resulted in players being unable to join an existing lobby with spaces after they had created their own solo instance.
- Fixed an issue in the Stunt Race - East Coast where the Stunt camera wouldn't initiate when performing the Stunt jump.
- Fixed some issues with the Stunt Race camera being difficult to control when driving through tube props or performing flips.
- Fixed an issue that caused some dashboard dials in the Buckingham Luxor to not display accurate information.
- Fixed an issue with the Obey Omnis that would cause the rev lights to be overly bright making it difficult to tell when they light up when accelerating.
- Fixed an issue that resulted in players having no spoiler on their Obey Omnis despite having it chosen after quickly scrolling through the menu in the mod shop.
- Fixed an issue that prevented the Obey Omnis from being picked up by the Cargobob helicopter.
- Fixed an issue with spoilers on players' custom Annis RE-7B, Emperor ETR1 and Obey Omnis.
- Fixed an issue that meant if a player had gone over the RP cap on their last Stunt, the RP would then be removed from the player during their next Stunt.
- Fixed issues where players would not receive the correct amount of RP bonus when performing Stunts during Stunt Races.
- Fixed an issue that resulted in open face helmets changing to a different type when players removed them.
- Fixed an issue where the accuracy of distant player blips would be incorrect and cause visual issues on the radar.
- Fixed an issue that could cause the radar to render incorrectly during Stunt Races.
- Fixed an issue that resulted in a news feed article for Valentine's Day incorrectly appearing when joining Freemode from a Stunt Race.
- Fixed an issue that resulted in players receiving the message "Your Rank is too low" after going On Call and quitting from a Premium Race lobby.
- Fixed an issue that resulted in players experiencing frame rate drops after calling in the Turreted Limo twice while onboard a Yacht located at LSIA.
- Fixed an issue that caused the flag on the Galaxy Super Yacht to disappear when viewed at certain angles.
- Fixed an issue that resulted in Crew best lap times being overwritten with personal lap times, even when the score doesn't beat the Crew best.
- Fixed issues that resulted in players being unable to use their personal vehicle in Premium Races.
- Fixed an issue in Stunt Races that would cause the radio station to change upon respawning.

- Fixed an issue that resulted in players not spawning in their chosen location when entering GTA Online.
- Fixed an issue that resulted in players being unable to purchase the Crew Emblem mod for the Vapid Contender.
- Fixed an issue that resulted in Crew Emblems not being removed from clothing after changing Crew.
- Fixed an issue that caused Crew Emblem Tattoos to not be cleared from players after they switched Crews.
- Fixed an issue that resulted in large t-shirt Crew Emblems being removed after entering the character selector and returning back to Freemode.
- Fixed an issue that caused Crew colors and Emblems on players' Principe Nemesis motorcycles to not persist into a new session.
- Fixed an issue that caused decals to appear corrupt on Stunt Suits when a player had a Crew Emblem equipped and entered a Stunt Race.
- Fixed an issue that caused the Two Tone Tank Top to display Crew Emblems incorrectly.
- Fixed an issue that caused the Diamond Back Stomach Tattoo to overlap with clothing items and Crew Emblems.
- Fixed an issue that caused Crew Emblems to be removed inconsistently on players' outfits after changing their Crew.
- Fixed an issue that resulted in the 3x RP Icon appearing for Jobs in the pause menu when queuing for an active Premium Race.
- Fixed an issue that caused remote players on motorcycles that were throwing projectile weapons to become out of sync.
- Fixed some collision issues with the environment when riding the Western Company Gargoyle Motorcycle.
- Fixed an issue that resulted in players being spawned under the map after dying on their Western Gargoyle motorcycle.
- Fixed an issue with the wheel axles of the Imponte Dukes being slightly off-center from the chassis.
- Fixed an issue with the default spoiler of the Emperor ETR1 that caused it to detach when crashing the vehicle.
- Fixed an issue that caused the exit / entry animation for Benny's mod shop to not correctly play.
- Fixed an issue that caused players to get stuck in the sky after joining a Friend from the pause menu.
- Fixed an issue that resulted in players becoming invisible after passing out from drinking on the Yacht.
- Fixed an issue that resulted in players being unable to stand up from the sofa after completing Prison Break – Finale.
- Fixed an issue that resulted in players experiencing a constant "Transaction Pending" spinner after transferring a character.
- Fixed an issue that resulted in players receiving no audio for the Benny's Original Motorworks Mod shop Tutorial.

- Fixed an issue on the Stunt Race - Trench where the checkpoint was too close to a jump resulting in players flying over it repeatedly.
- Fixed an issue that caused players to get stuck on a black screen when trying to play Flight School.
- Fixed an issue that resulted in the "Friends in session" ticker feed message flashing between black and red.
- Fixed an issue that resulted in players getting stuck loading into the Learn the Ropes Tutorial Mission after exiting the game, rebooting then re-joining Online.
- Fixed an issue that caused players' gamertags to be misaligned at the top of the Interaction Menu.
- Fixed an issue that resulted in incorrect player names to display on the leaderboard during Playlists containing Stunt Races.
- Fixed an issue that caused players' speed to appear with an RP icon after it in the d-pad down leaderboard during the Freemode Challenge – Highest Speed.
- Fixed an issue that resulted in an overlap on the Stunt Race leaderboard when players left then re-joined the Race.
- Fixed an issue in the Stunt Race – Splits where certain alarms wouldn't play during the second lap.
- Fixed an issue that caused the Bike Helmet option to get changed to "None" after changing the option, adjusting the visor option then closing the Interaction Menu.
- Fixed an issue that caused players' visor option to not retain their up / down choice.
- Fixed clipping issues when wearing certain hats inside the Lampadati Tropos Rallye.
- Fixed an issue with players' helmets spinning when being equipped after changing visor.
- Fixed an issue that caused the Stunt Race flag filter to appear in other lobbies.
- Fixed an issue that resulted in spectators experiencing overlapping UI during vehicle selection screens.
- Fixed an issue that resulted in the Bike Helmet option in the Interaction Menu being randomly set to "None".
- Fixed an issue that caused players' characters to equip a helmet despite all helmet and headgear related options being set to "None".

General / Miscellaneous – PS4 Only

- Fixed an issue that caused players to be stuck after deleting an Online character then quickly signing out of PSN and back in while doing so.
- Fixed an issue that resulted in players getting stuck in the sky after attempting to enter GTA Online via an activity feed message.

General / Miscellaneous – Xbox One Only

- Fixed an issue that resulted in players being unable to leave GTA Online via the switch wheel after swapping sessions during a service outage.
- Fixed an issue that resulted in references to PS4 parental restrictions when trying to apply a Crew Emblem to a bike while having "See and Share Content" set to Blocked.
- Fixed a networking issue that may have caused a timeout error when joining a large GTA Online session.
- Fixed an issue that caused bonus RP to not be earned correctly when singing in the shower using a headset.

General / Miscellaneous – PC Only

- Fixed a crash that occurred after players signed out of Social Club when attempting to make a new profile.
- Fixed an issue that caused Social Club UI to close straight away when being opened by a controller.
- Fixed an issue that reset weapon wheel positions when changing the window mode setting, or using Alt + Tab while in full screen mode.
- Fixed an issue that resulted in The Black JC Helmet (given as part of the Black Jock Cranley Suit reward outfit) to not be available in the Interaction Menu.
- Fixed an issue that resulted in an incorrect button prompt displaying when switching between controller and keyboard at the Office computer.
- Fixed an issue that caused the displayed image on in-game monitor screens to mimic the screen setting on the users' display when running Nvidia Surround / Eyefinity.

Was this article helpful?
516 of 875 thought so.

YES

NO



🌐 ENGLISH ▾

CORPORATE

PRIVACY

LEGAL



EXHIBIT 17



Gaming Reviews, News, Tips and More.



LATEST E3 2019 SPLITSCREEN HIGHLIGHT REEL VIEWPOINTS COSPLAY THE BEST

Quality Assured: What It's Really Like To Test Games For A Living



Jason Schreier

1/18/17 11:30am



There's an old commercial for Westwood College that's become something of a running joke in the video game world. Two young men sit at a couch, hammering away at PlayStation controllers. A woman walks in. "Hey guys, finish testing that game yet?" she asks. "I've got another one I need designed."

This piece originally appeared 7/27/15.

“We just finished level three and need to tighten up the graphics a little bit,” says one of the men. Then he turns to his friend, smiling like he just won the lottery. “Hey, I can’t believe we got jobs doing this.”

“I know,” the other guy says. “And my mom said I would never get *anywhere* with these games.”

For a very long time, that’s how people have imagined the life of a video game tester, not as 9-to-5 job but as the fantasy of teenagers everywhere. Who wouldn’t want to sit on a comfy couch and play games all day, taking the occasional break to tighten up the graphics on level three?

Reality is a little different. Video game quality assurance (QA), as testing is called, is often perceived as “playing games for a living” but might better be described as *breaking* them. It’s a low-paying, occasionally rewarding, often frustrating job that has both more and less to do with the quality of today’s games than you might expect.

A professional QA tester doesn’t just sit by the television, crack a Mountain Dew, and saunter through level 5 of the latest shooter; he or she spends 14 straight hours running into different walls to see if they’re all solid. Proper video-game testing is more akin to abstract puzzle-solving than it is to getting a top score in *Donkey Kong*, despite what you may have seen in college commercials like Westwood’s. “It takes a very specific attitude and outlook to really be good in the QA world,” a veteran game tester told me. “It goes beyond a passion for video games, and definitely beyond the notion that you get to play video games for a living.”

Game testers are, by nature, underappreciated; they are only noticed when something goes wrong. QA veterans say the job is stressful, tedious, and often seen as a doorway to other parts of game development rather than a viable career path. Often, testers work on temporary contracts or for outsourcing companies that don’t let them communicate directly with a game’s developers. And when a game is particularly buggy or broken—as many recent releases have been—it’s customary for observers to blame QA.

They are the safeguards, after all—the final wall between a programmer’s mistakes and a customer’s money. It’s in the name: Quality Assurance. They’re supposed to assure quality.

But when a big game ships broken, is QA really to blame? How could testers possibly not find some of the bugs that show up in the games we play? Why do so many servers break all the time? Just what do QA people do all day, anyway?

Over the past few months I’ve had extensive conversations with several dozen current and former QA testers—many of whom spoke anonymously in order to protect their careers—in an attempt to explore the world of video game testing and try to explain what it’s really like to play games for a living. Some said they hated working in QA; others said they couldn’t imagine doing anything else. Almost all agreed on one thing: not a lot of people understand how QA actually works.



For as long as there have been video games, there have been bugs. Some are relatively harmless and have even attained mythological status, like *Pokémon*’s enigmatic Missingno. Others are seared into video game history, like Minus World, an unbeatable *Super Mario Bros.* level that you can only

access by glitching into a wall. Many bugs have been discovered, abused, and enjoyed by the assiduous speedrunning community—how else would you beat *Ocarina of Time* in 17 minutes?

Those are the friendly bugs, though. Most video game glitches are irritating at best and game-breaking at worst, which is why every game goes through quality assurance, an extensive testing process implemented to ensure that everything's working properly. The term "QA" draws from the world of products—microwaves, cars, assembly lines—and in many ways video game testing is no different. A tester's job is to poke, prod, and play the hell out of a game until all the kinks are gone, like a factory-worker smoothing out the latest toy.

There's no game industry standard when it comes to QA procedure—every game is different and every company has its own process—but a tester typically spends months playing builds of the same game over and over again in all sorts of different ways. The more bugs a tester finds, the higher his or her perceived value to the company. This is one hell of a challenge, of course: video games are complicated sets of interlocking systems that require careful, meticulous bug-testing, which can involve testing the same level repeatedly with slight variations—using a new character, wielding a different weapon, following an alternate pathway—and recording everything that happens.

Take *Grand Theft Auto V*, for example. On Rockstar's giant open-world game, QA testers had to divide and conquer. "You would have individual testers assigned certain missions, or tasks, mini games etc.," said one person who helped test the game. "Normally starting with the big stuff and working down. So doing story missions in order, then heists, then side missions and random characters until you moved on to testing the strip club and prostitutes."

Sometimes, that tester said, they'd also have to devote tons of time to granular parts of the game, like when Rockstar's designers asked a group of QA staff to test everything players could do with the game's automated taxi

service. They quickly found that taking a taxi to a new mission would trigger the mission without properly disposing of the cab, leading to some amusing moments as a taxi drove around and tried to back up during cut-scenes.

“I think working on a project like that is made a lot better by the small moments where something truly stupid happens,” the tester said. “Talking pigs randomly standing up like a person and walking away, randomly being shot out of the sky in a plane by an ambient pedestrian whose physics had fired him into space. Trevor pulling his trousers down then never animating to pull them back up and spending the rest of the entire game with his trousers around his ankles. Franklin’s dog used to instantly die if he touched water... he’d just fall into a pool and sink to the bottom like a rock as soon as his paws got wet.”

Finding bugs is just the first step—the second, significantly harder process is trying to reproduce glitches so the company’s engineers can zap them. A tester can’t just write down something like “Trevor’s pants won’t stay on” and send it to the programming team; what could engineers possibly do with that information? In order to track down, isolate, and fix a bug, coders need to know exactly how it happens, which can be a tricky riddle to solve given how many variables are in video games. Good QA testers quickly learn to keep track of every action they take—from big to small—so they can at least try to reproduce any bug they stumble upon. “I love that working QA is

often like being paid to solve puzzles,” said veteran tester Rob Hodgson, who’s done the job for eight years now. “Figuring out how to reproduce that bizarre error you encountered, step by exacting step, is thrilling to the right kind of person.”

A typical day for a QA tester could vary drastically based on the project, the role, and the position. An outsourcer might have to spend 10 hours slamming into every wall in the latest *Call of Duty* to see what breaks (“collision testing”). An embedded QA tester might have to work with a programmer to find out why the framerate keeps dropping on the Android version of their new mobile game. The varied and monotonous nature of QA can lead to some unexpected challenges; for example, one tester who worked on the rhythm game *Rock Band* said the clack-clack-clack of plastic drumming could be so maddening, they had to set a rule: no instruments on Tuesdays and Thursdays.

As they play, testers have to type up bug reports, using software like JIRA to explain what happened and how it happened. Programmers—who ideally at this point are no longer working on new content and are exclusively fixing bugs—analyze the reports and respond as necessary, sometimes with questions, problems, and occasionally snarky comments.

When a console game is almost done, it has to go through certification, a process in which the publisher (like, say, EA) will ask the console manufacturer (Sony, Microsoft, or Nintendo) to check the game for game-crashing bugs. During this certification process, a second layer of QA staff called compliance testers must go through the game with their own fine combs, checking to see if everything lives up to expectations. Every console-maker has its own stringent checklist dictating everything from error messages to achievements, and if a game doesn’t make the cut, the publisher will have to fix things and try again, deadlines be damned. “Microsoft required all games to be able to access the Xbox 360 menu from literally everywhere in the game,” said one tester who worked in compliance for a major game publisher. “Sony required that you couldn’t skip the first

time you see the publisher/development studio screens at the beginning of the game. Nintendo didn't want any swearing in their games so every text input had an extensive filter we had to try and break.”

The Telltale Speedrun

“I didn't play *BioShock Infinite* for at least two years after its release,” an ex-QA tester told me recently. He'd worked for 2K and spent extensive time testing out *Infinite*, and he wasn't really happy with how the game turned out, noting that it didn't live up to the original.

“The one thing that got me to play the game again was seeing speed runs of *BioShock*. [We'd spent] a lot of nights speed-running the game. It was interesting to see what the speed-runners used to cut levels.”

One night recently, the tester saw this video of a speedrunner breaking the game's world record. When he got to the 11-minute mark, he started freaking out.

“I got so mad because they use a bug to get outside the level and automatically move forward,” the tester said in an e-mail. “I SHOULD HAVE CAUGHT IT!”

For a long time the video game industry presented QA testing as a dream job: hey kids, come get paid to play games all day! But in recent years, horror stories have emerged—QA testers tell tales of monotonous work, grueling hours, and poor treatment from companies that see them as replaceable cogs in the development machine.

There's the money problem, for example. The job has low requirements—typical entry-level tester positions don't need much experience or even a college degree—and there's a ton of demand, so the pay is mediocre. A 2014 *Gamasutra* salary survey pegged the average QA tester's yearly pay at

\$54,833, but that only reflected full-time staff, most testers are contractors working either directly with developers or at third-party QA houses that juggle multiple publisher clients. Many of those contractors told me their salaries ranged from \$10 to \$15 an hour—that averages out to \$21–31,000 per year.

Testers also talk of feeling disrespected at the workplace. Many in QA, especially contractors, have told me they weren't allowed to speak directly to the developers, and that their only communication came in the form of written bug reports. "It was also sort of an unspoken rule that temps shouldn't directly contact the devs," one tester told me. "Any communication was typically routed through the full-time QA Leads. As testers, all our interactions with dev were through comments in the bug database, which is far from an ideal form of communication. It was easy to interpret a developer's comment/question on a bug as snarky/irritated when that wasn't necessarily their intention."

This isn't the case at every studio—"When you give your testers benefits, upward mobility, job security, and respect, it attracts the right people for the job," said Ariel Smith, a tester at the MMO studio Cryptic who told me she loves the gig—but disrespect for QA is certainly a widespread trend. Several testers told me they've had to use side entrances to enter their offices and that they weren't allowed to mingle with the rest of the staff. Others said it's common for other developers to screw with them in all sorts of ways; one frequent story, for example, is of the engineer who fixes a bug yet continually sends QA messages like "could not reproduce." In a typical studio, QA is seen as the bottom of the totem pole. This is in part because of the nature of their job—a tester's role is to show other people where they screwed up. That's always going to bruise some egos.

"The QA only care about finding bugs; the developers only care about fixing them," a former tester told me. "They aren't a team and they aren't working together. It's almost like a game of tennis. The testers actually want the build to be broken because it ensures they have work to do. So, the two sides are kind of working against each other, which isn't healthy for production."



At some game companies, higher-ups give testers strict bug quotas and threaten to shorten their contracts if they don't find enough glitches, which can result in a weird sort of tension as testers compete over who's finding the biggest bugs first. Sometimes, QA employees will find creative ways to work more hours so they get paid more and make themselves seem more valuable to the company. "There were some testers who would hold onto bugs to make sure there's overtime," one tester told me. "If there was no OT scheduled for the weekend they would enter a [major] bug Friday afternoon. In some cases this would cause overtime."

QA testers also have to deal with game development's other systemic issues—particularly, mandatory crunch and frequent layoffs. Big developers tend to hire dozens of QA testers toward the end of big projects only to let them all go once the game ships. Instead of partying with the rest of the developers, they're out looking for new jobs.

This all adds up to what would seem to outsiders like an unquestionably heinous gig, but the job does have bright points. A number of current and former testers have told me that despite QA's many challenges, testing video games can be rewarding and educational in a unique way.

“I’ve enjoyed my time as a tester and I would do it again if I had to,” said Obed Navas, a former tester who worked on games like *BioShock* and *Call of Duty*. “Even though ‘QA Tester’ might not be the most glamorous title, and you run the risk of losing the passion to play at home, in the end, being able to see your name in the credits, and having specific items on you that people know you can’t find anywhere ...and them asking you where you got it, being able to respond ‘I worked on it,’ it’s a great feeling and I take pride in that.”

Lunchtime Woes

Rob Hodgson, a video game tester, was working on an early build of the multiplayer game *Fallen Earth* during lunch one day when suddenly the servers crashed. Not long afterwards, they crashed again. Baffled, he tried to reproduce the bug, but he had no luck—neither he nor anyone else on his team could figure out what was causing such a catastrophic error.

“It made no sense,” Hodgson told me recently. “We got one of the programmers to watch the logs as it happened, and he all but shrugged and threw up his hands. It seemed like someone had run off the edge of the world, and the servers were choking, trying to track him. But no one on the [quality assurance] team was testing the world edges, and the designers were all out to lunch!”

After scratching their heads for a while, Hodgson and team asked around and eventually found out what was bringing down the servers. “Turns out that the designers going out to lunch was precisely the problem,” Hodgson said. “One of them had just been setting his horse on autorun and going out to eat. Some days it would hit a tree and get stuck. Some days it would get into the endless, flat plains beyond, and melt the servers.”

From *Assassin's Creed Unity* to the PC version of *Arkham Knight*, today's games seem to be shipping with more problems than ever before. It's easy to pin the blame on QA. Sometimes, that very well may be the case—a number of veterans told me that they've worked alongside sloppy, apathetic testers who just didn't care enough to do things right. Two different former testers told me they'd watch colleagues smoke weed during lunchtime just about every day; one said the QA staff would all wear sunglasses in a futile attempt to hide it.

But lots of testers say they're *finding* most if not all the bugs that ship in today's games. The problem is that nobody's fixing them.

Most testers operate using a process called triage, where bugs are prioritized based on importance. Top priority are issues that make the game crash—the “showstoppers,” as they're called. Other glitches are categorized based on how important testers think they are. Usually, producers and programmers will take the time to fix showstoppers—it'd be hard for any game to make it past certification with any of those. But small and even moderate-sized bugs often stick to games like barnacles, the victims of tight deadlines and programmers who can only do so much in the time they're given.

“We would often find that either the risk of fixing a bug or the time it would take weren't worth it, especially when we were doing something really uncommon or deliberately breaking the game,” said one tester who worked on *The Elder Scrolls V: Skyrim*, a game that's known for being both extraordinarily massive and uncommonly buggy.

“Some of our bugs would get resolved as ‘won't fix’ or ‘post-release.’ Post-release meant, ‘This would be nice, but it isn't really needed right now. If it becomes a thing people are calling out on the forums or online, maybe we'll reassess this.’ I think there's a lot more leeway nowadays than there was maybe 20 years ago, [because] you can say, ‘OK we have this nasty bug but it can be fixed in the day-one patch.’ That's acceptable behavior now.”

As games get bigger and bigger, making leaps in both graphical fidelity and mechanical complexity, they're generating more and more bugs—bugs that might not be so easy to catch or fix. Development studios can't just change a schedule because they're overwhelmed by glitches. Unless the publisher agrees to what could be a costly delay, that holiday release date ain't going anywhere, no matter how broken the game might be. So during testing, several QA staffers told me, they focus just on catching the bugs that would prevent them from getting through console certification. That might mean ignoring some of a game's other, possibly more significant problems.

“The game developers and the publishers had independent QA teams,” recalls one compliance tester who worked for a major publisher. “We would submit a bug with as much detail to recreate the condition as possible, the development team would determine if it was really a bug or not, and then they would either invalidate the bug or attempt to fix it. We would receive a revised build with ‘X’ number of bugs handled, double-check the problem was fixed, and then move on to the next thing. This aspect of the process, from a gamer's perspective, was the most frustrating. Bugs that caused inconvenience for the player were often considered invalid because they wouldn't affect the ability to release the game, and could be addressed later if people got upset.”

Other compliance testers say that, due to deadlines, they've had to find loopholes or do the bare minimum necessary to sneak through certification and start printing discs, promising that a day-one patch would come on release day.

“If we were getting down to the wire and multiplayer was super broken, we'd focus test efforts on single-player/offline areas instead,” a tester told me over e-mail. “[Console-makers] would often fail games for severe issues in online functionality, but still allow us to release to manufacturing on the condition we provided a day-one title update to fix issue(s) X, Y, and/or Z. There wasn't a way to guarantee all players would get a patch for single player/offline bugs since offline gameplay wouldn't require an internet connection. So we'd tighten up offline content for final certification, knowing there were issues with online content, and then scramble to patch the online issues in time for launch. This was (unfortunately) a rather common process, and was usually due to a rushed & unrealistic timeline rather than dev/QA incompetence.”

Sometimes, bugs will make it into a game for reasons that seem baffling to outside observers. Writes one tester who did contract work for Nintendo of America: “The Sky Drop bug in *Pokémon Black and White* was something that we caught, but the game had already been released in Japan at that point so instead of fixing the issue they purposefully left it in the game to keep parity.” The tester added that despite these idiosyncrasies, Nintendo makes an explicit point of recording every single bug and asking testers to include video footage for every report they submit. “I think this contributes greatly to their reputation for having quality games,” he said.

One other compliance tester's explanation for the increasing complexity of bug-testing might make gamers angrier than ever at publishers: it's DLC, he says.

“Let's say we have a game with 40 pieces of microtransactions/DLC,” the tester said in an e-mail. “Even if some or all these 40 DLC packages are simple unlocks of on-disc content, we have to test every possible combination of these 40 DLC packages, in all game modes, using different

combinations of storage media (20 DLC packs on the hard drive, 15 on a USB flash drive, 5 on a memory unit, etc), mixing up what the host has versus what clients have in different online game modes, verifying graceful handling in the event some dipshit has all his DLC on an external storage device and decides to unplug that device, with each Title Update (1.01, 1.2, etc), with different save data, etc., etc.

“It could (and did) get the point where it’s literally impossible for humans to physically test all possible combinations/situations. It’s something that would get exponentially more difficult as more variables are added—like if you can purchase DLC in-game.”

In short, this shit is tough. Multiplayer games are especially difficult for today’s QA departments to handle; even hundreds of testers can’t successfully replicate what will happen when hundreds of thousands of people are all shooting aliens on live servers. And while some testers say developers are working more closely with QA than they have in the past, nobody can change an unrealistic deadline.

“Milestone schedules nowadays are totally absurd,” one former tester told me. “What [a] lot of people probably don’t understand is that even for titles that have been in development for, say, three years, only 1.5 of those years is actually full, proper production. And of that, only nine months in QA. And of that, only three months at full QA capacity. By then, we might be at content lock. And so by the time we are able to speak out, the game is reaching Beta. This is a bit of an exaggeration for some larger titles, but reality for many I’ve worked with. When QA speak, we are heard... there’s just no time to react.”

‘No Hard Feelings’

Writes one former tester:

We were bundled away into bullpen. I was working with people that had been testing [game] for years and years, were so burnt out on it, but were still going because they had no other hope. Out of the probable hundreds of testers they go through a year, I think maybe two or three ever got hired on. They weren't acquiring any new, valuable experience or skills during all that time. We got condescending emails saying "Congratulations! You've just earned eight hours of vacation time!" like we were kids getting stars for good behavior. I don't need that. Just put it on my paystub.

Meanwhile, new cars were showing up in the lot every couple of months. Ultra fancy, decked out cars that probably cost more than four or five testers' annual income. They were throwing parties for themselves. When you're grinding away and can't afford new clothes regardless and you see that sort of thing, it takes a toll. When we launched, there was a massive round of layoffs. I think close to around 100, but I have no idea. All I know is, QA was pretty much wiped to bare bones. A few hours after they were all escorted off the campus, the launch party kicked off.

Things got bad for me in a lot of ways; a girl I was seeing at the time [left me]... I met a lot of good people that knew how terrible the situation was for us temps, and good people were leaving because of it. It frustrated everyone I worked with. I got bumped around from team to team. That last team was small with way too much work, but I learned a lot and they were really good people. When I told them I was leaving, there were no hard feelings. There was almost a hint of encouragement, like I was telling them I was going to quit drinking or something beneficial like that.

Is QA a viable long-term career? Or will it forever be a means to an end, a pathway to other, more interesting parts of game development?

While reporting for this story, I talked to around 60 QA testers, and of those, only four said they saw QA as a long-term career. Many used QA as a way to segue into producer or designer jobs at game companies; others tried it for a few months or years and then gave up, moving on to more lucrative fields. Those who do stay in QA and succeed in the role can find themselves moving into QA management, a field that's more about handling other people than

it is helping make video games. That's unsatisfying for some folks, especially given that the stereotype of testers being young, immature, and unkempt is often true.

“As you move into the role of QA Management, you start to have experiences that nobody could ever prepare you for,” said one veteran QA lead. “Testers showing up in pajamas for work, testers sexually harassing one another, testers getting caught stealing... Even the most mundane thing like ordering in food can become a challenge. QA is a field that skews very young. Having the average age of a team be 18-19 is not uncommon, and often times this is their first real job. It's a massive challenge to deal with young personalities and hormones.”

Several experienced game developers have told me there are few employees more valuable than a senior QA person—someone who knows how to properly write bug reports; who knows to focus on big-picture issues instead of some clipping or graphical issues; who knows how to work with other people to maximize efficiency and avoid dupes. Those people are rare—low pay and poor treatment combine to drive many QA testers away from the field before they get very far.

“Before I left, I sat down with my manager, the associate producer on the game, and asked him [for] advice,” said James, a former tester who asked me not to use his last name. “I wanted to know how to make the jump from temp QA to permanent QA in development or anywhere else. He blatantly told me, ‘Distance yourself from QA... the sooner you can get out of QA and distance yourself from ever working in it the better.’ After about another eight months, I left QA for good and gave up on working in the video game industry. I am now working at a developer of commercial software, and life is a little less fun, but much more stable. The money is also much better.”

Based On A True Story

The movie *Grandma's Boy*, which came out in 2006 and stars Allen Covert as a video game tester, is something of a running joke among QA. People always ask testers: how much of *Grandma's Boy* is true? The answer is usually “not a lot.”

One scene in the movie, in which the egotistical video game designer J.P. looks over a failed prototype and watches his main character's head fall off, is often brought up as indicative of the movie's inaccuracies (skip to 1:31):

Silly, right? Bugs don't really happen like that. Except for when they do.

“It was in a certain Kinect game that involved obstacle courses in a TV show style presentation,” a tester for a major publisher told me. “There were quick intros of the show's announcers at the beginning of each level where they'd provide commentary and set the scene. Once, and only once, one of the reporter's head shifted off his neck and into empty space next to his body - then dropped straight down off of the screen. Nobody could ever reproduce it, and trust me, we tried.

“It was surreal - here I'd developed a distaste for *Grandma's Boy* due to its inaccurate and somewhat trivializing take on my profession (‘Did that guy's head just fall off? That shit doesn't HAPPEN in real games!’), and it was being proven right in a small way. I'll never forget it.”

Fantasies like Westwood College's are a far cry from the real world of video game testing. In real QA, the pay is low, the work is tough, and many testers find themselves in unenviable positions, outsourcing or doing contract work for game developers who don't even know their names. (Writes one tester: “I was once sent an email from HR asking for input on planning the company's holiday party—I responded with some feedback and was then told that temporary employees weren't invited.”)

Even those who found QA to be a satisfying, rewarding job say things could be better. Way better. A widespread push for change could lead not just to healthier work environments but to more experienced, efficient testers who

work directly with game developers to ensure that more bugs get fixed.

“Game testing is, in general, a job of unsung and untrained heroes,” reads a report by an old QA company called ST Labs. “For some companies, a typical game testing strategy still consists of throwing sheer numbers of testers on a game as it nears release and hoping that they find the big problems. Understanding the games issues and technology and going about game testing in a systematic and thorough manner is still a new concept for most game development companies.”

That report, written by Duri Price and Ilya Pearlman, came out in 1997. Eighteen years and four console generations later, not much has changed. The good game testers are still unsung and untrained heroes, doing their best to get games fixed despite unreasonable schedules, unmotivated co-workers, and a job that’s unappreciated at best.

There’s another way of looking at it, too.

“You **will** get stuck on shitty games, you **will** get stuck with shitty people, and you **will** get stuck with a Producer/PM that thinks QA Testers are **shitty** and beneath them,” one veteran said. “You will work long hours, be underappreciated and see your amazing bug that you spent six hours working on be swept aside and labelled Known Shippable. But you will also see your name in the credits, you will also realize that you contributed to something really cool, and you will also move on to another amazing project.”



Open kinja-labs.com

Top illustration by Jim Cooke

You can reach the author of this post at jason@kotaku.com or on Twitter at [@jasonscreier](https://twitter.com/jasonscreier).

ABOUT THE AUTHOR

Jason
Schreier

Jason Schreier

News editor. Author of *Blood, Sweat, and Pixels*.

You May Like

Sponsored Links by Taboola

If You're Over 40 And Own A Computer, This Game Is A Must-Have!

Throne: Free Online Games

Feeling Old? Do This Once A Day And Watch What Happens

LCR Health

\$699 average annual savings for drivers who switch and save.

Progressive

The Early Signs Of Type 2 Diabetes - Research Type 2 Diabetes Treatments

Yahoo Search

Lift Sagging Skin And Jowls Without Surgery (Do This)

City Beauty

If Your Dog Eats Grass (Do This Everyday)

Ultimate Pet Nutrition

EXHIBIT 18

SAN DIEGO (/CAREERS/OFFICES/ROCKSTAR-SAN-DIEGO) . ONLINE GAME SERVICES (/CAREERS/OPENINGS/DEPARTMENT/ONLINE-GAME-SERVICES) TECHNICAL QA

Rockstar San Diego is seeking a Technical QA Test Analyst to join our Online Services team.

Successful candidates will help the QA team and work with online services, analytics, game development, and the global QA organization to ensure that reliable and high-quality online and backend services are delivered and maintained throughout development. As QA staff, you will create and refine test plans to ensure increasingly thorough test coverage, will provide the latest testing status of features in development, and will collaborate closely with the extended team to deliver world-class online features and services.

RESPONSIBILITIES

- Build or extend test automation frameworks for online services.
- Create and refine test plans for a wide range of online services and extensions of game features utilizing the backend servers for multiple game projects.
- Diagnose code and logs to determine root cause of issues in development and production environments.
- Track history of end-to-end test planning and collaborating closely with development teams to generate and execute on those plans.
- Increase awareness of feature issues that require extra attention.
- Assist in certification and signoff that features are ready for release at different development stages.
- Continuously make improvements to the testing process to make it more efficient and effective.

QUALIFICATIONS

- Bachelor's Degree in Computer Science or related field or equivalent experience (preferably technical discipline).
- 3+ years of experience testing highly complex web applications or services.
- Experience and proficiency with one or more coding or scripting languages (Python, JavaScript, XML/HTML, C#, C/C++/C#, Java, SQL).
- Familiar with a myriad of testing methodologies such as test-driven development, unit testing, compatibility testing, user experience testing, performance testing, integration testing, and traditional regression testing.
- Experience with gray box, white box and various automation test methods.
- Closely familiar with the software development cycles and version control.
- Experience with bug management, tracking and reporting programs (i.e. TestTrack Pro, JIRA, Bugzilla).

DESIRED

- Experience in a software engineering role with a highly complex and large consumer-based software product or online service.
- Proficient in developing and executing unit tests.
- Shipped game titles on Playstation, Xbox and/or PC.

APPLY NOW (HTTPS://APP.JOBVITE.COM/COMPANYJOBS/CAREERS.ASPX?K=APPLY&J=OGB85FWO&C=QY09VFWA&L=CCKHVF)

 [Tweet \(http://twitter.com/share\)](http://twitter.com/share)

 FOLLOW ROCKSTAR GAMES CAREERS (HTTPS://WWW.LINKEDIN.COM/COMPANY/ROCKSTAR-GAMES)

SEE ALL 'SAN DIEGO' POSITIONS (/CAREERS/OFFICES/ROCKSTAR-SAN-DIEGO)

SEE ALL ONLINE GAME SERVICES JOBS (/CAREERS/OPENINGS/DEPARTMENT/ONLINE-GAME-SERVICES)

CORPORATE ([HTTP://WWW.ROCKSTARGAMES.COM/CORPINFO](http://www.rockstargames.com/corpinfo)) PRIVACY ([HTTPS://WWW.ROCKSTARGAMES.COM/PRIVACY](https://www.rockstargames.com/privacy)) LEGAL
([HTTPS://WWW.ROCKSTARGAMES.COM/LEGAL](https://www.rockstargames.com/legal)) ACCESSIBILITY ([/CAREERS/ACCESSIBILITY](https://www.rockstargames.com/careers/accessibility))

EXHIBIT 19



US006714966B1

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

(10) **Patent No.:** **US 6,714,966 B1**
(45) **Date of Patent:** ***Mar. 30, 2004**

- (54) **INFORMATION DELIVERY SERVICE**
- (75) **Inventors:** **Fred B. Holt**, Seattle, WA (US); **Virgil E. Bourassa**, Bellevue, WA (US)
- (73) **Assignee:** **The Boeing Company**, Seattle, WA (US)
- (*) **Notice:** Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 467 days.

5,734,865 A	3/1998	Yu
5,737,526 A	4/1998	Periasamy et al.
5,754,830 A	5/1998	Butts et al.
5,761,425 A	6/1998	Miller
5,764,756 A	6/1998	Onweller
5,790,548 A	8/1998	Sistanizadeh et al.
5,790,553 A	8/1998	Deaton, Jr. et al.
5,799,016 A	8/1998	Onweller
5,802,285 A	9/1998	Hirviniemi
5,850,592 A	* 12/1998	Ramanathan 455/7
5,864,711 A	1/1999	Mairs et al.
5,867,660 A	2/1999	Schmidt et al.
5,867,667 A	2/1999	Butman et al.
5,870,605 A	2/1999	Bracho et al.

This patent is subject to a terminal disclaimer.

(List continued on next page.)

- (21) **Appl. No.:** **09/629,043**
- (22) **Filed:** **Jul. 31, 2000**
- (51) **Int. Cl.⁷** **G06F 15/16**
- (52) **U.S. Cl.** **709/204; 709/205; 709/203; 709/243; 463/92**
- (58) **Field of Search** **709/204, 205, 709/227, 243, 203; 463/40, 42**

OTHER PUBLICATIONS

PR Newswire, "Microsoft Boosts Accessibility to Internet Gaming Zone with Latest Release," Apr. 27, 1998, pp. 1ff.*
 PR Newswire, "Microsoft Announces Launch Date for UltraCorps, Its Second Premium Title for the Internet Gaming Zone," Ma 27, 1998, pp. 1 ff.*
 Business Wire, "Boeing and Panthesis Complete SWAN Transaction," Jul. 22, 2002, pp. 1ff.*

(List continued on next page.)

(56) **References Cited**
U.S. PATENT DOCUMENTS

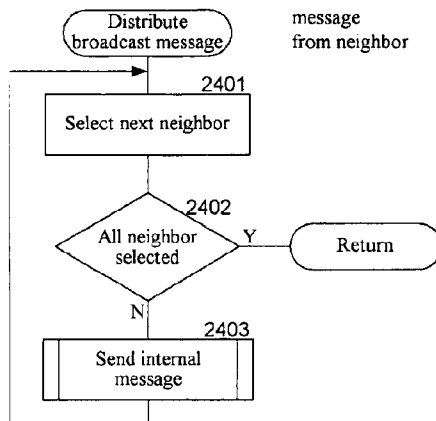
4,912,656 A	3/1990	Cain et al.
5,056,085 A	10/1991	Vu
5,058,105 A	* 10/1991	Mansour et al. 370/228
5,079,767 A	* 1/1992	Perlman 370/408
5,117,422 A	* 5/1992	Hauptschein et al. 370/255
5,309,437 A	5/1994	Perlman et al.
5,426,637 A	6/1995	Derby et al.
5,459,725 A	* 10/1995	Bodner et al. 370/390
5,471,623 A	* 11/1995	Napolitano, Jr. 709/243
5,535,199 A	7/1996	Amri et al.
5,568,487 A	10/1996	Sithon et al.
5,636,371 A	6/1997	Yu
5,644,714 A	* 7/1997	Kikinis 709/219
5,673,265 A	9/1997	Gupta et al.
5,696,903 A	12/1997	Mahany
5,732,074 A	3/1998	Spaur et al.
5,732,086 A	* 3/1998	Liang et al. 370/410
5,732,219 A	3/1998	Blumer et al.

Primary Examiner—Dung C. Dinh
Assistant Examiner—Brad Edelman
 (74) *Attorney, Agent, or Firm*—Perkins Coie LLP

(57) **ABSTRACT**

A computer network for providing an information delivery service for a plurality of participants over the network is disclosed. Each participant has connections to at least three neighbor participants. An originating participant sends data to the other participants by sending the data through each of its connections to its neighbor participants. Further, each participant sends data that it receives from a neighbor participant to its other neighbor participants. The network is m-regular where m is the exact number of neighbor participants of each participant and the network is an incomplete graph.

17 Claims, 39 Drawing Sheets



US 6,714,966 B1

Page 2

U.S. PATENT DOCUMENTS

5,874,960 A 2/1999 Mairs et al.
 5,899,980 A 5/1999 Wilf et al.
 5,907,610 A 5/1999 Onweller
 5,925,097 A * 7/1999 Gopinath et al. 709/200
 5,928,335 A 7/1999 Morita
 5,935,215 A 8/1999 Bell et al.
 5,948,054 A 9/1999 Nielsen
 5,949,975 A 9/1999 Batty et al.
 5,956,484 A 9/1999 Rosenberg et al.
 5,970,232 A * 10/1999 Passint et al. 709/238
 5,974,043 A 10/1999 Solomon
 5,987,506 A 11/1999 Carter et al.
 6,003,088 A 12/1999 Houston et al.
 6,013,107 A 1/2000 Blackshear et al.
 6,023,734 A 2/2000 Ratcliff et al.
 6,029,171 A 2/2000 Smiga et al.
 6,032,188 A 2/2000 Mairs et al.
 6,038,602 A 3/2000 Ishikawa
 6,047,289 A 4/2000 Thorne et al.
 6,094,676 A 7/2000 Gray et al.
 6,115,580 A * 9/2000 Chuprun et al. 455/1
 6,167,432 A * 12/2000 Jiang 709/204
 6,173,314 B1 * 1/2001 Kurashima et al. 709/204
 6,199,116 B1 3/2001 May et al.
 6,216,177 B1 4/2001 Mairs et al.
 6,223,212 B1 4/2001 Batty et al.
 6,243,691 B1 6/2001 Fisher et al.
 6,268,855 B1 7/2001 Mairs et al.
 6,271,839 B1 8/2001 Mairs et al.
 6,272,548 B1 * 8/2001 Cotter et al. 709/239
 6,285,363 B1 9/2001 Mairs et al.
 6,304,928 B1 10/2001 Mairs et al.
 6,321,270 B1 * 11/2001 Crawley 709/238
 6,463,078 B1 * 10/2002 Engstrom et al. 370/466
 6,524,189 B1 * 2/2003 Rautila 463/40
 2002/0027896 A1 * 3/2002 Hughes et al. 370/342

OTHER PUBLICATIONS

Azar et al., "Routing Strategies for Fast Networks," May 1992, INFOCOM '92, Eleventh Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 1, pp. 170-179.*
 Cho et al., "A Flood Routing Method for Data Networks," Sep. 1997, Proceedings of 1997 International Conference on Information, Communications, and Signal Processing, vol. 3, pp. 1418-1422.*
 Komine et al., "A Distributed Restoration Algorithm for Multiple-Link and Node Failures of Transport Networks," Dec. 199 Global Telecommunications Conference, 1990, and Exhibition, IEEE, vol. 1, pp. 459-463.*
 Peercy et al., "Distributed Algorithms for Shortest-Path, Deadlock-Free Routing and Broadcasting in Arbitrarily Faulty Hypercubes," Jun. 1999, 20th International Symposium of Fault-Tolerant Computing, 1990, pp. 218-225.*
 Alagar, S. and Venkatesan, S., "Reliable Broadcast in Mobile Wireless Networks," Department of Computer Science, University of Texas at Dallas, Military Communications Conference, 1995, MILCOM '95 Conference Record, IEEE San Diego, California, Nov. 5-8, 1995 (pp. 236-240).
 International Search Report for The Boeing Company, International patent application No. PCT/US01/24240, Jun. 5, 2002 (7 pages).
 U.S. patent application Ser. No. 09/629,570, Bourassa et al., filed Jul. 31, 2000.

U.S. patent application Ser. No. 09/629,577, Bourassa et al., filed Jul. 31, 2000.
 U.S. patent application Ser. No. 09/629,575, Bourassa et al., filed Jul. 31, 2000.
 U.S. patent application Ser. No. 09/629,572, Bourassa et al., filed Jul. 31, 2000.
 U.S. patent application Ser. No. 09/629,023, Bourassa et al., filed Jul. 31, 2000.
 U.S. patent application Ser. No. 09/629,576, Bourassa et al., filed Jul. 31, 2000.
 U.S. patent application Ser. No. 09/629,024, Bourassa et al., filed Jul. 31, 2000.
 U.S. patent application Ser. No. 09/629,042, Bourassa et al., filed Jul. 31, 2000.
 Murphy, Patricia, A., "The Next Generation Networking Paradigm: Producer/Consumer Model," Dedicated Systems Magazine—2000 (pp. 26-28).
 The Gamer's Guide, "First-Person Shooters," Oct. 20, 1998 (4 pages).
 The O'Reilly Network, "Gnutella: Alive, Well, and Changing Fast," Jan. 25, 2001 (5 pages) [http://www.open2p.com/lpt/...](http://www.open2p.com/lpt/) [Accessed Jan. 29, 2002].
 Oram, Andy, "Gnutella and Freenet Represents True Technological Innovation," May 12, 2000 (7 pages) The O'Reilly Network [http://www.oreillynet.com/lpt/...](http://www.oreillynet.com/lpt/) [Accessed Jan. 29, 2002].
 Internetworking Technologies Handbook, Chapter 43 (pp. 43-1-43-16).
 Oram, Andy, "Peer-to-Peer Makes the Internet Interesting Again," Sep. 22, 2000 (7 pages) The O'Reilly Network [http://linux.oreillynet.com/lpt/...](http://linux.oreillynet.com/lpt/) [Accessed Jan. 29, 2002].
 Monte, Richard, "The Random Walk for Dummies," MIT Undergraduate Journal of Mathematics (pp. 143-148).
 Srinivasan, R., "XDR: External Data Representation Standard," Sun Microsystems, Aug. 1995 (20 pages) Internet RFC/STD/FYI/BCP Archives <http://www.faqs.org/rfcs/rfc1832.html> [Accessed Jan. 29, 2002].
 A Database Corporate White Paper, "A Primer on the T.120 Series Standards," Copyright 1995 (pp. 1-16).
 Kessler, Gary, C., "An Overview of TCP/IP Protocols and the Internet," Apr. 23, 1999 (23 pages) Hill Associates, Inc., <http://www.hill.com/library/publications/t...> [Accessed Jan. 29, 2002].
 Bondy, J.A., and Murty, U.S.R., "Graph Theory with Applications," Chapters 1-3 (pp. 1-47), 1976 American Elsevier Publishing Co., Inc., New York, New York.
 Cormen, Thomas H. et al., Introduction to Algorithms, Chapter 5.3 (pp. 84-91), Chapter 12 (pp. 218-243), Chapter 13 (p. 245), 1990, The MIT Press, Cambridge, Massachusetts, McGraw-Hill Book Company, New York.
 The Common Object Request Broker: Architecture and Specification, Revision 2.6, Dec. 2001, Chapter 12 (pp. 21-1-12-10), Chapter 12 (pp. 13-1-13-56), Chapter 16 (pp. 16-1-16-26), Chapter 18 (pp. 18-1-18-52), Chapter 20 (pp. 20-1-20-22).
 The University of Warwick, Computer Science Open Days, "Demonstration on the Problems of Distributed Systems," <http://www.dcs.warwick.ac.uk...> [Accessed Jan. 29, 2002].

* cited by examiner

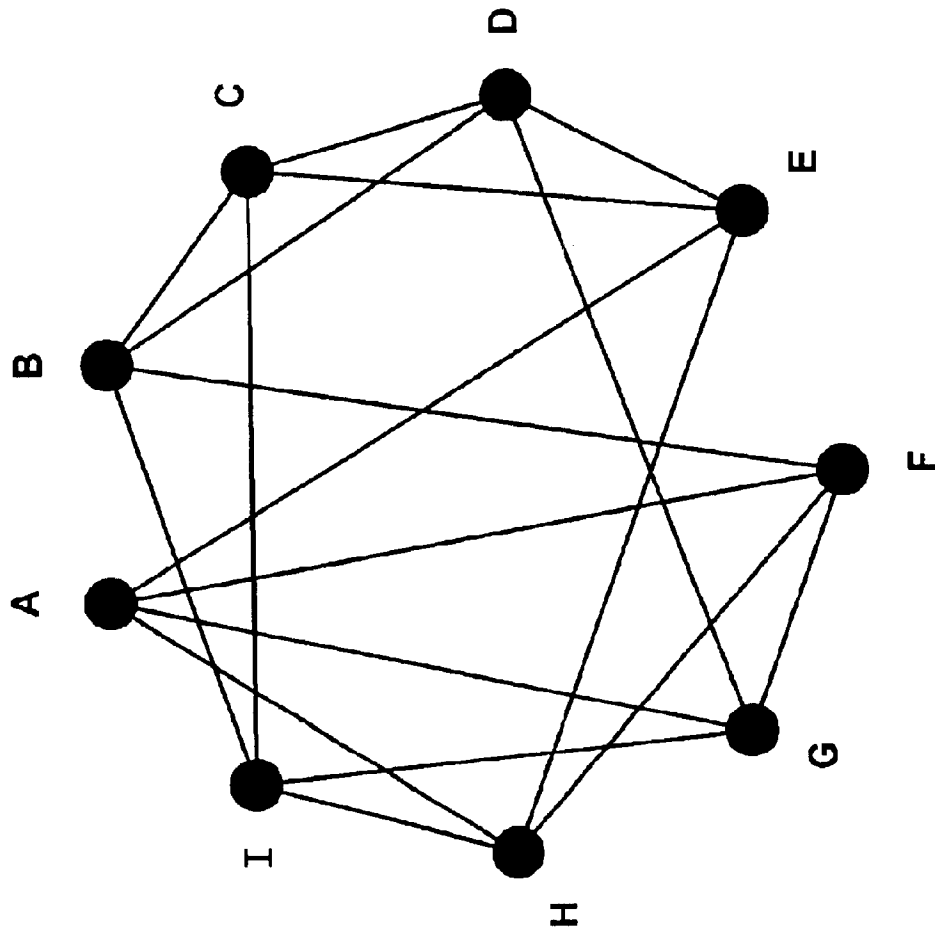


Fig. 1

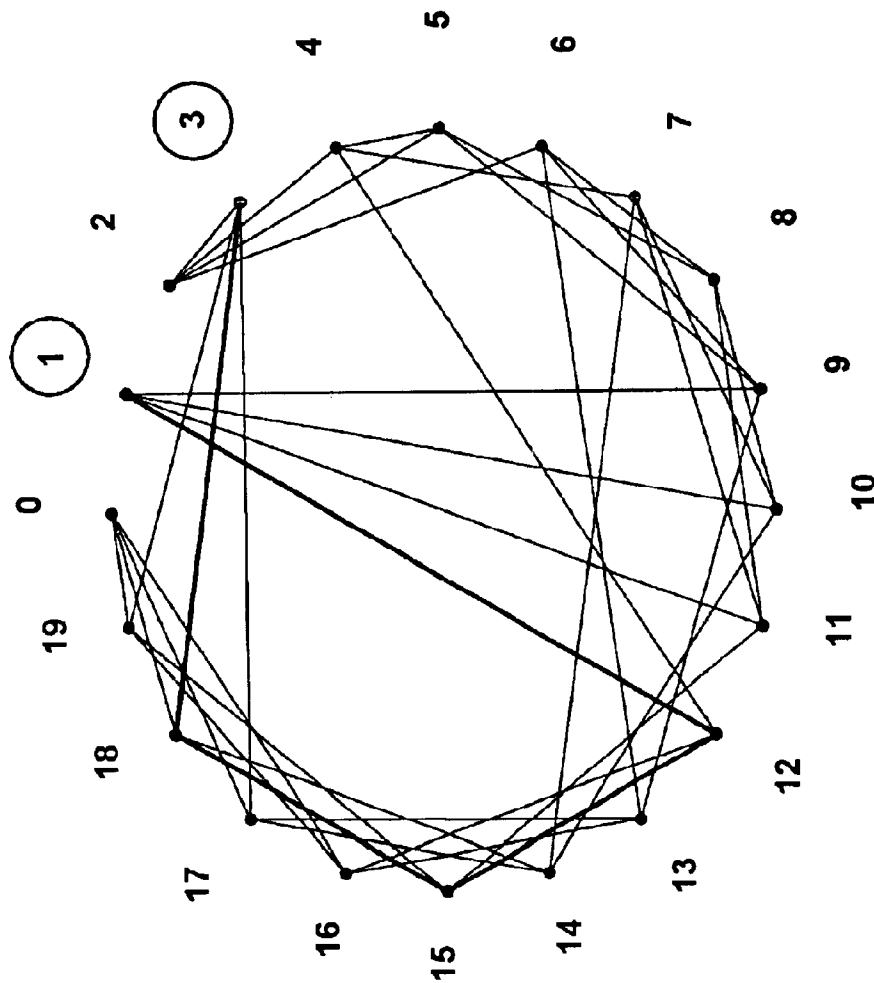


Fig. 2

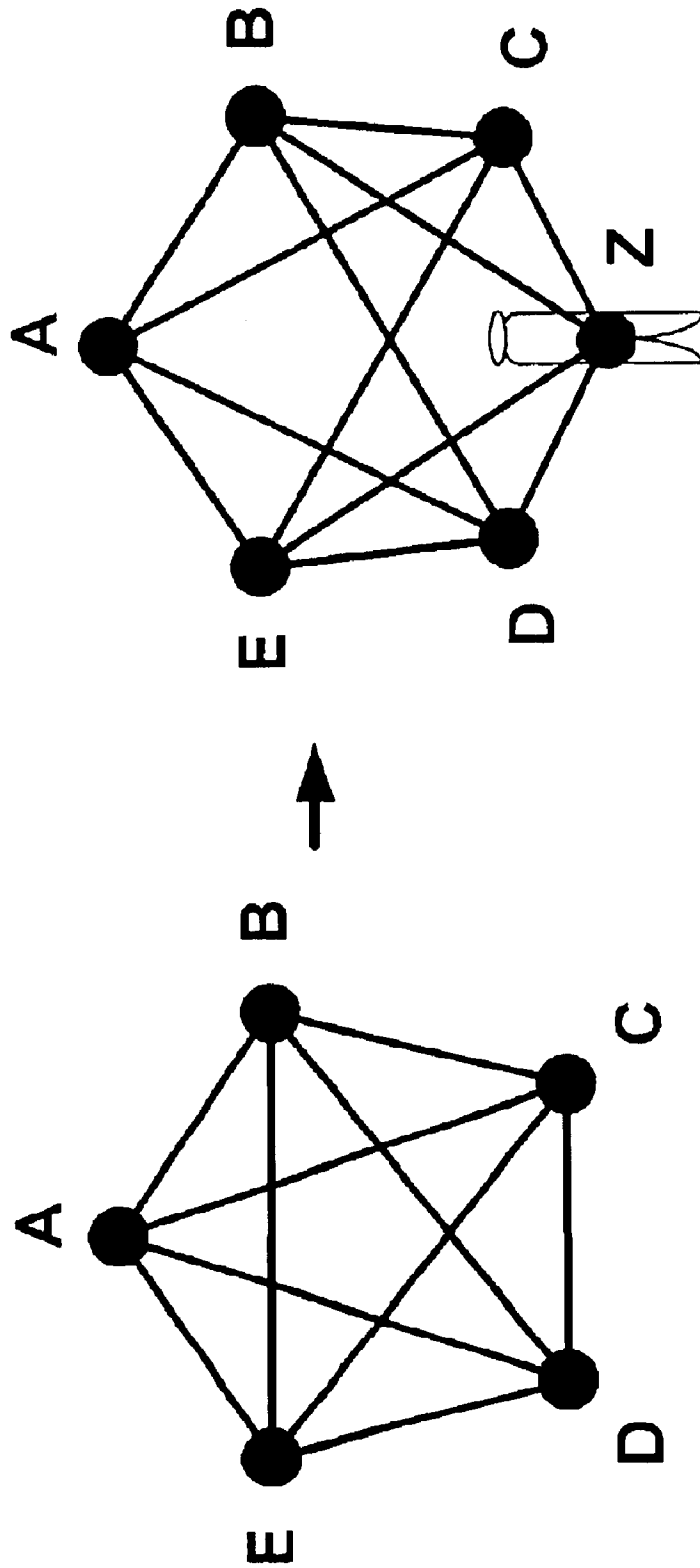


Fig. 3B

Fig. 3A

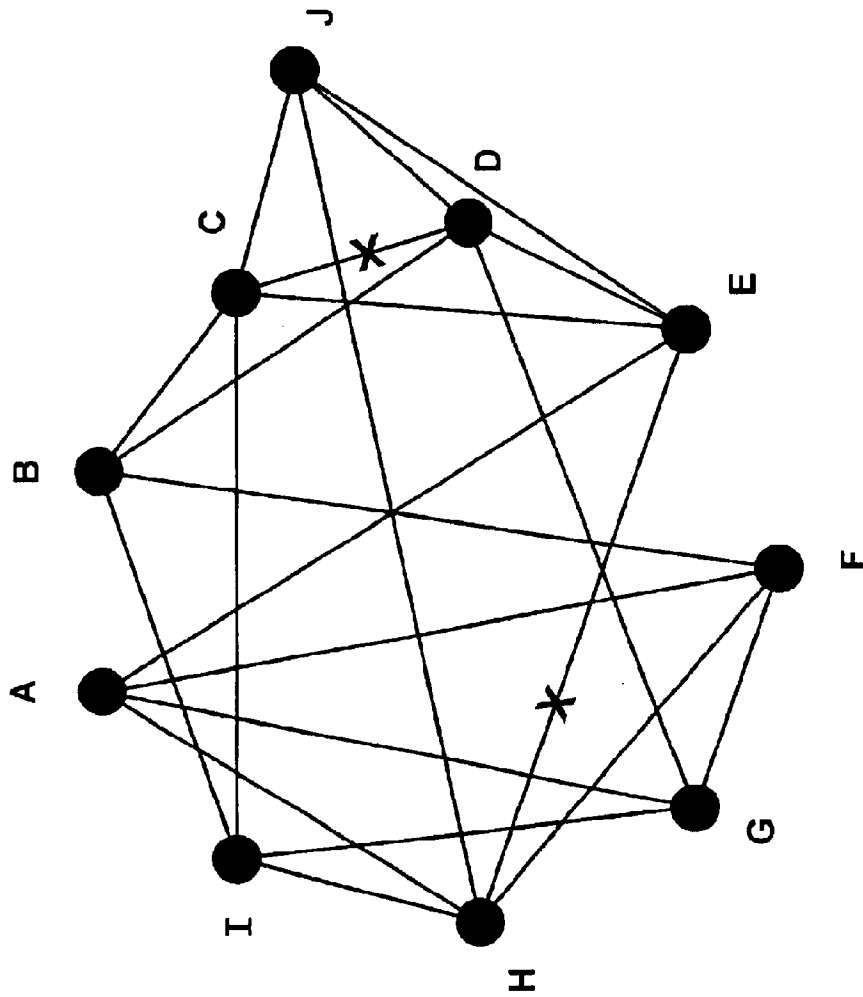


Fig. 4A

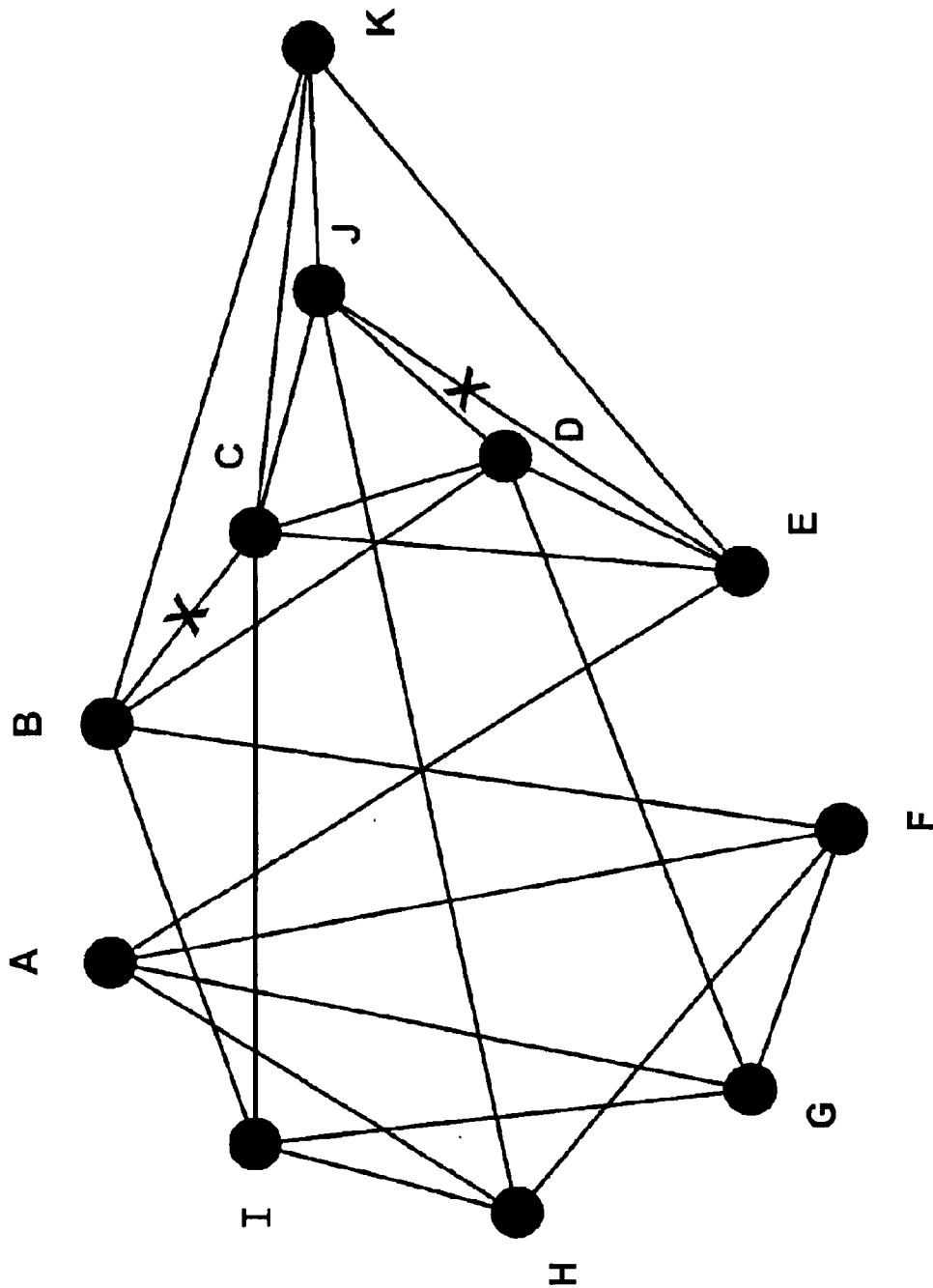


Fig. 4B

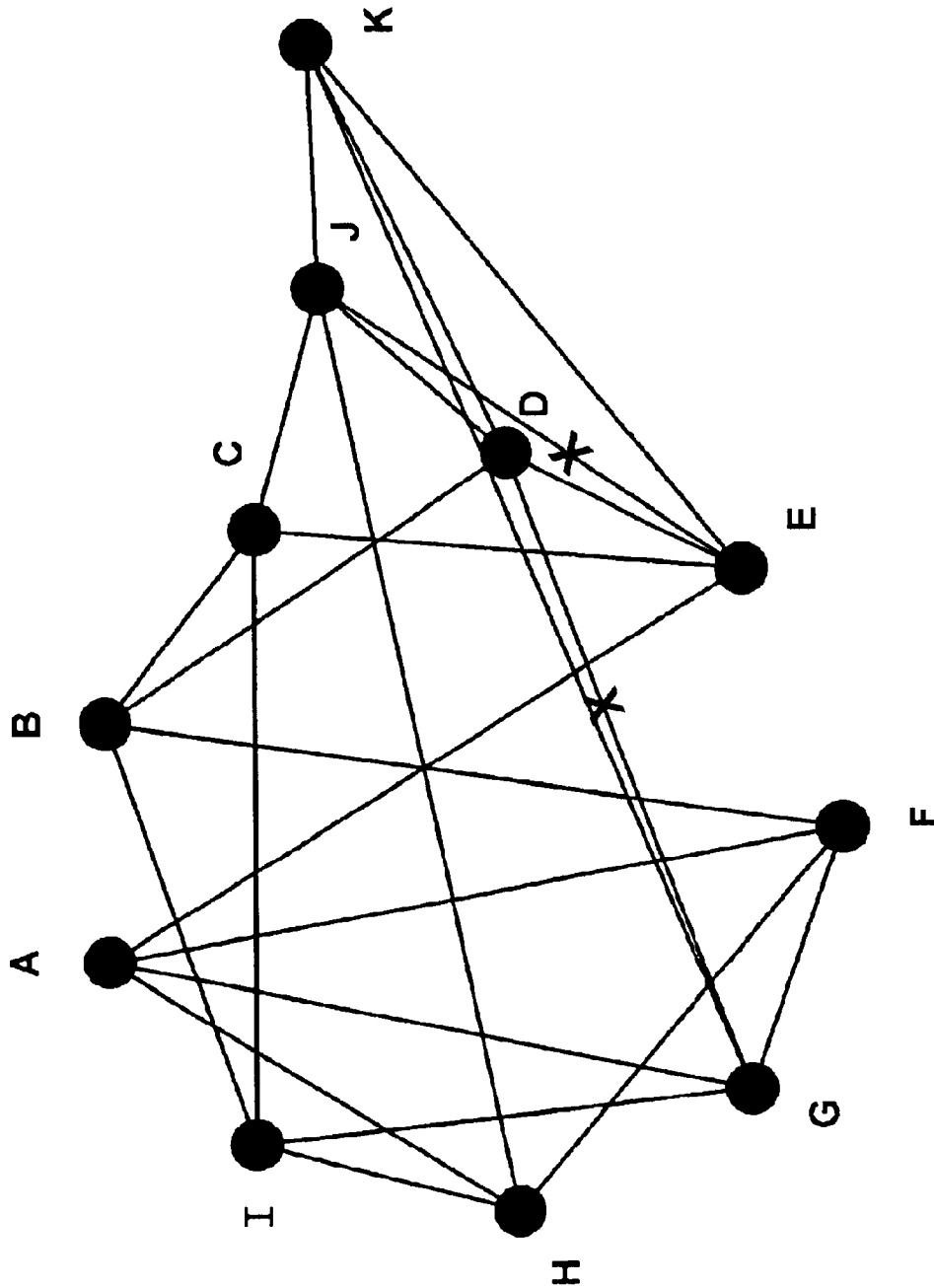


Fig. 4C

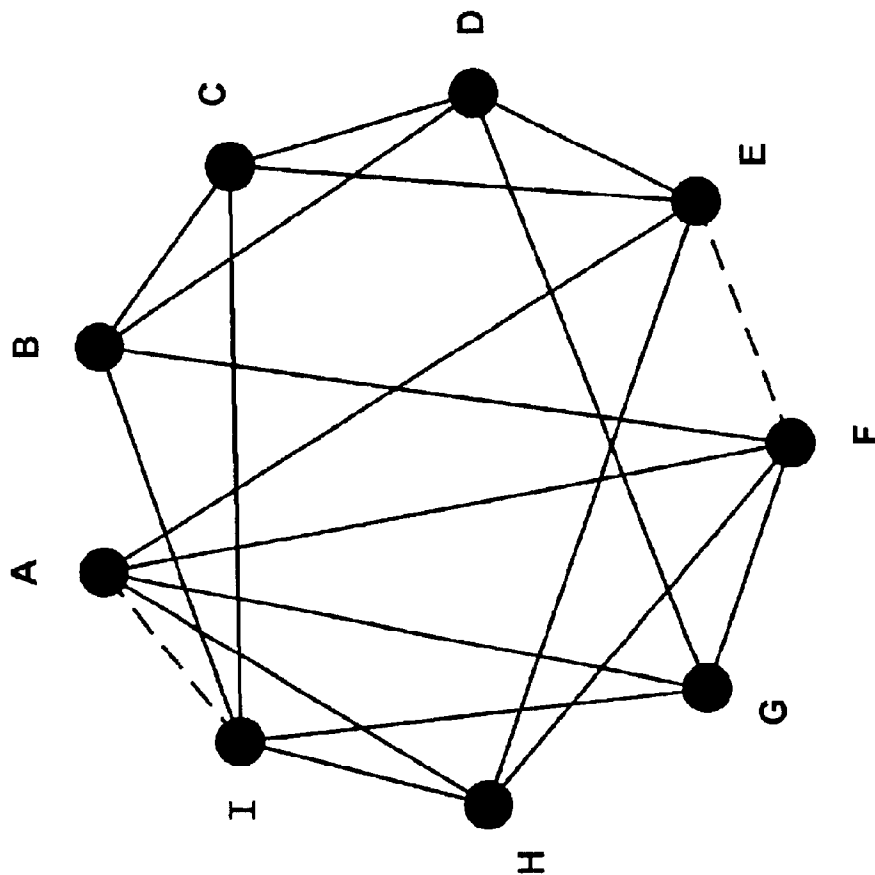


Fig. 5A

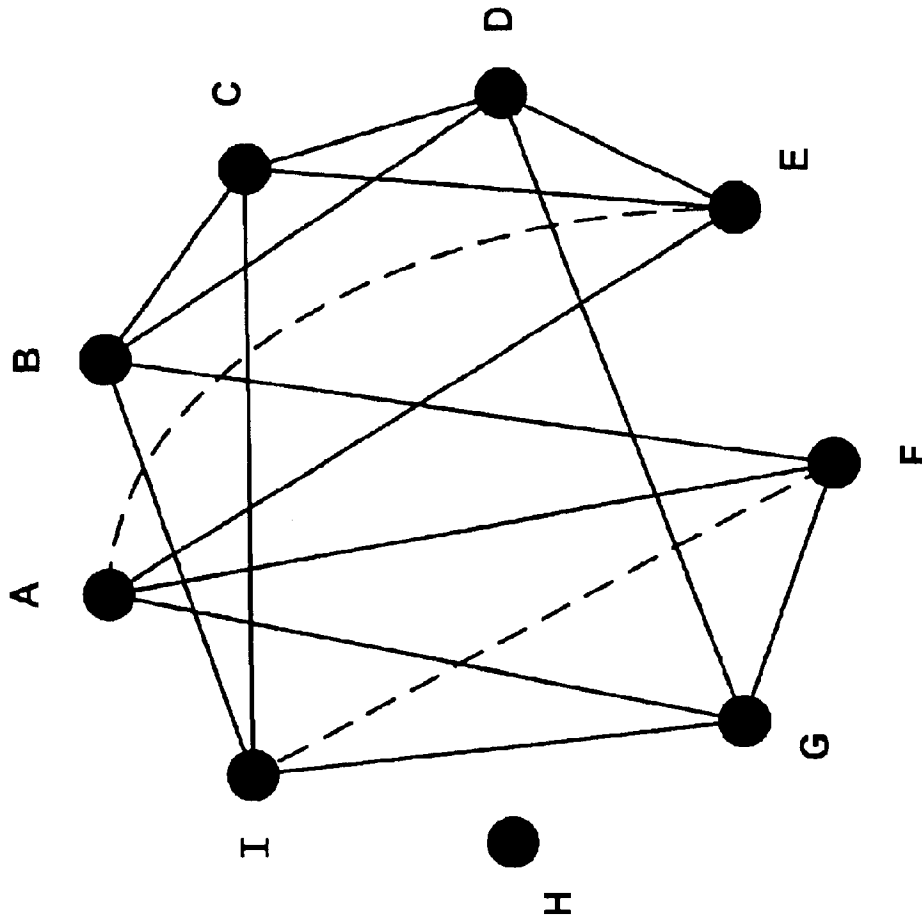


Fig. 5B

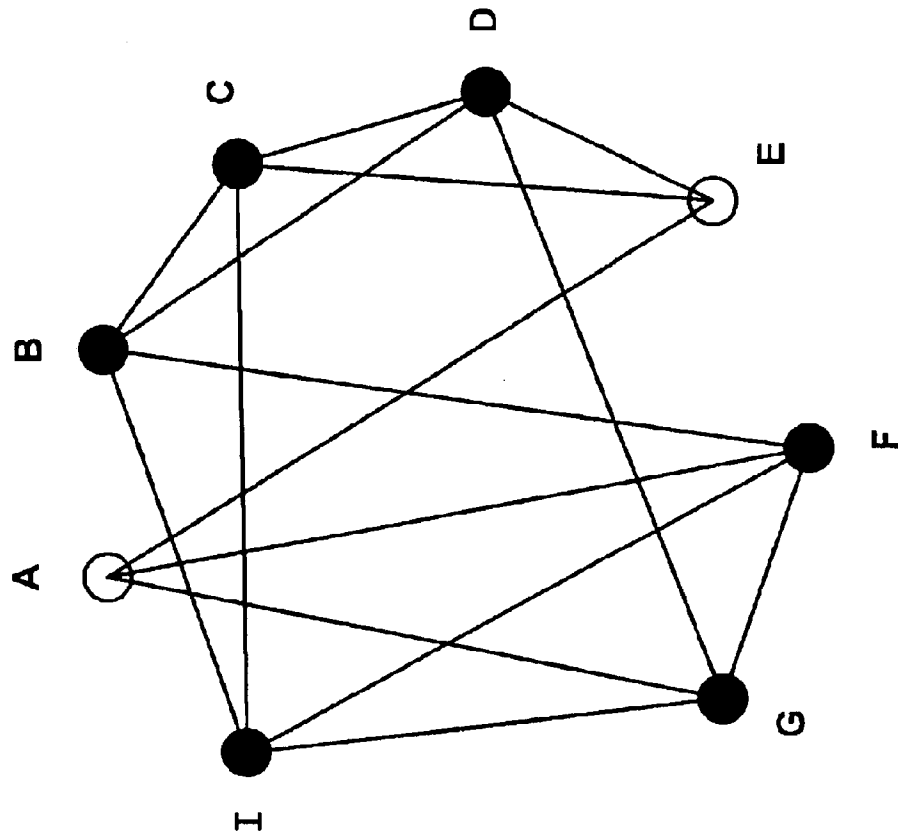


Fig. 5C

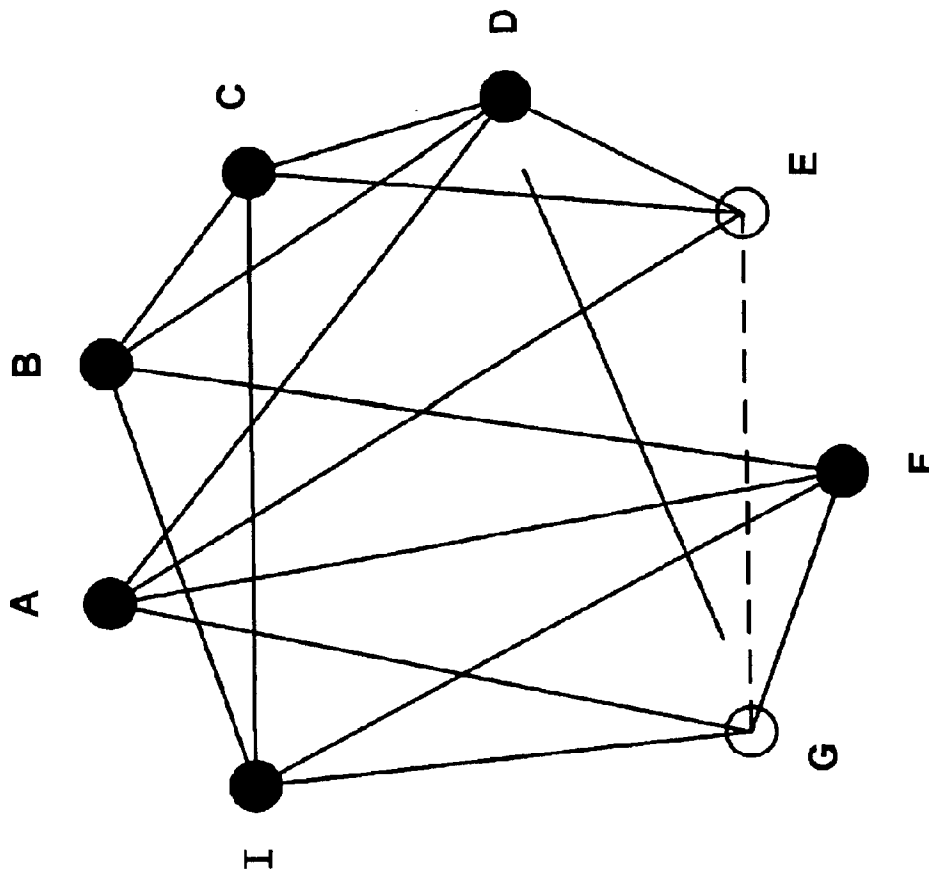


Fig. 5D

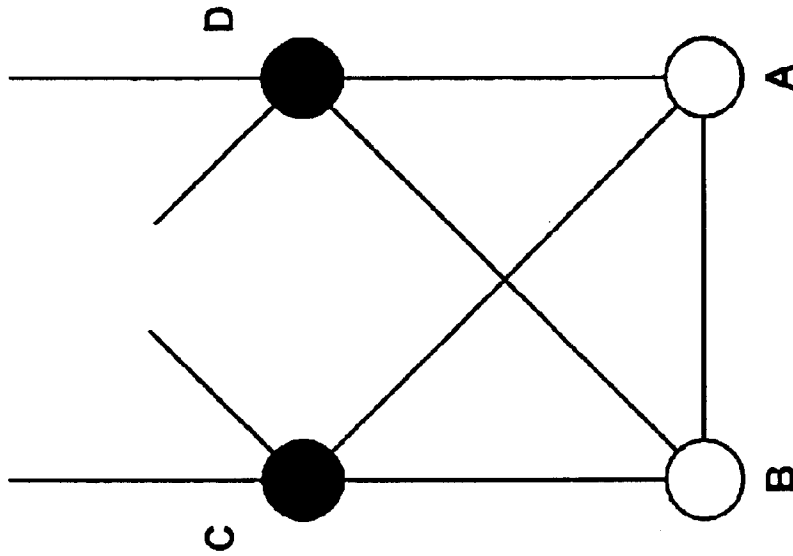


Fig. 5F

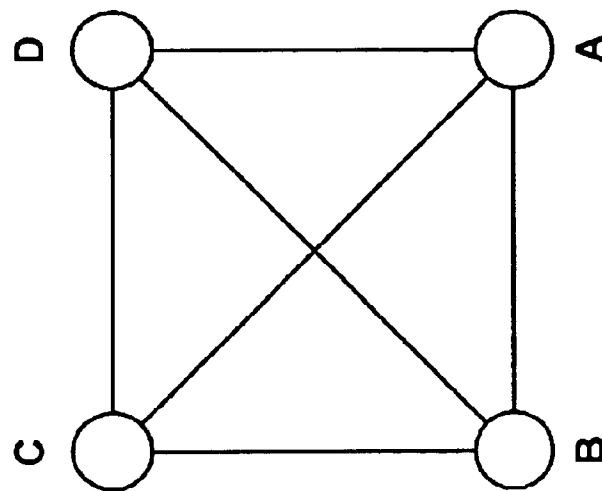


Fig. 5E

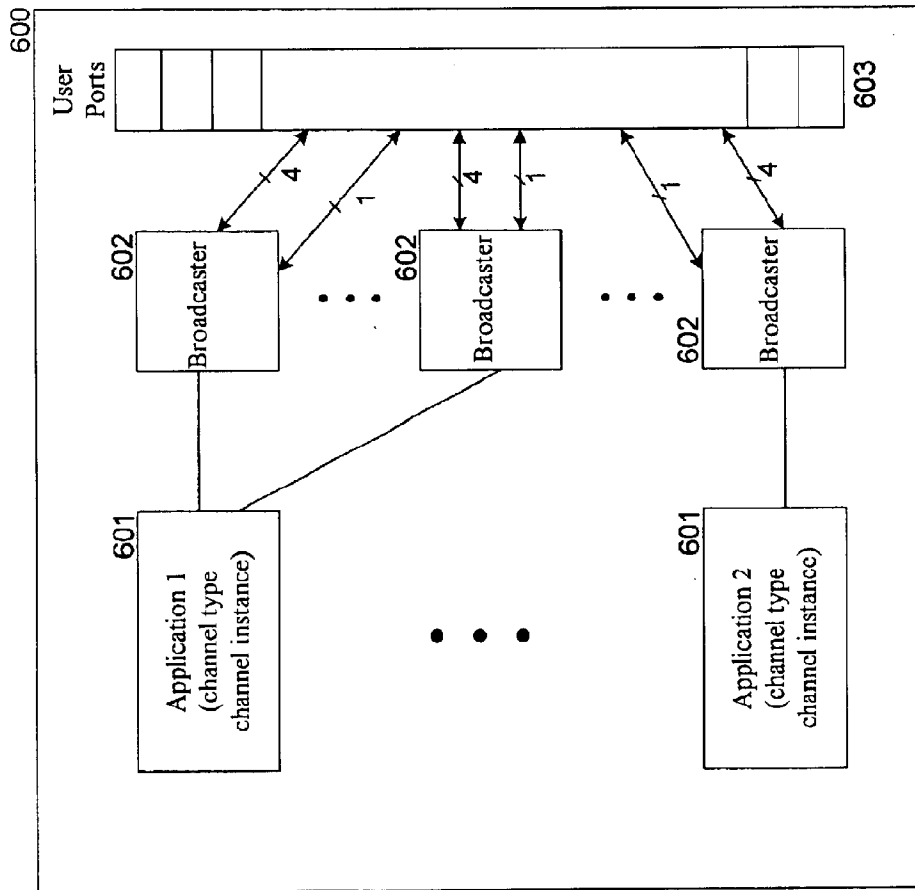


Fig. 6

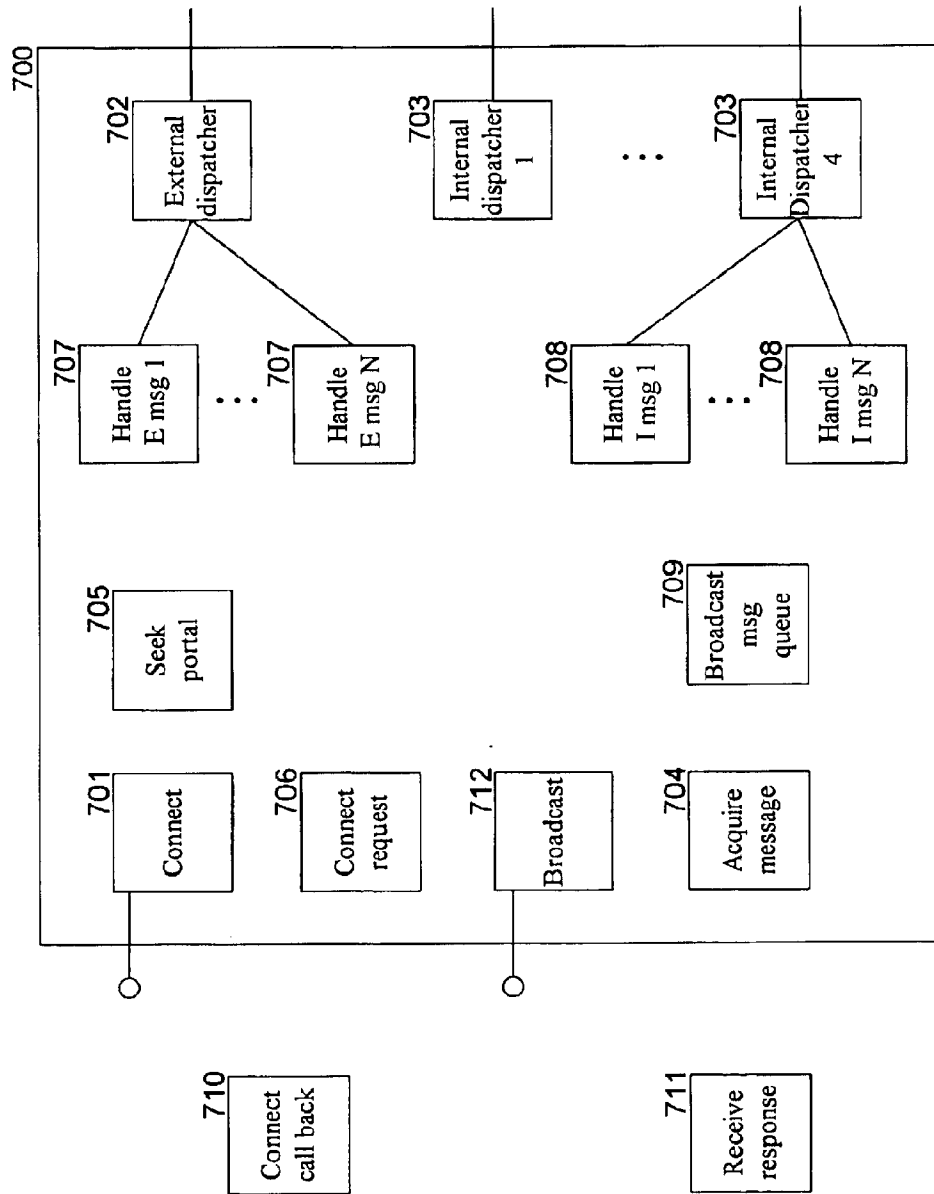
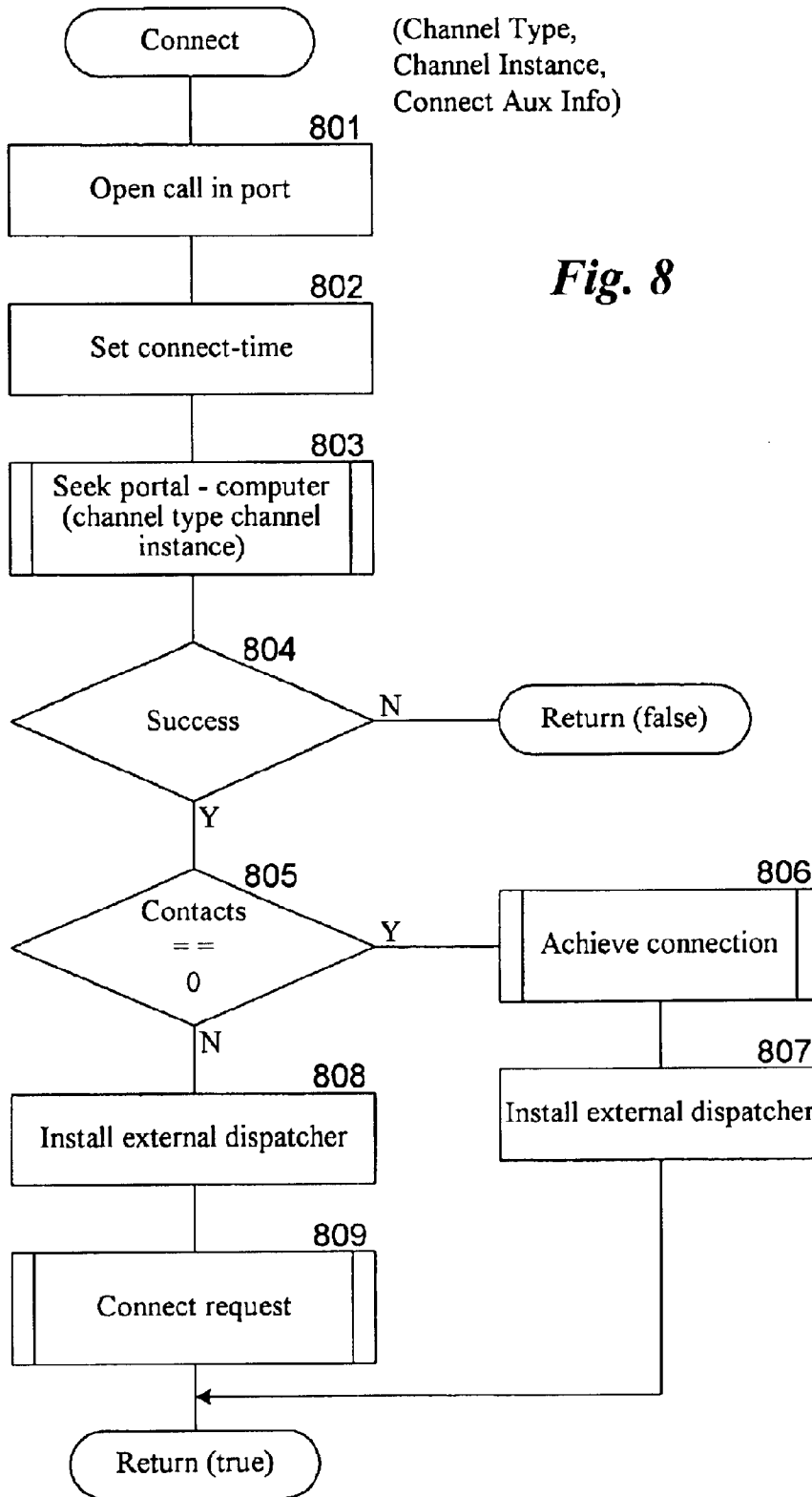
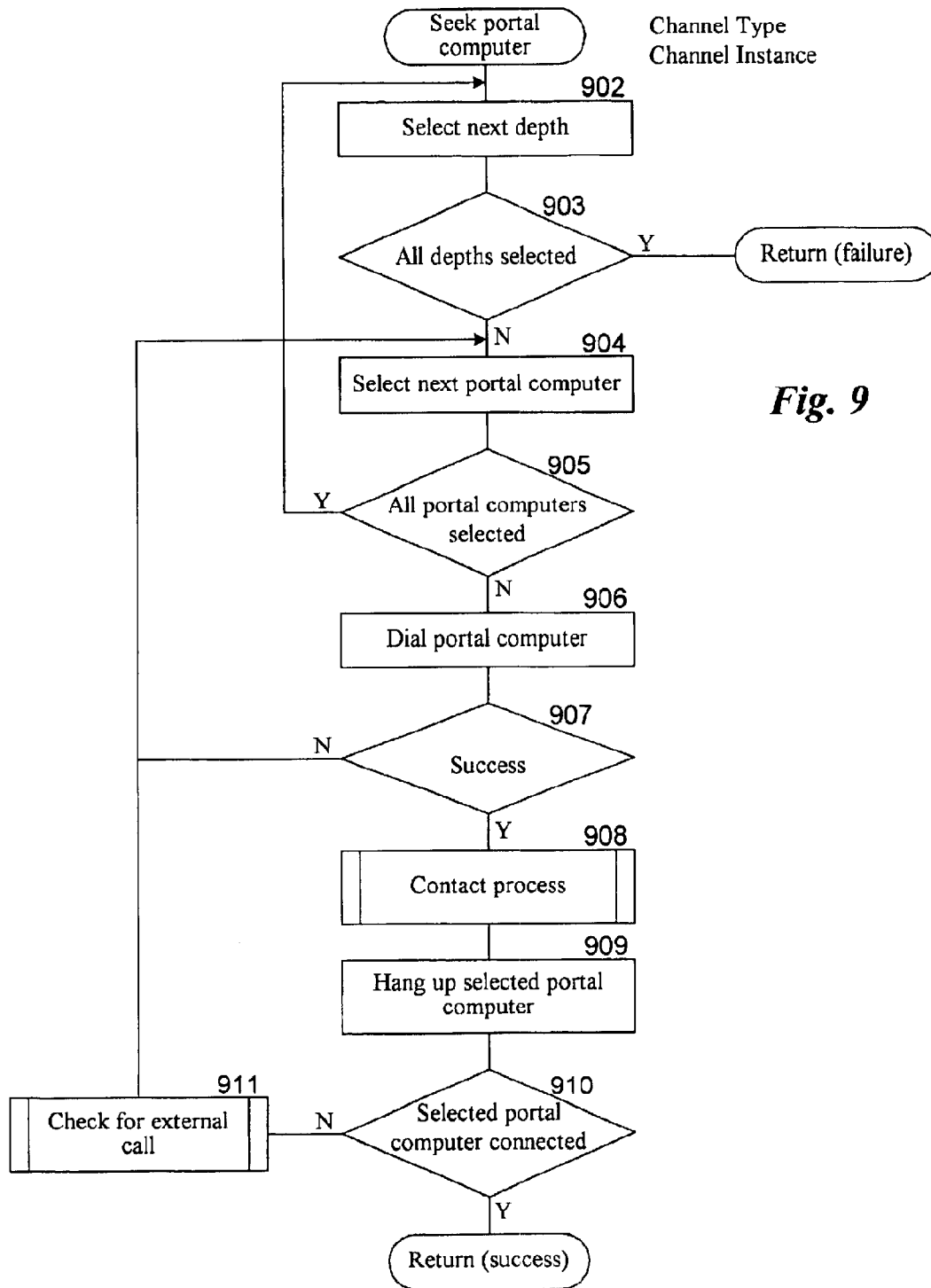


Fig. 7





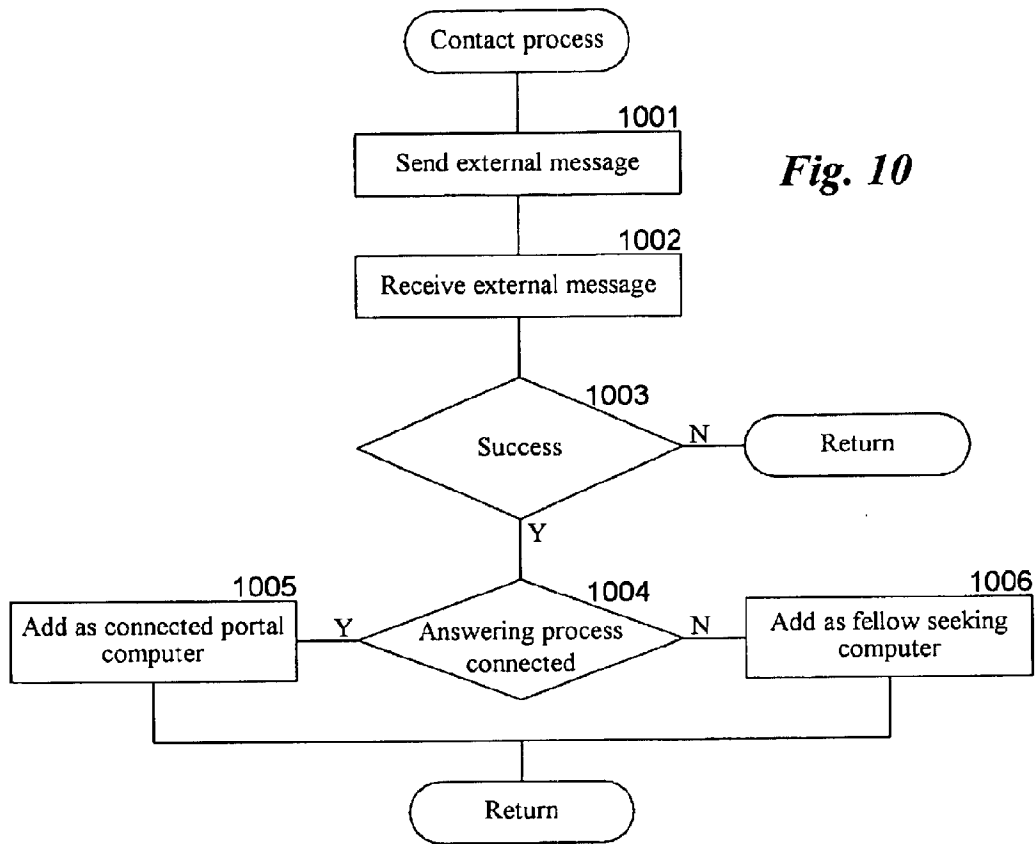


Fig. 11

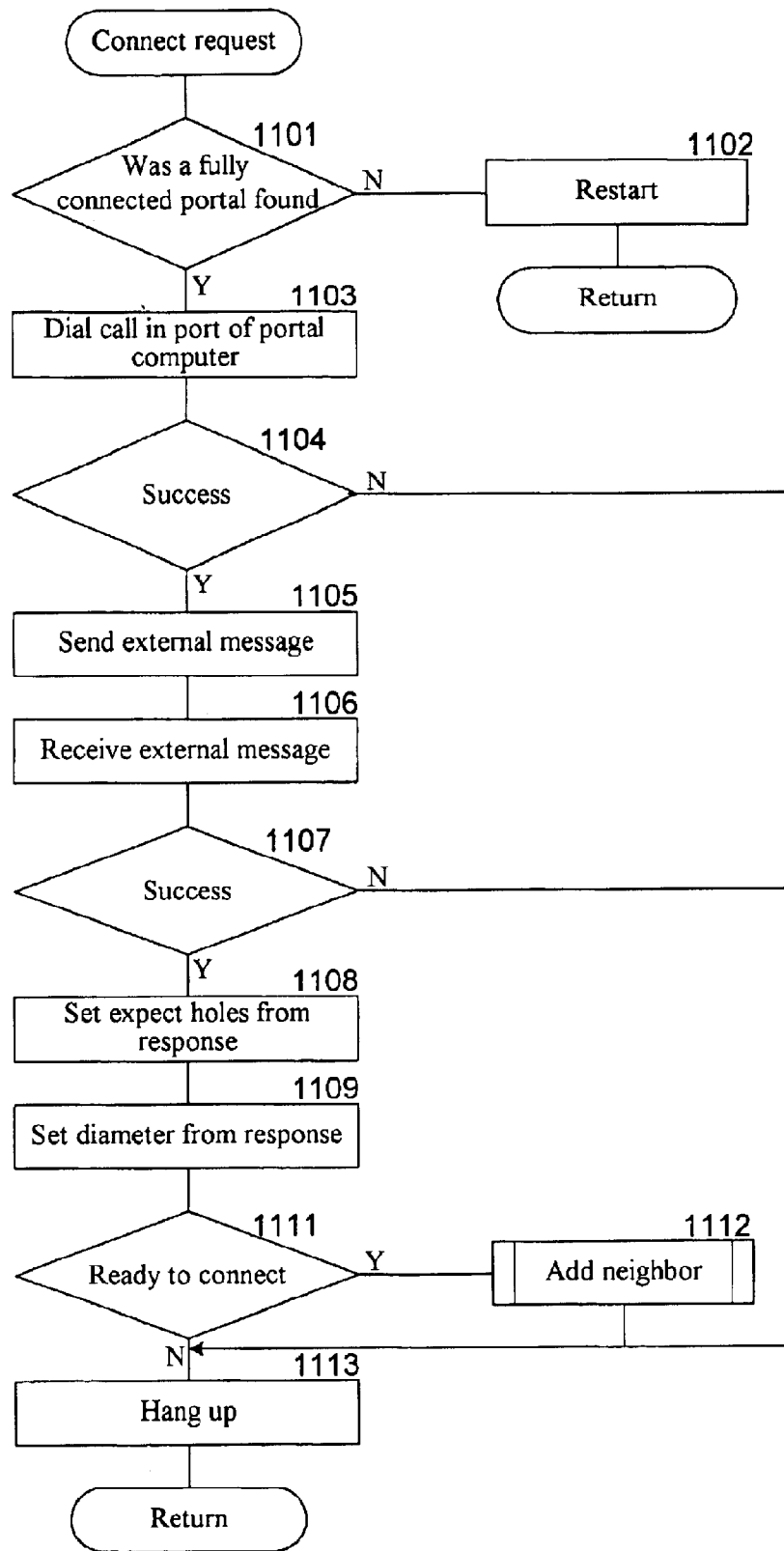
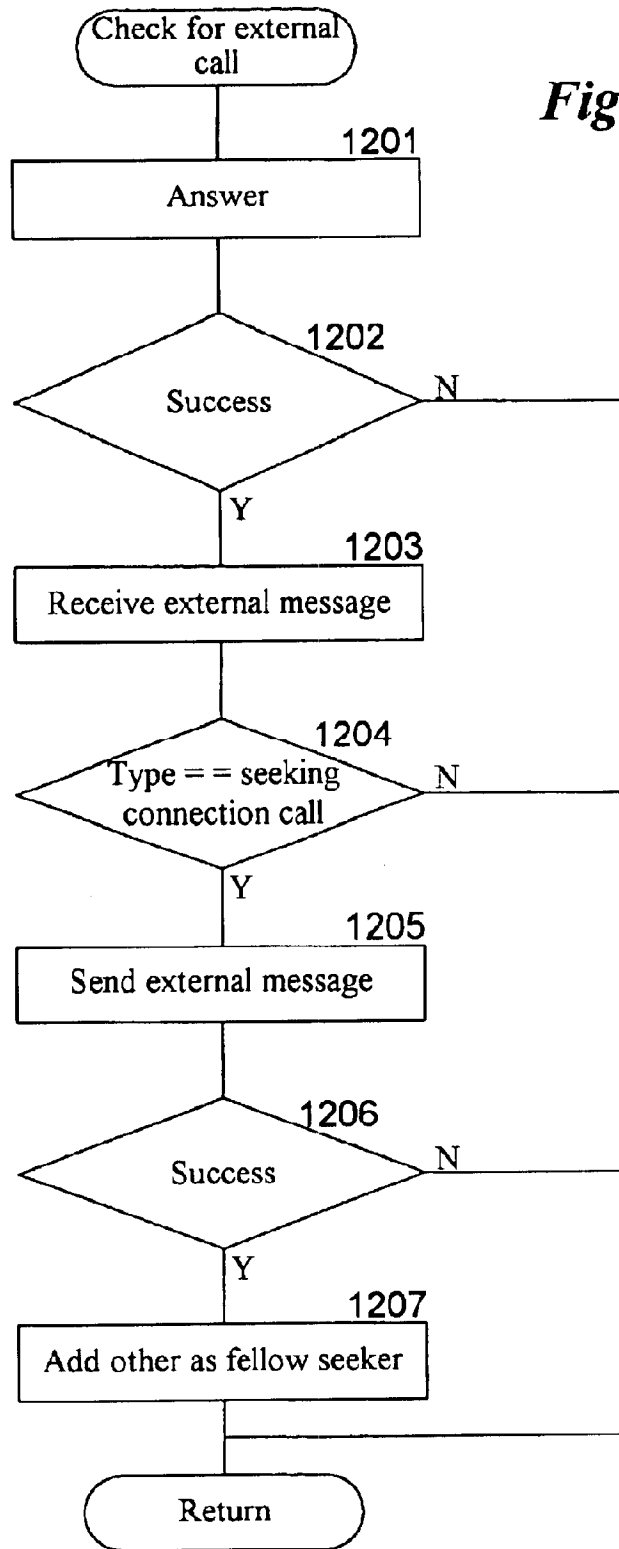


Fig. 12



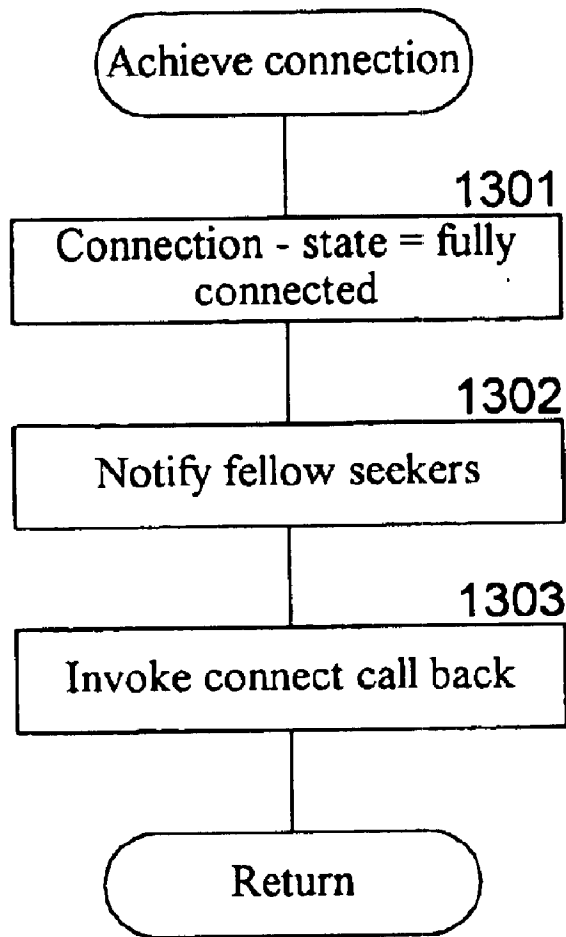


Fig. 13

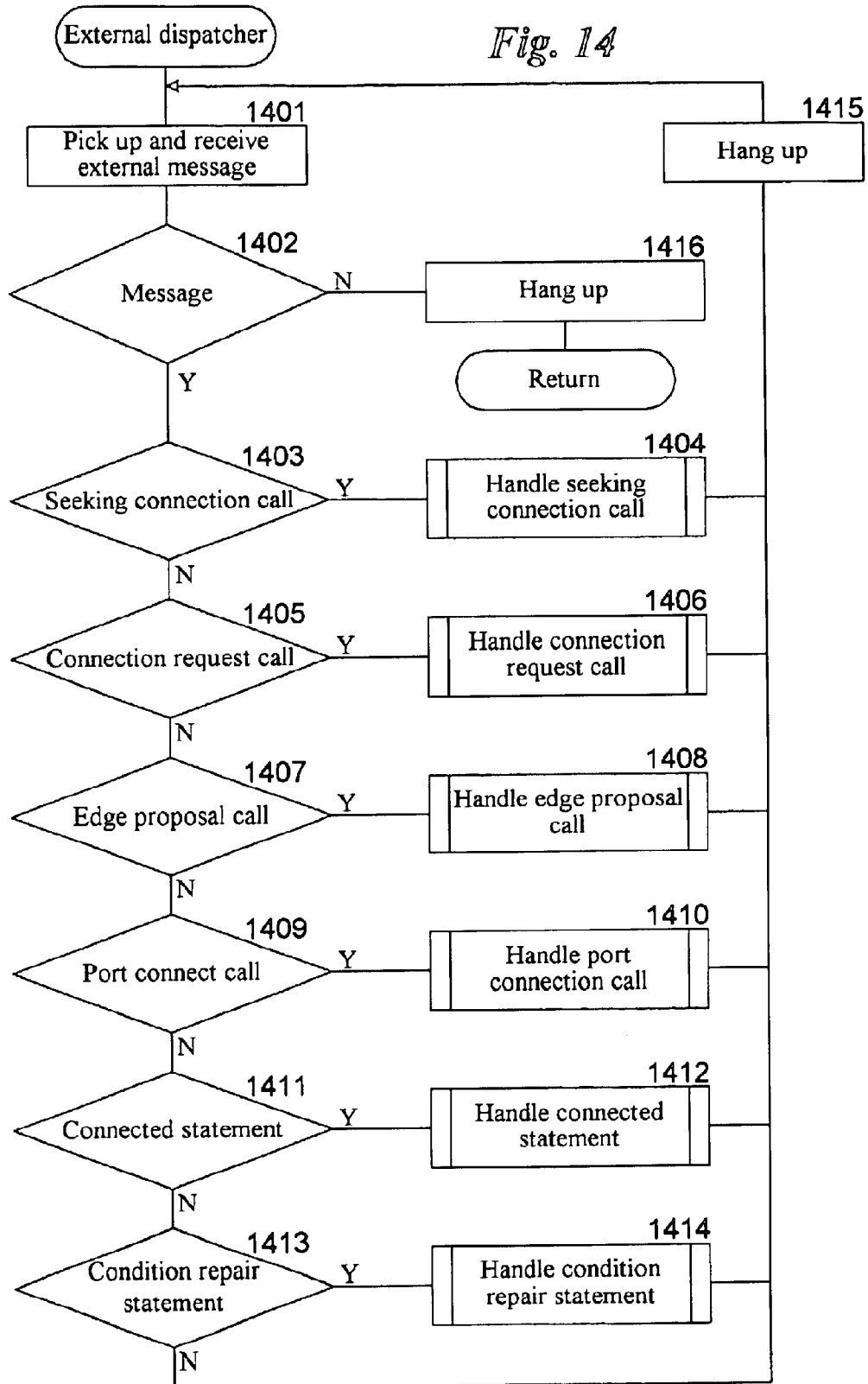
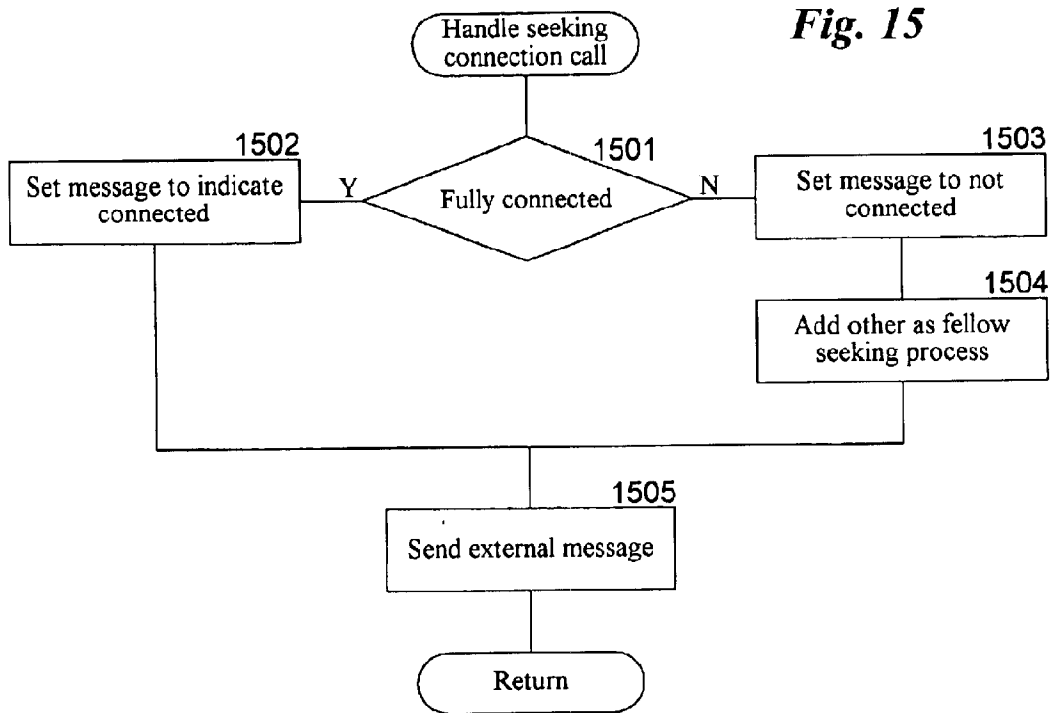


Fig. 15



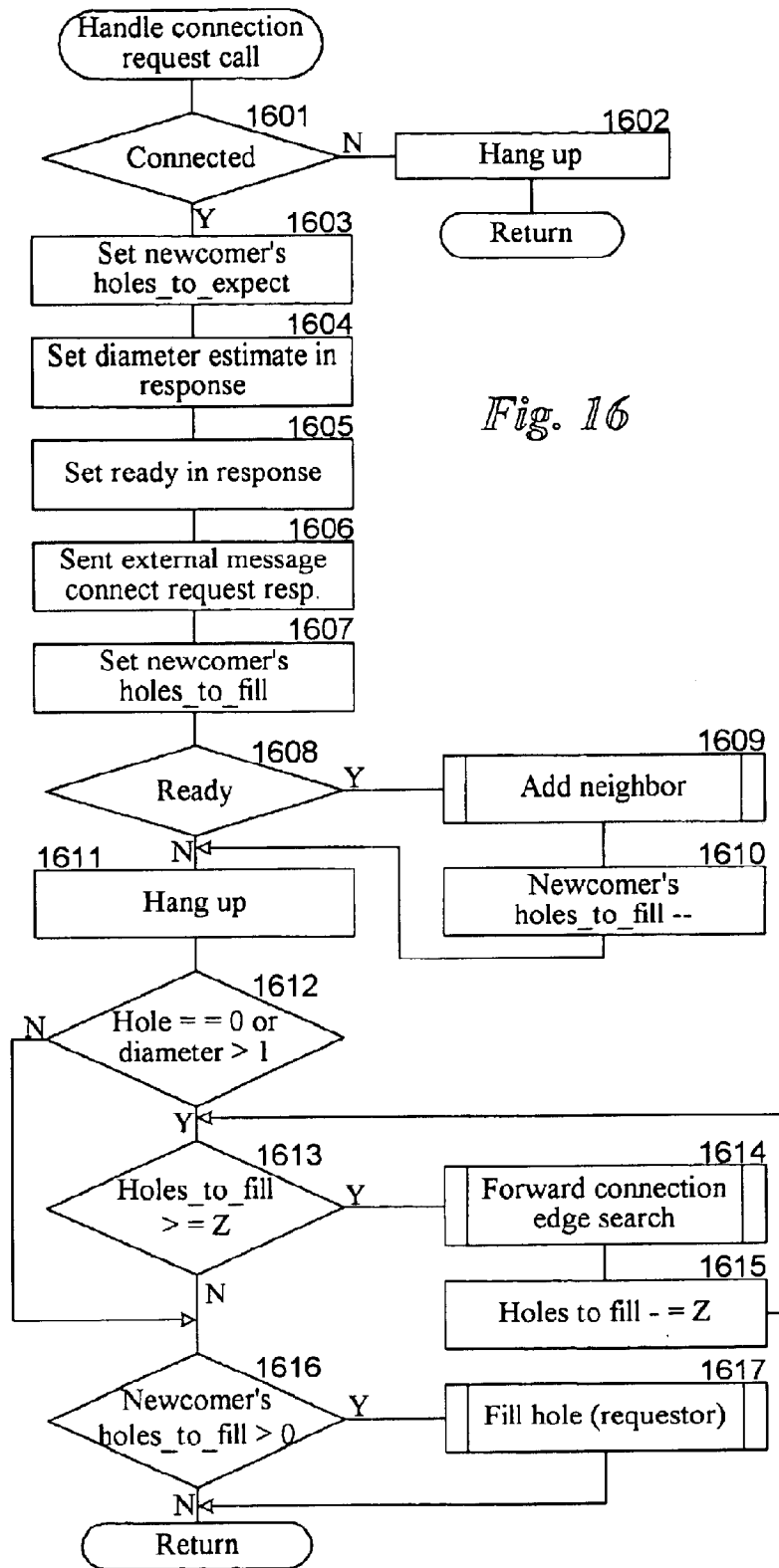


Fig. 16

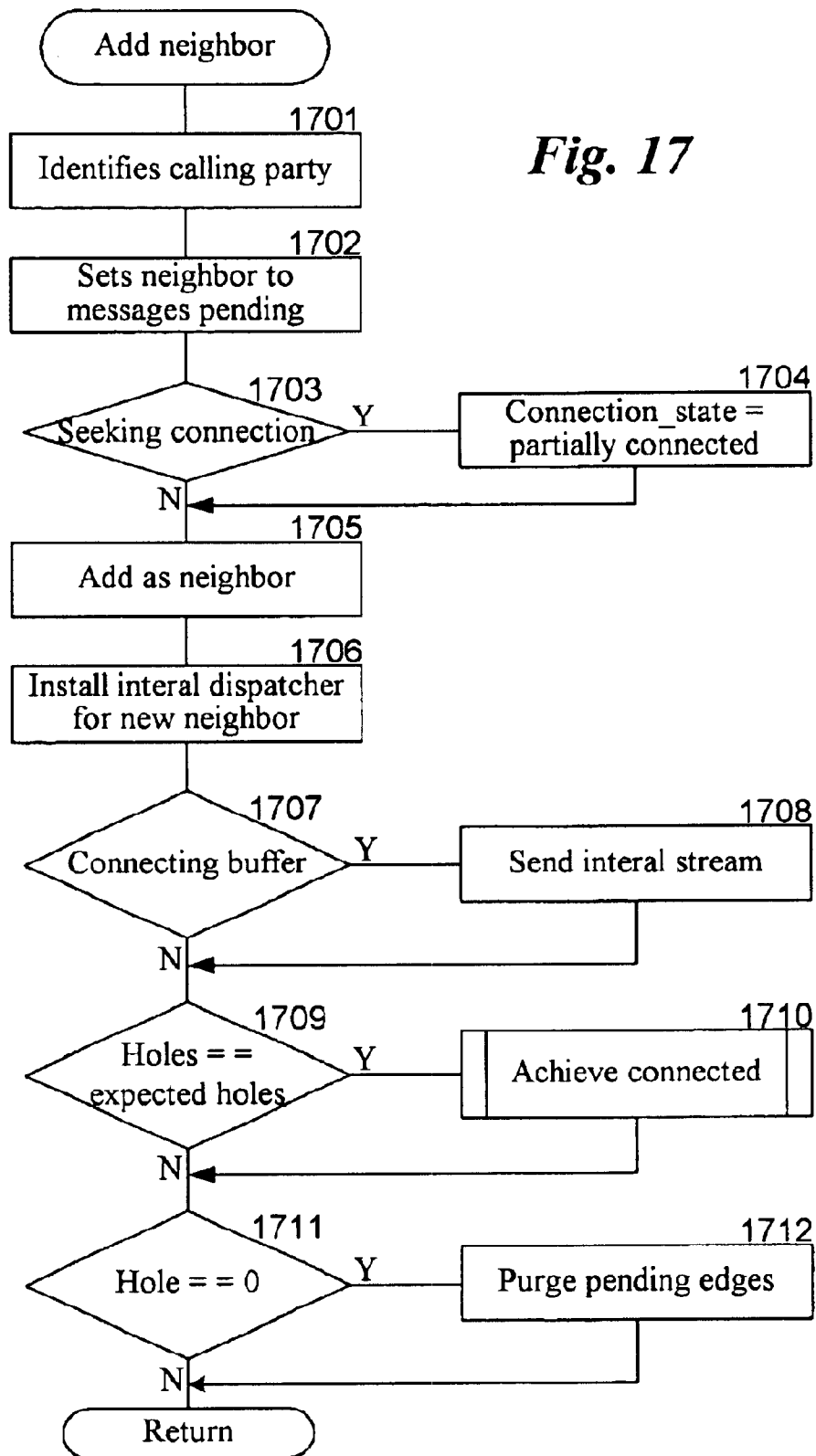
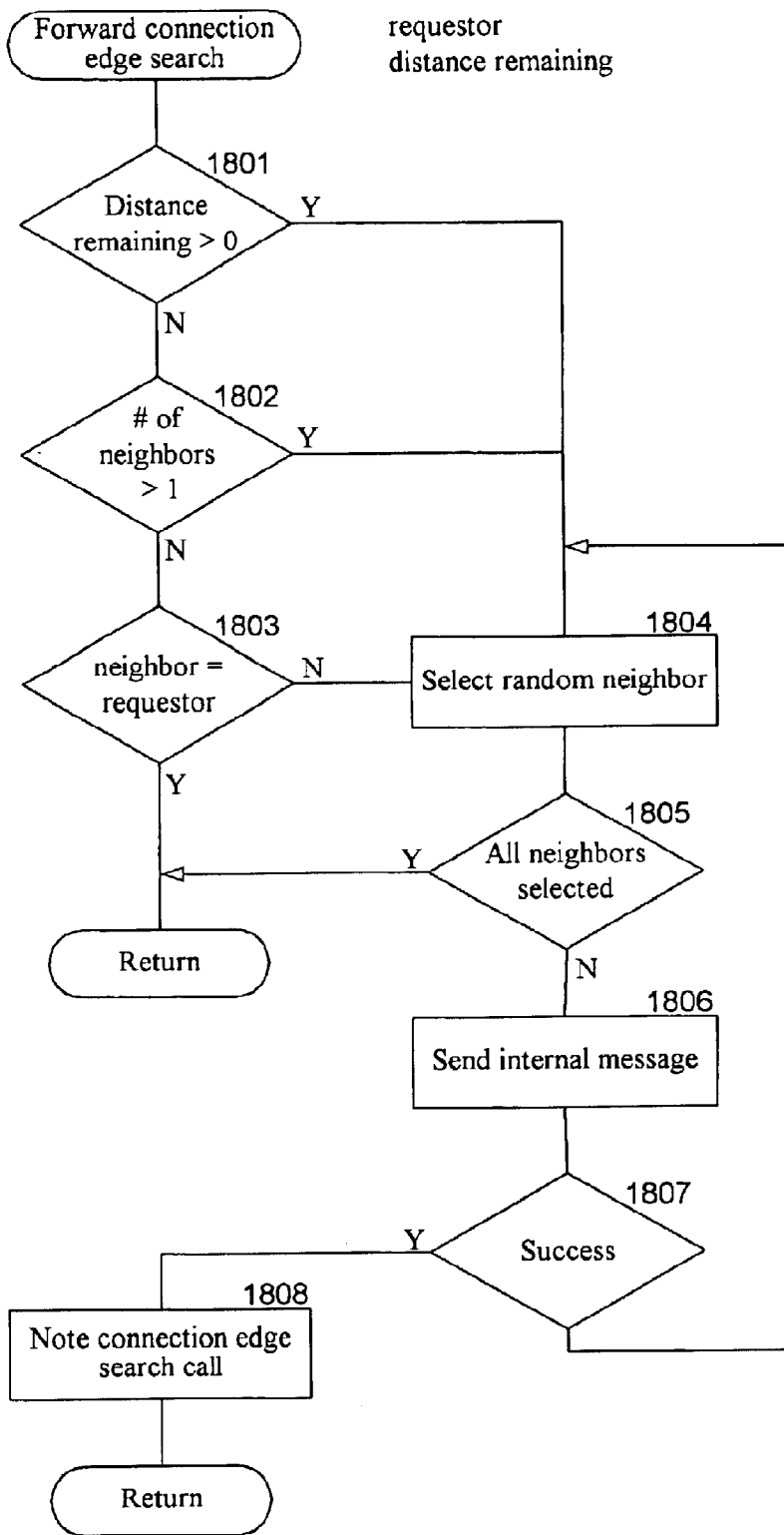
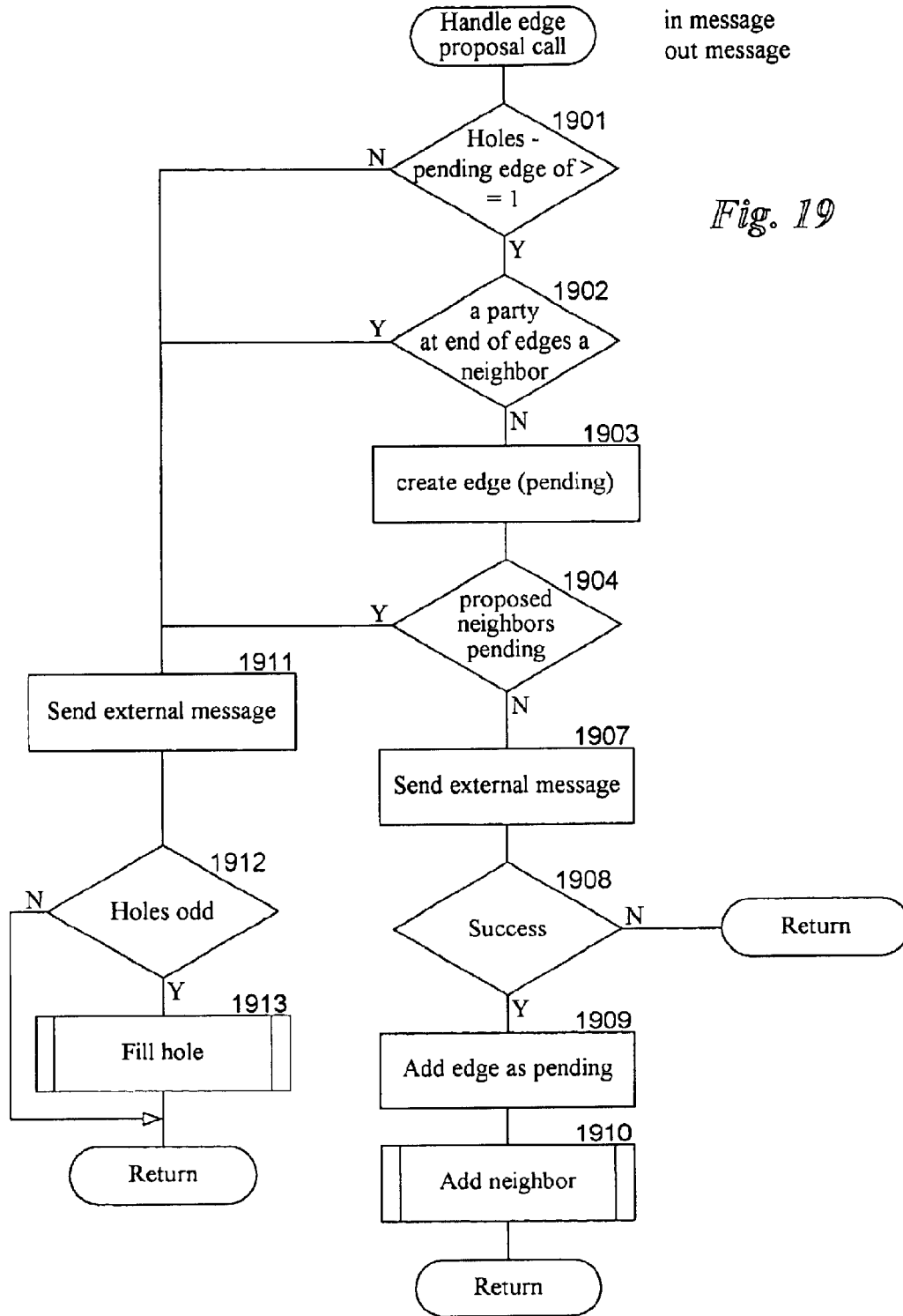


Fig. 18





in message
out message

Fig. 19

Fig. 20

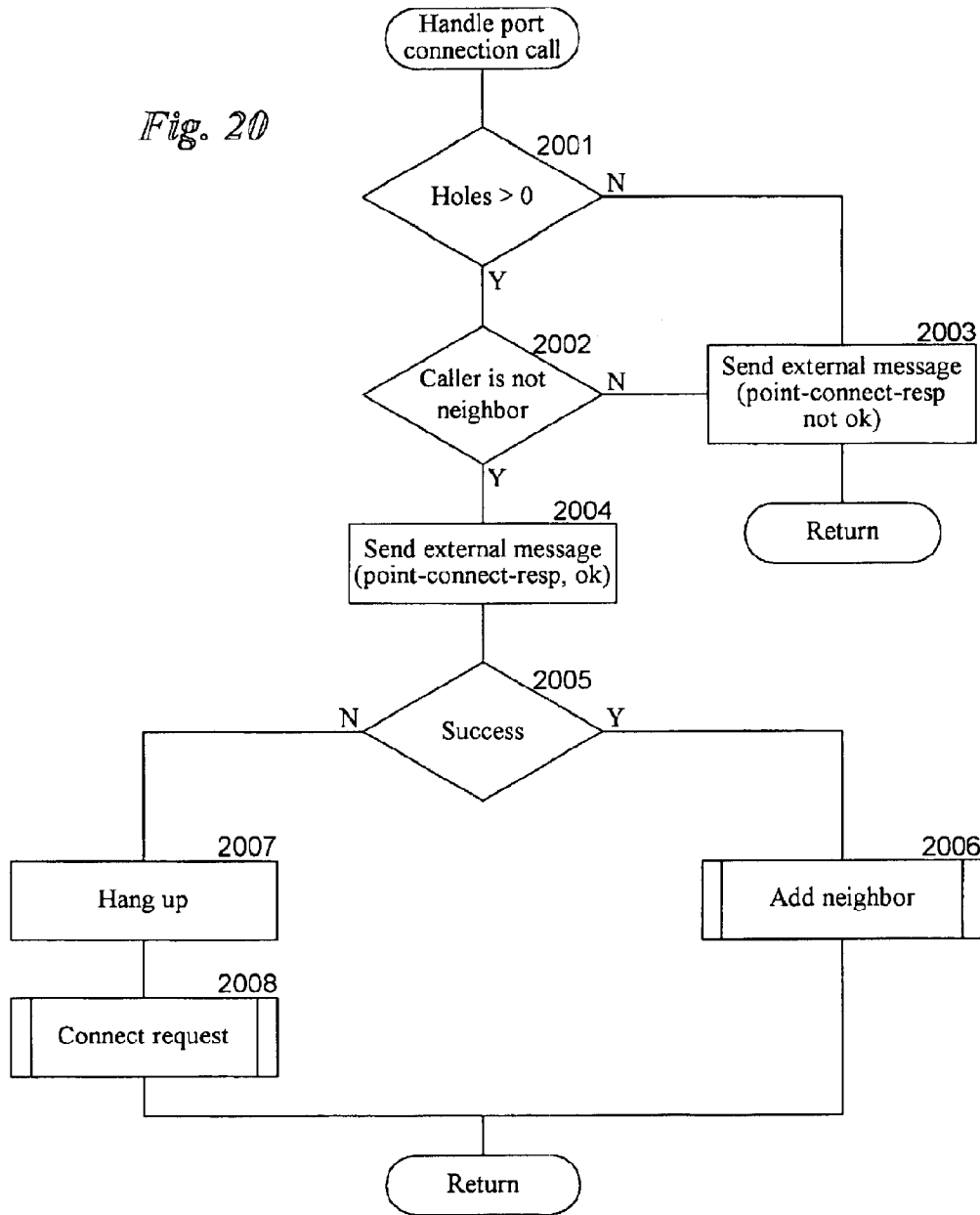


Fig. 21

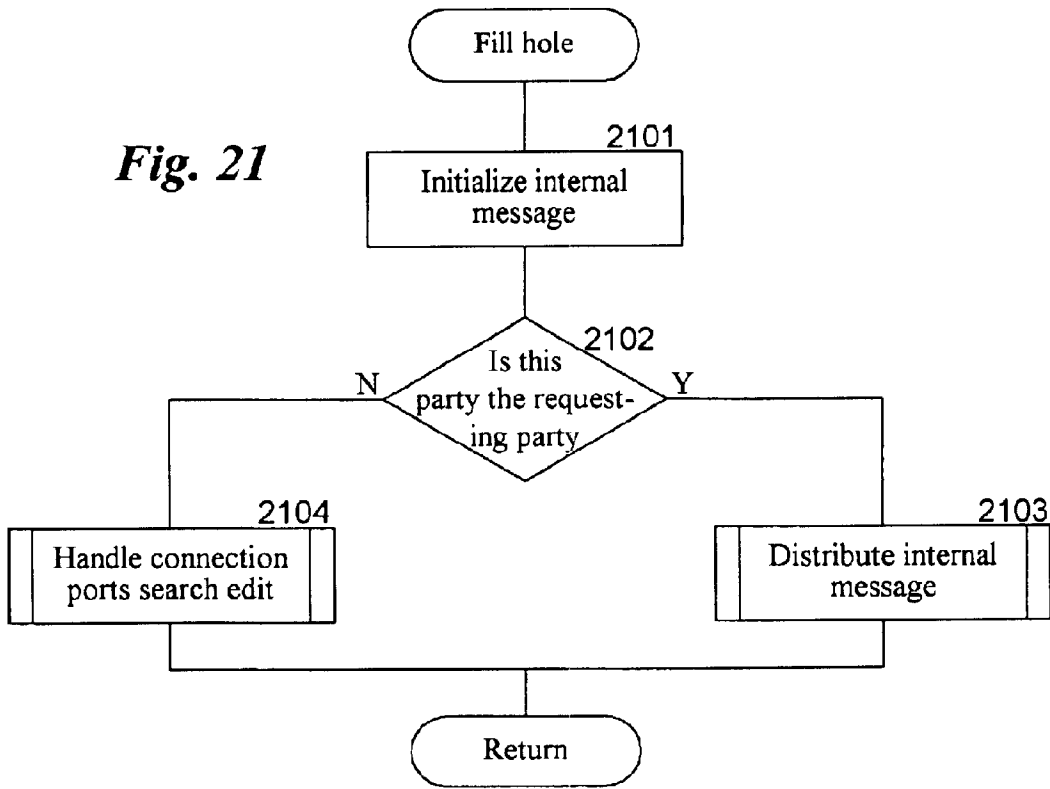


Fig. 22

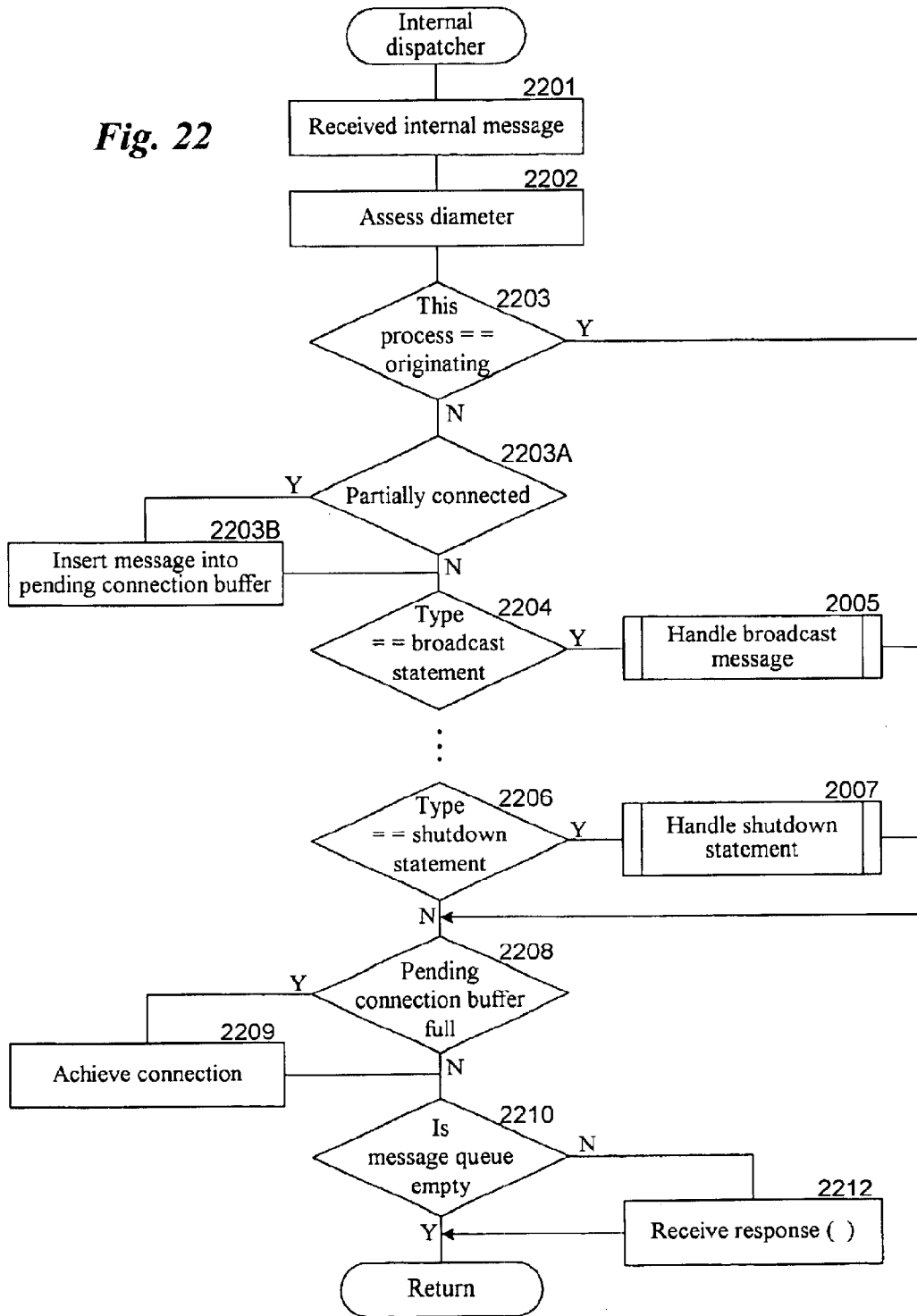


Fig. 23

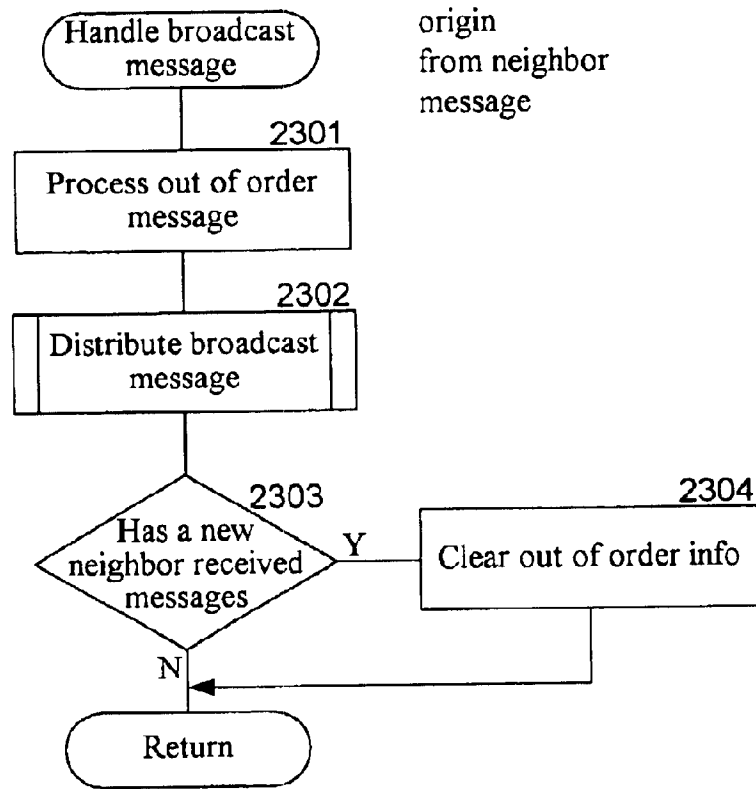
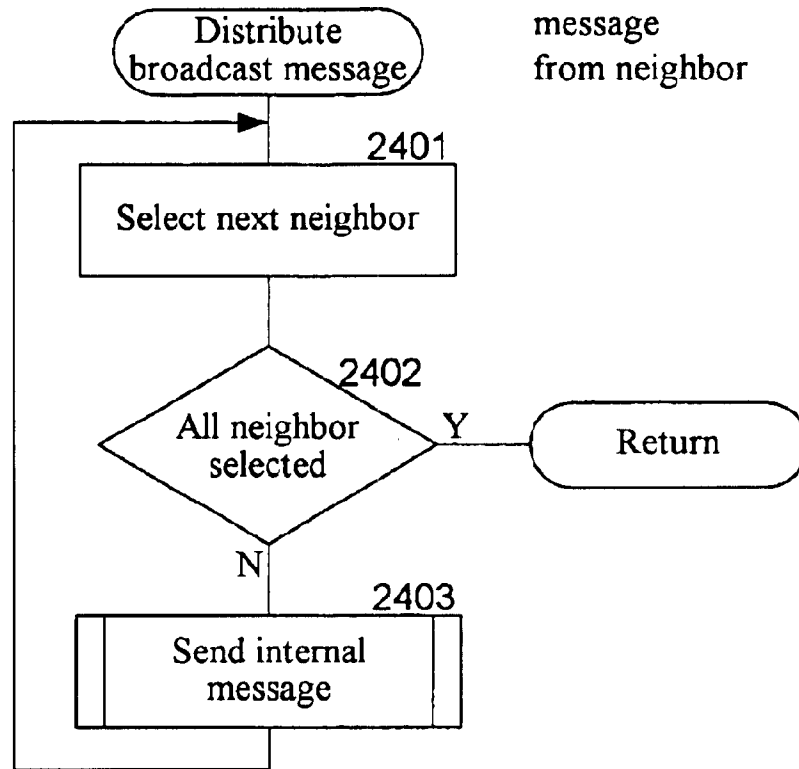


Fig. 24



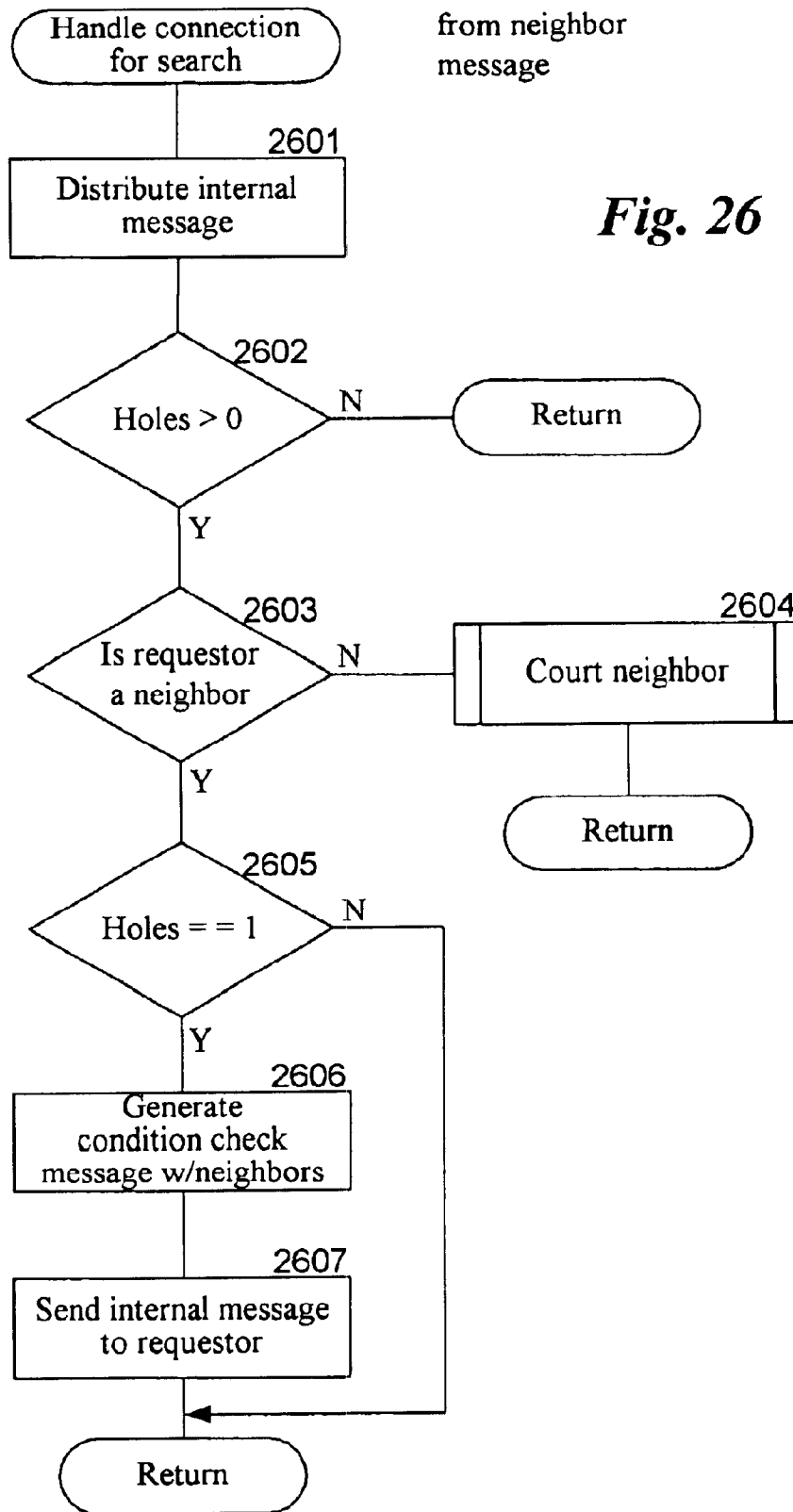
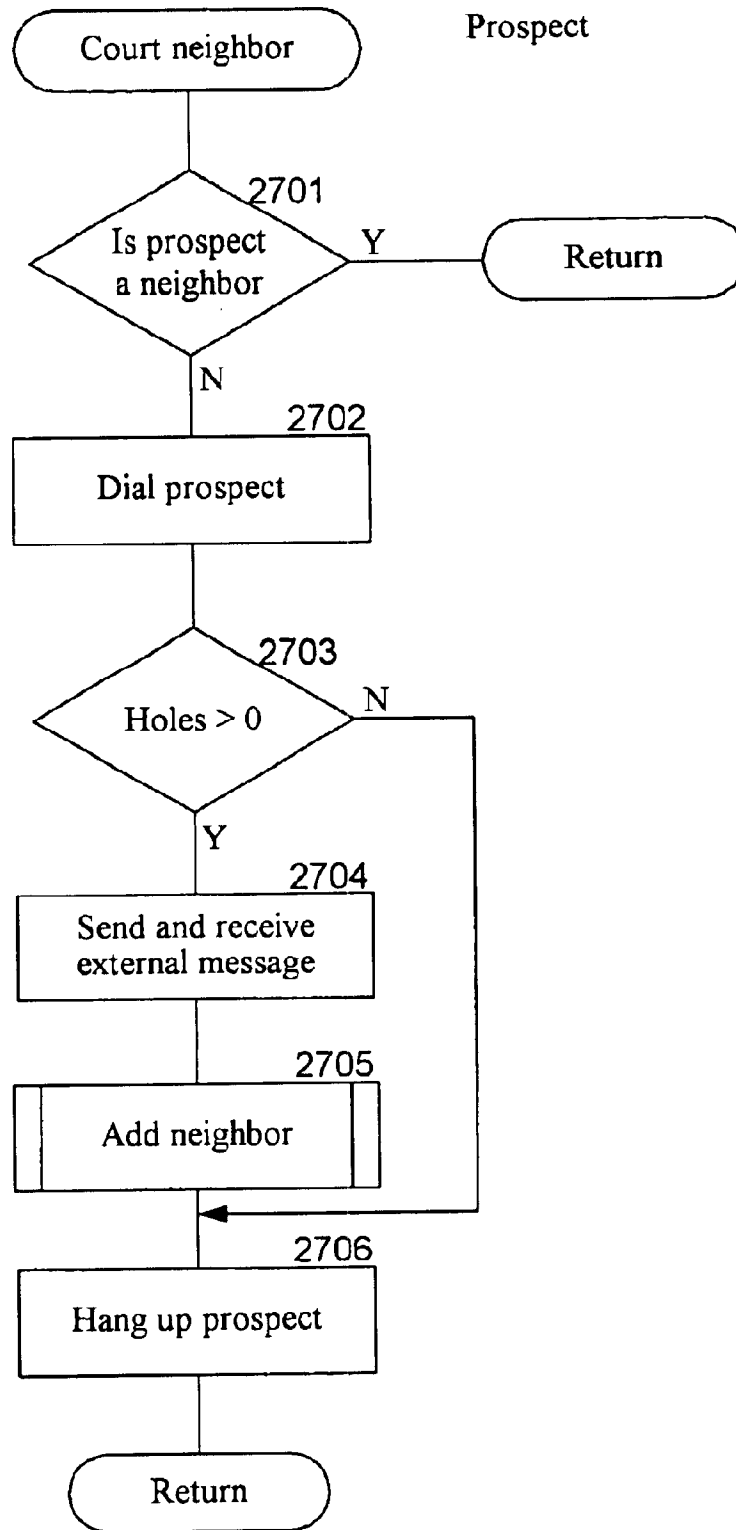


Fig. 27



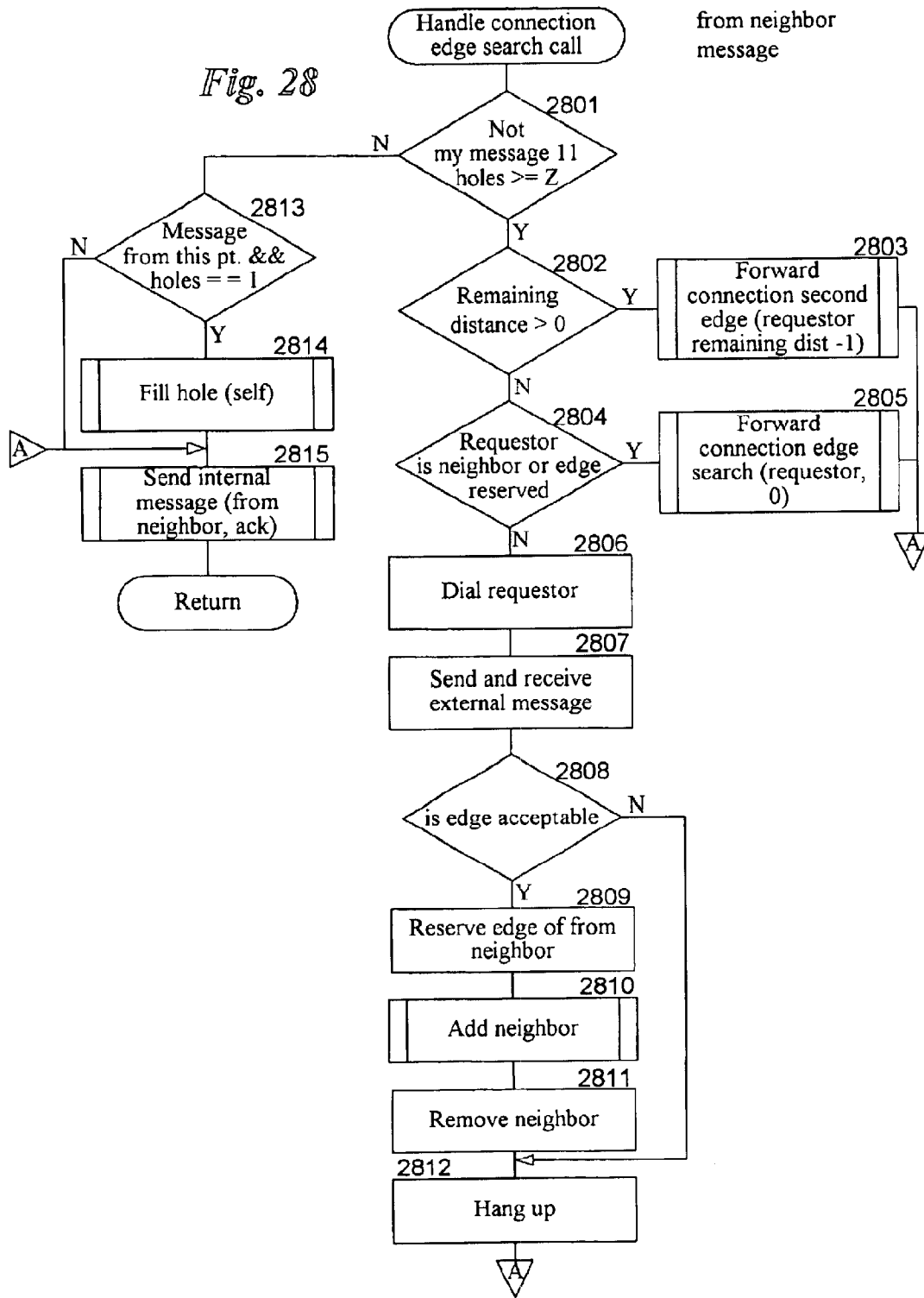


Fig. 29

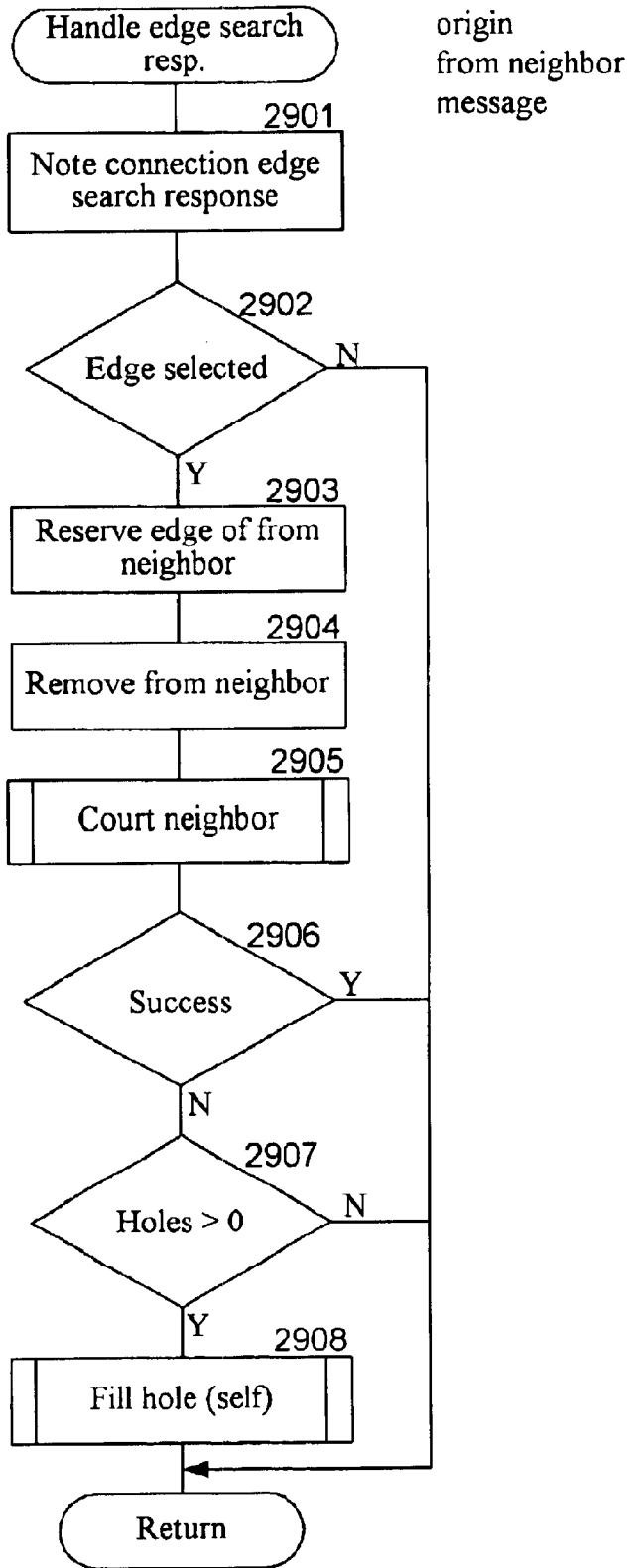


Fig. 30

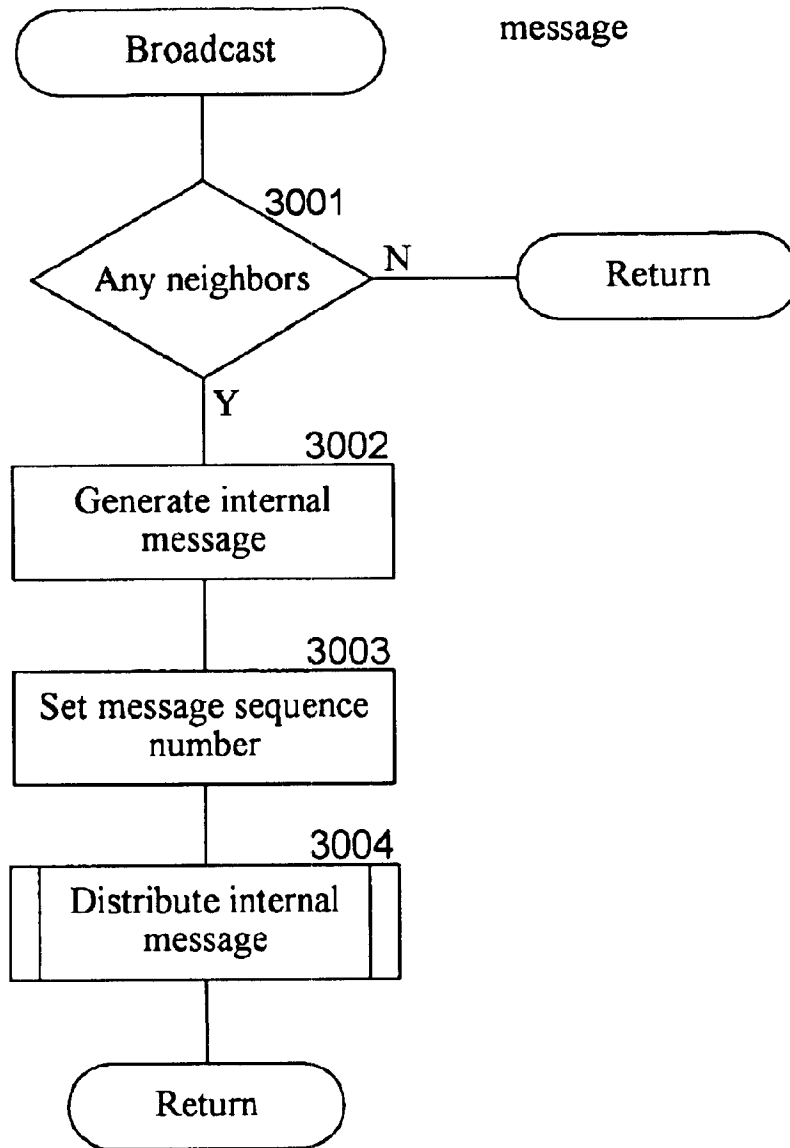


Fig. 31

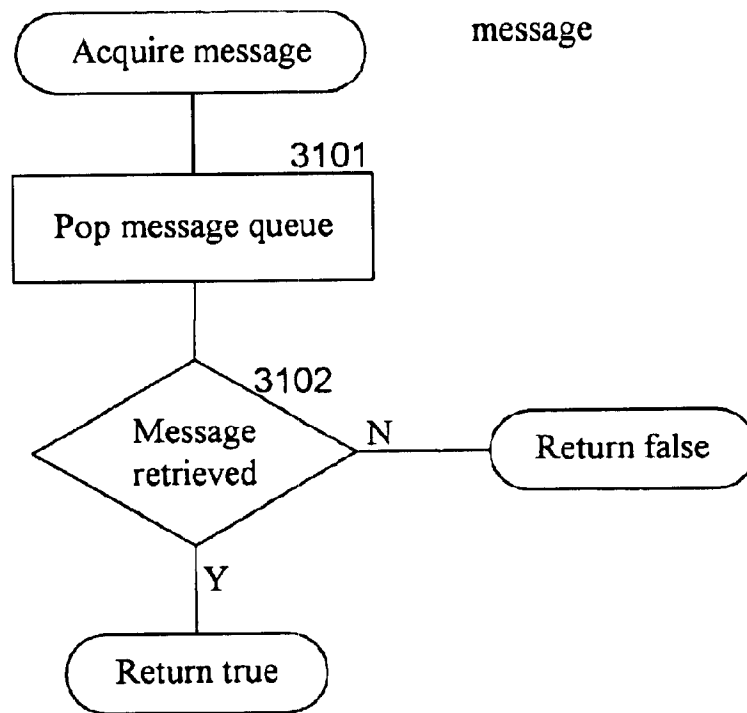


Fig. 32

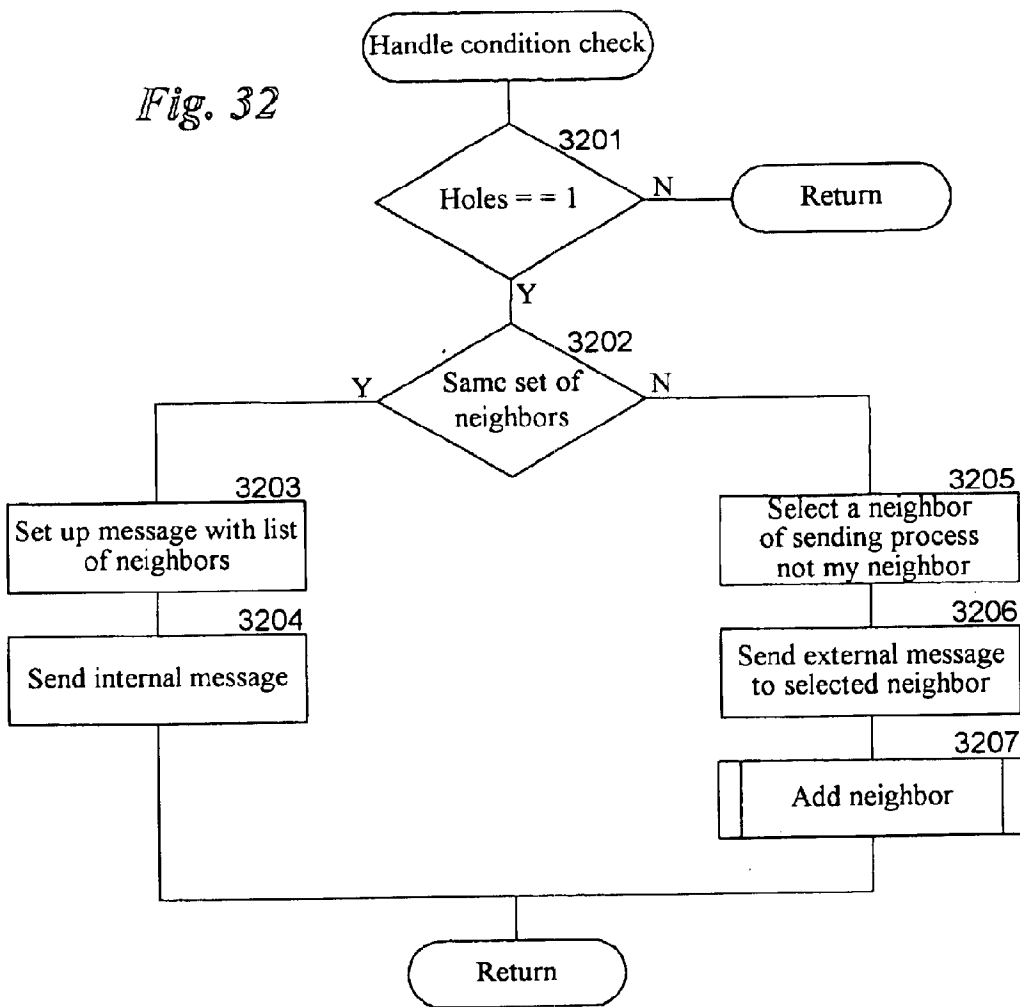


Fig. 33

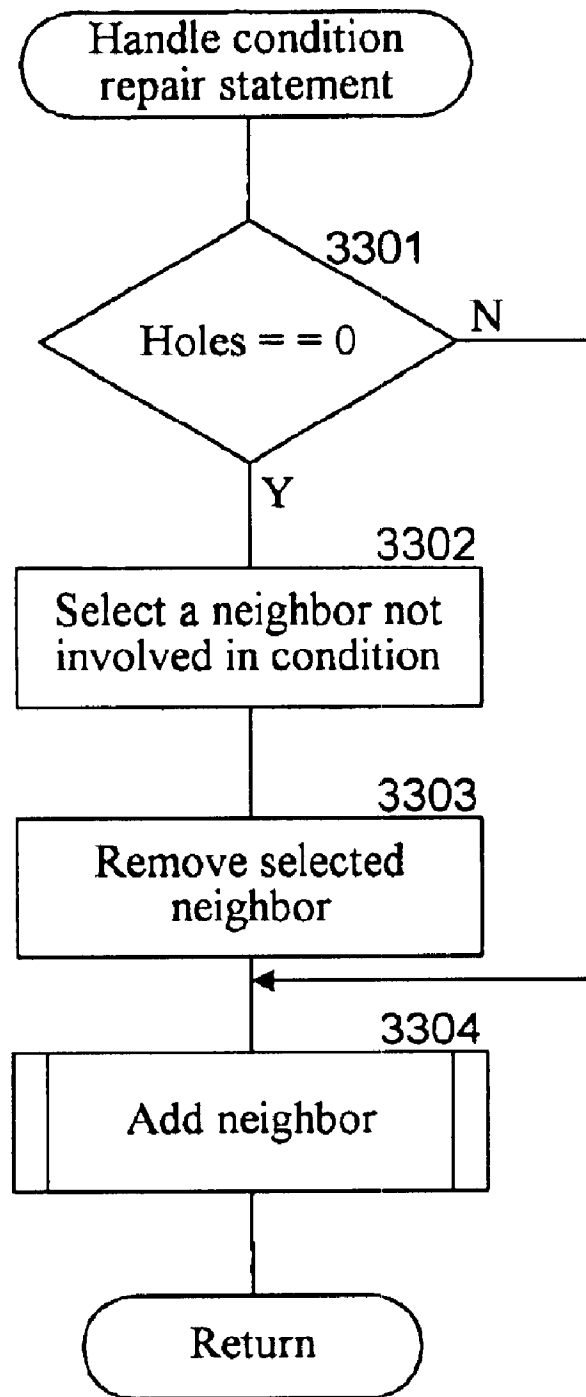
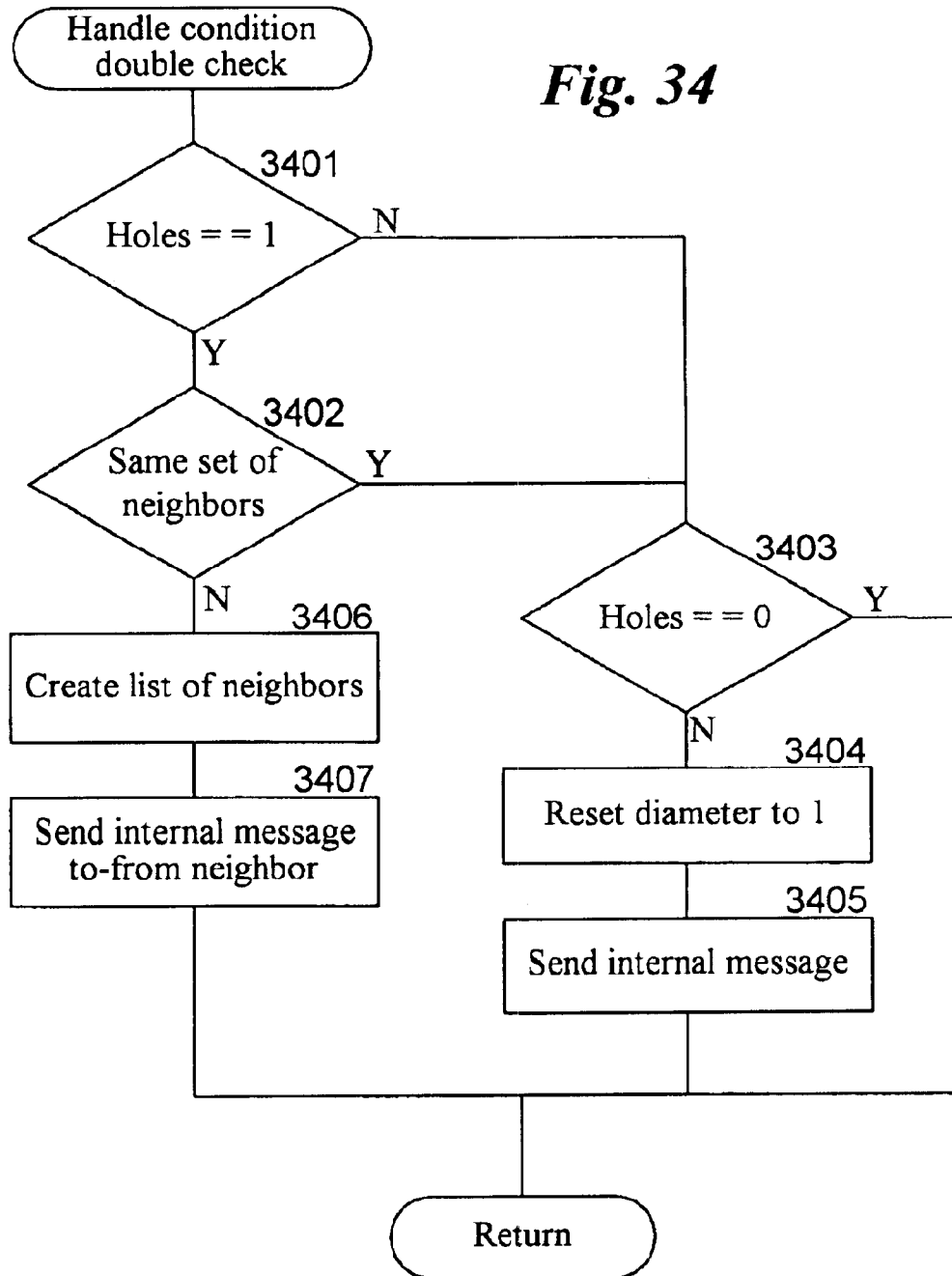


Fig. 34



US 6,714,966 B1

1

INFORMATION DELIVERY SERVICE**CROSS-REFERENCE TO RELATED APPLICATIONS**

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

TECHNICAL FIELD

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

BACKGROUND

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

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

The client/server middleware systems provide a server that coordinates the communications between the various clients who are sharing the information. The server functions as a central authority for controlling access to shared resources. Examples of client/server middleware systems include remote procedure calls ("RPC"), database servers, and the common object request broker architecture ("CORBA"). Client/server middleware systems are not particularly well suited to sharing of information among many participants. In particular, when a client stores information

2

to be shared at the server, each other client would need to poll the server to determine that new information is being shared. Such polling places a very high overhead on the communications network. Alternatively, each client may register a callback with the server, which the server then invokes when new information is available to be shared. Such a callback technique presents a performance bottleneck because a single server needs to call back to each client whenever new information is to be shared. In addition, the reliability of the entire sharing of information depends upon the reliability of the single server. Thus, a failure at a single computer (i.e., the server) would prevent communications between any of the clients.

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

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

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

BRIEF DESCRIPTION OF THE DRAWINGS

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

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

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

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

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

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

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

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

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

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

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

US 6,714,966 B1

3

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4

DETAILED DESCRIPTION

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

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

FIG. 1 illustrates a graph that is 4-regular and 4-connected which represents the broadcast channel. Each of the nine nodes A-I represents a computer that is connected to the broadcast channel, and each of the edges represents an "edge" connection between two computers of the broadcast channel. The time it takes to broadcast a message to each computer on the broadcast channel depends on the speed of the connections between the computers and the number of connections between the originating computer and each other computer on the broadcast channel. The minimum number of connections that a message would need to traverse between each pair of computers is the "distance" between the computers (i.e., the shortest path between the two nodes of the graph). For example, the distance between computers A and F is one because computer A is directly connected to computer F. The distance between computers A and B is two because there is no direct connection between computers A and B, but computer F is directly connected to computer B. Thus, a message originating at computer A would be sent directly to computer F, and then sent from

US 6,714,966 B1

5

computer F to computer B. The maximum of the distances between the computers is the “diameter” of broadcast channel. The diameter of the broadcast channel represented by FIG. 1 is two. That is, a message sent by any computer would traverse no more than two connections to reach every other computer. FIG. 2 illustrates a graph representing 20 computers connected to a broadcast channel. The diameter of this broadcast channel is 4. In particular, the shortest path between computers 1 and 3 contains four connections (1–12, 12–15, 15–18, and 18–3).

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

Composing the Graph

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

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

6

pairs is broken, and a connection between computer Z and each of computers B, C, D, and E is established as indicated by FIG. 3B. The process of breaking the connection between two neighbors and reconnecting each of the former neighbors to another computer is referred to as “edge pinning” as the edge between two nodes may be considered to be stretched and pinned to a new node.

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

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

A seeking computer could connect to the broadcast channel by connecting to computers either directly connected to the found portal computer or directly connected to one of its neighbors. A possible problem with such a scheme for identifying the neighbors for the seeking computer is that the diameter of the broadcast channel may increase when each seeking computer uses the same found portal computer and

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

Broadcasting Through the Graph

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

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

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

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

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

Decomposing the Graph

A connected computer disconnects from the broadcast channel either in a planned or unplanned manner. When a computer disconnects in a planned manner, it sends a disconnect message to each of its four neighbors. The

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

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

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

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

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

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

FIGS. 5E and 5F further illustrate the neighbors with empty ports condition. FIG. 5E illustrates the neighbors with empty ports condition in the small regime. In this example, if computer E disconnected in an unplanned manner, then

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

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

Port Selection

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

to be connected to multiple broadcast channels that are uniquely identified by channel type and channel instance. The algorithm may be "seeded" with channel type and channel instance in order to generate a unique ordering of port numbers for each broadcast channel. Thus, a seeking computer will dial the ports of a portal computer in the same order as the portal computer used when allocating its call-in port.

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

Locating a Portal Computer

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

When a seeking computer locates a portal computer that is itself not fully connected, the two computers do not connect when they first locate each other because the broadcast channel may already be established and accessible through a higher-ordered port number on another portal

US 6,714,966 B1

13

computer. If the two seeking computers were to connect to each other, then two disjoint broadcast channels would be formed. Each seeking computer can share its experience in trying to locate a portal computer with the other seeking computer. In particular, if one seeking computer has searched all the portal computers to a depth of eight, then the one seeking computer can share that it has searched to a depth of eight with another seeking computer. If that other seeking computer has searched to a depth of, for example, only four, it can skip searching through depths five through eight and that other seeking computer can advance its searching to a depth of nine.

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

Identifying Neighbors for a Seeking Computer

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

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

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

14

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

External Data Representation

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

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

M-Regular

In the embodiment described above, each fully connected computer has four internal connections. The broadcast technique can be used with other numbers of internal connections. For example, each computer could have 6, 8, or any even number of internal connections. As the number of internal connections increase, the diameter of the broadcast channel tends to decrease, and thus propagation time for a message tends to decrease. The time that it takes to connect a seeking computer to the broadcast channel may, however, increase as the number of internal connections increases. When the number of internal connectors is even, then the broadcast channel can be maintained as m-regular and m-connected (in the steady state). If the number of internal connections is odd, then when the broadcast channel has an odd number of computers connected, one of the computers will have less than that odd number of internal connections. In such a situation, the broadcast network is neither m-regular nor m-connected. When the next computer connects to the broadcast channel, it can again become

m-regular and m-connected. Thus, with an odd number of internal connections, the broadcast channel toggles between being and not being m-regular and m-connected.

Components

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

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

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

FIG. 7 is a block diagram illustrating the sub-components of the broadcaster component in one embodiment. The broadcaster component includes a connect component 701, an external dispatcher 702, an internal dispatcher 703 for

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

An Information Delivery Service

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

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

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

A user may subscribe to a broadcast channel for an individual topic, which corresponds to a leaf node in the hierarchy, or a user may subscribe to a category of topics,

which corresponds to a non-leaf node in the hierarchy. For example, a user may subscribe to a category of sports scores or subscribe to the topic of NFL scores. In one embodiment, each topic would have its own broadcast channel. As a result, the subscribing to a category of topics would mean subscribing to multiple broadcast channels. Alternatively, a category of topics may have a single broadcast channel. If a user subscribes to just one topic in the category, the information delivery service application program executing at the subscriber's computer would simply disregard messages not related to the topic.

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

The following tables list messages sent by the broadcaster components.

Flow Diagrams

FIGS. 8-34 are flow diagrams illustrating the processing of the broadcaster component in one embodiment. FIG. 8 is a flow diagram illustrating the processing of the connect routine in one embodiment. This routine is passed a channel type (e.g., application name) and channel instance (e.g., session identifier), that identifies the broadcast channel to which this process wants to connect. The routine is also passed auxiliary information that includes the list of portal computers and a connection callback routine. When the connection is established, the connection callback routine is invoked to notify the application program. When this process invokes this routine, it is in the seeking connection state. When a portal computer is located that is connected and this routine connects to at least one neighbor, this process enters the partially connected state, and when the process eventually connects to four neighbors, it enters the fully connected state. When in the small regime, a fully connected process may have less than four neighbors. In block 801, the routine opens the call-in port through which the process is to communicate with other processes when establishing external and internal connections. The port is selected as the first available port using the hashing algorithm described above. In block 802, the routine sets the connect time to the current time. The connect time is used to identify the instance of the process that is connected through this external port. One process may connect to a broadcast channel of a certain channel type and channel instance using

Message Type	Description
<u>EXTERNAL MESSAGES</u>	
seeking_connection_call	Indicates that a seeking process would like to know whether the receiving process is fully connected to the broadcast channel
connection_request_call	Indicates that the sending process would like the receiving process to initiate a connection of the sending process to the broadcast channel
edge_proposal_call	Indicates that the sending process is proposing an edge through which the receiving process can connect to the broadcast channel (i.e., edge pinning)
port_connection_call	Indicates that the sending process is proposing a port through which the receiving process can connect to the broadcast channel
connected_stmt	Indicates that the sending process is connected to the broadcast channel
condition_repair_stmt	Indicates that the receiving process should disconnect from one of its neighbors and connect to one of the processes involved in the neighbors with empty port condition
<u>INTERNAL MESSAGES</u>	
broadcast_stmt	Indicates a message that is being broadcast through the broadcast channel for the application programs
connection_port_search_stmt	Indicates that the designated process is looking for a port through which it can connect to the broadcast channel
connection_edge_search_call	Indicates that the requesting process is looking for an edge through which it can connect to the broadcast channel
connection_edge_search_resp	Indicates whether the edge between this process and the sending neighbor has been accepted by the requesting party
diameter_estimate_stmt	Indicates an estimated diameter of the broadcast channel
diameter_reset_stmt	Indicates to reset the estimated diameter to indicated diameter
disconnect_stmt	Indicates that the sending neighbor is disconnecting from the broadcast channel
condition_check_stmt	Indicates that neighbors with empty port condition have been detected
condition_double_check_stmt	Indicates that the neighbors with empty ports have the same set of neighbors
shutdown_stmt	Indicates that the broadcast channel is being shutdown

one call-in port and then disconnects, and another process may then connect to that same broadcast channel using the same call-in port. Before the other process becomes fully connected, another process may try to communicate with it thinking it is the fully connected old process. In such a case, the connect time can be used to identify this situation. In block 803, the routine invokes the seek portal computer routine passing the channel type and channel instance. The seek portal computer routine attempts to locate a portal computer through which this process can connect to the broadcast channel for the passed type and instance. In decision block 804, if the seek portal computer routine is successful in locating a fully connected process on that portal computer, then the routine continues at block 805, else the routine returns an unsuccessful indication. In decision block 805, if no portal computer other than the portal computer on which the process is executing was located, then this is the first process to fully connect to broadcast channel and the routine continues at block 806, else the routine continues at block 808. In block 806, the routine invokes the achieve connection routine to change the state of this process to fully connected. In block 807, the routine installs the external dispatcher for processing messages received through this process' external port for the passed channel type and channel instance. When a message is received through that external port, the external dispatcher is invoked. The routine then returns. In block 808, the routine installs an external dispatcher. In block 809, the routine invokes the connect request routine to initiate the process of identifying neighbors for the seeking computer. The routine then returns.

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

routine hangs up on the selected portal computer. In decision block 910, if the answering process is fully connected to the broadcast channel, then the routine returns a success indicator, else the routine continues at block 911. In block 911, the routine invokes the check for external call routine to determine whether an external call has been made to this process as a portal computer and processes that call. The routine then loops to block 904 to select the next portal computer.

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

FIG. 11 is a flow diagram illustrating the processing of the connect request routine in one embodiment. This routine requests a process of a portal computer that was identified as being fully connected to the broadcast channel to initiate the connection of this process to the broadcast channel. In decision block 1101, if at least one process of a portal computer was located that is fully connected to the broadcast channel, then the routine continues at block 1103, else the routine continues at block 1102. A process of the portal computer may no longer be in the list if it recently disconnected from the broadcast channel. In one embodiment, a seeking computer may always search its entire search depth and find multiple portal computers through which it can connect to the broadcast channel. In block 1102, the routine restarts the process of connecting to the broadcast channel and returns. In block 1103, the routine dials the process of one of the found portal computers through the call-in port. In decision block 1104, if the dialing is successful, then the routine continues at block 1105, else the routine continues at block 1113. The dialing may be unsuccessful if, for example, the dialed process recently disconnected from the broadcast channel. In block 1105, the routine sends an external message to the dialed process requesting a connection to the broadcast channel (i.e., connection_request_call). In block 1106, the routine receives the response message (i.e., connection_request_resp). In decision block 1107, if the response message is successfully received, then the routine

continues at block **1108**, else the routine continues at block **1113**. In block **1108**, the routine sets the expected number of holes (i.e., empty internal connections) for this process based on the received response. When in the large regime, the expected number of holes is zero. When in the small regime, the expected number of holes varies from one to three. In block **1109**, the routine sets the estimated diameter of the broadcast channel based on the received response. In decision block **1111**, if the dialed process is ready to connect to this process as indicated by the response message, then the routine continues at block **1112**, else the routine continues at block **1113**. In block **1112**, the routine invokes the add neighbor routine to add the answering process as a neighbor to this process. This adding of the answering process typically occurs when the broadcast channel is in the small regime. When in the large regime, the random walk search for a neighbor is performed. In block **1113**, the routine hangs up the external connection with the answering process computer and then returns.

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

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

FIG. 14 is a flow diagram illustrating the processing of the external dispatcher routine in one embodiment. This routine is invoked when the external port receives a message. This routine retrieves the message, identifies the external message type, and invokes the appropriate routine to handle that message. This routine loops processing each message until all the received messages have been handled. In block **1401**, the routine answers (e.g., picks up) the external port and retrieves an external message. In decision block **1402**, if a message was retrieved, then the routine continues at block

1403, else the routine hangs up on the external port in block **1415** and returns. In decision block **1403**, if the message type is for a process seeking a connection (i.e., `seeking_connection_call`), then the routine invokes the handle seeking connection call routine in block **1404**, else the routine continues at block **1405**. In decision block **1405**, if the message type is for a connection request call (i.e., `connection_request_call`), then the routine invokes the handle connection request call routine in block **1406**, else the routine continues at block **1407**. In decision block **1407**, if the message type is edge proposal call (i.e., `edge_proposal_call`), then the routine invokes the handle edge proposal call routine in block **1408**, else the routine continues at block **1409**. In decision block **1409**, if the message type is port connect call (i.e., `port_connect_call`), then the routine invokes the handle port connection call routine in block **1410**, else the routine continues at block **1411**. In decision block **1411**, if the message type is a connected statement (i.e., `connected_stint`), the routine invokes the handle connected statement in block **1412**, else the routine continues at block **1212**. In decision block **1412**, if the message type is a condition repair statement (i.e., `condition_repair_stint`), then the routine invokes the handle condition repair routine in block **1413**, else the routine loops to block **1414** to process the next message. After each handling routine is invoked, the routine loops to block **1414**. In block **1414**, the routine hangs up on the external port and continues at block **1401** to receive the next message.

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

FIG. 16 is a flow diagram illustrating processing of the handle connection request call routine in one embodiment. This routine is invoked when the calling process wants this process to initiate the connection of the process to the broadcast channel. This routine either allows the calling process to establish an internal connection with this process (e.g., if in the small regime) or starts the process of identifying a process to which the calling process can connect. In decision block **1601**, if this process is currently fully connected to the broadcast channel, then the routine continues at block **1603**, else the routine hangs up on the external port in block **1602** and returns. In block **1603**, the routine sets the number of holes that the calling process should expect in the response message. In block **1604**, the routine sets the estimated diameter in the response message. In block **1605**, the routine indicates whether this process is ready to connect to the calling process. This process is ready to connect when the number of its holes is greater than zero and the calling process is not a neighbor of this process. In block **1606**, the routine sends to the calling process an external message that is responsive to the connection request call (i.e.,

US 6,714,966 B1

23

connection_request_resp). In block 1607, the routine notes the number of holes that the calling process needs to fill as indicated in the request message. In decision block 1608, if this process is ready to connect to the calling process, then the routine continues at block 1609, else the routine continues at block 1611. In block 1609, the routine invokes the add neighbor routine to add the calling process as a neighbor. In block 1610, the routine decrements the number of holes that the calling process needs to fill and continues at block 1611. In block 1611, the routine hangs up on the external port. In decision block 1612, if this process has no holes or the estimated diameter is greater than one (i.e., in the large regime), then the routine continues at block 1613, else the routine continues at block 1616.

In blocks 1613-1615, the routine loops forwarding a request for an edge through which to connect to the calling process to the broadcast channel. One request is forwarded for each pair of holes of the calling process that needs to be filled. In decision block 1613, if the number of holes of the calling process to be filled is greater than or equal to two, then the routine continues at block 1614, else the routine continues at block 1616. In block 1614, the routine invokes the forward connection edge search routine. The invoked routine is passed to an indication of the calling process and the random walk distance. In one embodiment, the distance is twice in the estimated diameter of the broadcast channel. In block 1614, the routine decrements the holes left to fill by two and loops to block 1613. In decision block 1616, if there is still a hole to fill, then the routine continues at block 1617, else the routine returns. In block 1617, the routine invokes the fill hole routine passing the identification of the calling process. The fill hole routine broadcasts a connection port search statement (i.e., connection_port_search_stint) for a hole of a connected process through which the calling process can connect to the broadcast channel. The routine then returns.

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

24

routine continues at block 1711. In block 1710, the routine invokes the achieve connected routine to indicate that this process is fully connected. In decision block 1711, if the number of holes for this process is zero, then the routine continues at block 1712, else the routine returns. In block 1712, the routine deletes any pending edges and then returns. A pending edge is an edge that has been proposed to this process for edge pinning, which in this case is no longer needed.

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

FIG. 19 is a flow diagram illustrating the processing of the handle edge proposal call routine. This routine is invoked when a message is received from a proposing process that proposes to connect an edge between the proposing process and one of its neighbors to this process for edge pinning. In decision block 1901, if the number of holes of this process minus the number of pending edges is greater than or equal to one, then this process still has holes to be filled and the routine continues at block 1902, else the routine continues at block 1911. In decision block 1902, if the proposing process or its neighbor is a neighbor of this process, then the routine continues at block 1911, else the routine continues at block 1903. In block 1903, the routine indicates that the edge is pending between this process and the proposing process. In decision block 1904, if a proposed neighbor is already pending as a proposed neighbor, then the routine continues at block 1911, else the routine continues at block 1907. In block 1907, the routine sends an edge proposal response as

an external message to the proposing process (i.e., edge_proposal_resp) indicating that the proposed edge is accepted. In decision block 1908, if the sending of the message was successful, then the routine continues at block 1909, else the routine returns. In block 1909, the routine adds the edge as a pending edge. In block 1910, the routine invokes the add_neighbor routine to add the proposing process on the external port as a neighbor. The routine then returns. In block 1911, the routine sends an external message (i.e., edge_proposal_resp) indicating that this proposed edge is not accepted. In decision block 1912, if the number of holes is odd, then the routine continues at block 1913, else the routine returns. In block 1913, the routine invokes the fill hole routine and then returns.

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

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

FIG. 22 is a flow diagram illustrating the processing of the internal dispatcher routine in one embodiment. This routine is passed an indication of the neighbor who sent the internal message. In block 2201, the routine receives the internal message. This routine identifies the message type and invokes the appropriate routine to handle the message. In block 2202, the routine assesses whether to change the estimated diameter of the broadcast channel based on the information in the received message. In decision block 2203, if this process is the originating process of the message or the message has already been received (i.e., a duplicate), then the routine ignores the message and continues at block 2208, else the routine continues at block 2203 A. In decision

block 2203 A, if the process is partially connected, then the routine continues at block 2203 B, else the routine continues at block 2204. In block 2203 B, the routine adds the message to the pending connection buffer and continues at block 2204. In decision blocks 2204–2207, the routine decodes the message type and invokes the appropriate routine to handle the message. For example, in decision block 2204, if the type of the message is broadcast statement (i.e., broadcast_stmt), then the routine invokes the handle broadcast message routine in block 2205. After invoking the appropriate handling routine, the routine continues at block 2208. In decision block 2208, if the partially connected buffer is full, then the routine continues at block 2209, else the routine continues at block 2210. The broadcaster component collects all its internal messages in a buffer while partially connected so that it can forward the messages as it connects to new neighbors. If, however, that buffer becomes full, then the process assumes that it is now fully connected and that the expected number of connections was too high, because the broadcast channel is now in the small regime. In block 2209, the routine invokes the achieve connection routine and then continues in block 2210. In decision block 2210, if the application program message queue is empty, then the routine returns, else the routine continues at block 2212. In block 2212, the routine invokes the receive response routine passing the acquired message and then returns. The received response routine is a callback routine of the application program.

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

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

FIG. 26 is a flow diagram illustrating the processing of the handle connection port search statement routine in one embodiment. This routine is passed an indication of the neighbor that sent the message and the message itself. In block 2601, the routine invokes the distribute internal message which sends the message to each of its neighbors other than the sending neighbor. In decision block 2602, if the number of holes of this process is greater than zero, then the routine continues at block 2603, else the routine returns. In decision block 2603, if the requesting process is a neighbor, then the routine continues at block 2605, else the routine continues at block 2604. In block 2604, the routine invokes the court neighbor routine and then returns. The court

27

neighbor routine connects this process to the requesting process if possible. In block 2605, if this process has one hole, then the neighbors with empty ports condition exists and the routine continues at block 2606, else the routine returns. In block 2606, the routine generates a condition check message (i.e., condition_check) that includes a list of this process' neighbors. In block 2607, the routine sends the message to the requesting neighbor.

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

FIG. 28 is a flow diagram illustrating the processing of the handle connection edge search call routine in one embodiment. This routine is passed a indication of the neighbor who sent the message and the message itself. This routine either forwards the message to a neighbor or proposes the edge between this process and the sending neighbor to the requesting process for edge pinning. In decision block 2801, if this process is not the requesting process or the number of holes of the requesting process is still greater than or equal to two, then the routine continues at block 2802, else the routine continues at block 2813. In decision block 2802, if the forwarding distance is greater than zero, then the random walk is not complete and the routine continues at block 2803, else the routine continues at block 2804. In block 2803, the routine invokes the forward connection edge search routine passing the identification of the requesting process and the decremented forwarding distance. The routine then continues at block 2815. In decision block 2804, if the requesting process is a neighbor or the edge between this process and the sending neighbor is reserved because it has already been offered to a process, then the routine continues at block 2805, else the routine continues at block 2806. In block 2805, the routine invokes the forward connection edge search routine passing an indication of the requesting party and a toggle indicator that alternatively indicates to continue the random walk for one or two more computers. The routine then continues at block 2815. In block 2806, the routine dials the requesting process via the call-in port. In block 2807, the routine sends an edge proposal call external message (i.e., edge_proposal_call) and receives the response (i.e., edge_proposal_resp). Assuming that the response is successfully received, the routine continues at block 2808. In decision block 2808, if the response indicates that the edge is acceptable to the requesting process, then the routine continues at block 2809, else the routine continues at block 2812. In block 2809, the routine reserves the edge between this process and the sending neighbor. In block 2810, the routine adds the requesting process as a neighbor by invoking the add_neighbor routine. In block 2811, the routine

28

removes the sending neighbor as a neighbor. In block 2812, the routine hangs up the external port and continues at block 2815. In decision block 2813, if this process is the requesting process and the number of holes of this process equals one, then the routine continues at block 2814, else the routine continues at block 2815. In block 2814, the routine invokes the fill hole routine. In block 2815, the routine sends an connection edge search response message (i.e., connection_edge_search_response) to the sending neighbor indicating acknowledgement and then returns. The graphs are sensitive to parity. That is, all possible paths starting from a node and ending at that node will have an even length unless the graph has a cycle whose length is odd. The broadcaster component uses a toggle indicator to vary the random walk distance between even and odd distances.

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

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

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

FIGS. 32-34 are flow diagrams illustrating the processing of messages associated with the neighbors with empty ports condition. FIG. 32 is a flow diagram illustrating processing of the handle condition check message in one embodiment. This message is sent by a neighbor process that has one hole and has received a request to connect to a hole of this

process. In decision block 3201, if the number of holes of this process is equal to one, then the routine continues at block 3202, else the neighbors with empty ports condition does not exist any more and the routine returns. In decision block 3202, if the sending neighbor and this process have the same set of neighbors, the routine continues at block 3203, else the routine continues at block 3205. In block 3203, the routine initializes a condition double check message (i.e., condition_double_check) with the list of neighbors of this process. In block 3204, the routine sends the message internally to a neighbor other than sending neighbor. The routine then returns. In block 3205, the routine selects a neighbor of the sending process that is not also a neighbor of this process. In block 3206, the routine sends a condition repair message (i.e., condition_repair_stmt) externally to the selected process. In block 3207, the routine invokes the add neighbor routine to add the selected neighbor as a neighbor of this process and then returns.

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

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

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

What is claimed is:

1. A computer network for providing an information delivery service for a plurality of participants, each participant having connections to at least three neighbor participants, wherein an originating participant sends data to the other participants by sending the data through each of its connections to its neighbor participants and wherein each participant sends data that it receives from a neighbor participant to its other neighbor participants, further wherein the network is m-regular, where m is the exact number of neighbor participants of each participant and further wherein the number of participants is at least two greater than m thus resulting in a non-complete graph.

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

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

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

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

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

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

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

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

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

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

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

13. An information delivery service comprising:

a plurality of broadcast channels, each broadcast channel for distributing information relating to a topic, each of the broadcast channels for providing said information related to a topic to a plurality of participants, each participant having connections to at least three neighbor participants, wherein an originating participant sends data to the other participants by sending the data through each of its connections to its neighbor participants and wherein each participant sends data that it receives from a neighbor participant to its neighbor participants, further wherein the network is m-regular, where m is the exact number of neighbor participants of each participant and further wherein the number of participants is at least two greater than m thus resulting in a non-complete graph;

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

means for connecting to the identified broadcast channel.

14. The information delivery service of claim 13 wherein means for identifying a topic of interest includes accessing a web server that maps topics to corresponding broadcast channel.

15. The information deliver service of claim 13 wherein a broadcast channel is formed by subscriber computers that are each interconnected to at least three other subscriber computers.

16. A computer network for providing an information delivery service for a plurality of participants, each partici-

US 6,714,966 B1

31

pant having connections to exactly four neighbor participants, wherein an originating participant sends data to the other participants by sending the data through each of its connections to its neighbor participants and wherein each participant sends data that it receives from a neighbor participant to its neighbor participants, further wherein the

32

network is in a stable 4-regular state and wherein there are at least six participants to result in a non-complete graph.

17. The computer network of claim 16 wherein a computer hosts more than one participant.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 6,714,966 B1
DATED : March 30, 2004
INVENTOR(S) : Fred B. Holt et al.

Page 1 of 1

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

Column 9,

Line 8, "(e.g.," should be -- (e.g., --;

Column 21,

Line 55, "stint" should be -- stmt --;

Column 22,

Lines 19 and 23, "stint" should be -- stmt --;

Column 23,

Line 33, "stint" should be -- stmt --;

Column 25,

Line 67, delete space between "2203" and "A"

Column 26,

Line 1, delete space between "2203" and "A";

Lines 2 and 3, delete space between "2203" and "B";

Column 27,

Line 32, insert period between "itself" and "This";

Column 29,

Line 62, "(e.g.," should be -- (e.g., --;

Signed and Sealed this

Twenty-ninth Day of June, 2004



JON W. DUDAS

Acting Director of the United States Patent and Trademark Office

(12) **INTER PARTES REVIEW CERTIFICATE** (1153rd)

United States Patent
Bourassa et al.

(10) **Number:** **US 6,714,966 K1**
(45) **Certificate Issued:** **Apr. 11, 2019**

(54) **INFORMATION DELIVERY SERVICE**

(75) Inventors: **Virgil E. Bourassa; Fred B. Holt**

(73) Assignee: **ACCELERATION BAY, LLC**

Trial Numbers:

IPR2015-01951 filed Sep. 24, 2015

IPR2016-00935 filed Apr. 22, 2016

IPR2015-01953 filed Sep. 24, 2015

IPR2016-00936 filed Apr. 22, 2016

Inter Partes Review Certificate for:

Patent No.: **6,714,966**

Issued: **Mar. 30, 2004**

Appl. No.: **09/629,043**

Filed: **Jul. 31, 2000**

The results of IPR2015-01951 joined with IPR2016-00935; IPR2015-01953 joined with IPR2016-00936 are reflected in this inter partes review certificate under 35 U.S.C. 318(b).

INTER PARTES REVIEW CERTIFICATE
U.S. Patent 6,714,966 K1
Trial No. IPR2015-01951
Certificate Issued Apr. 11, 2019

1

AS A RESULT OF THE INTER PARTES
REVIEW PROCEEDING, IT HAS BEEN
DETERMINED THAT:

Claim 12 is found patentable.

Claims 1-11 and 16-17 are cancelled.

19. (substitute for claim 7) *A computer network for providing an information delivery service for a plurality of participants, each participant having connections to at least three neighbor participants, wherein an originating participant sends data to the other participants by sending the data through each of its connections to its neighbor participants and wherein each participant sends data that it receives from a neighbor participant to its other neighbor participants,*

2

further wherein the network is m-regular, where m is the exact number of neighbor participants of each participant,
further wherein the number of participants is at least two greater than m thus resulting in a non-complete graph,
further wherein the connections are peer-to-peer connections,
further wherein the network is formed through a broadcast channel that overlays an underlying network,
further wherein the information delivery service is provided by at least one information delivery service application program executing on each computer of the computer network that interacts with the broadcast channel, and
further wherein participants can join and leave the network using the broadcast channel.

* * * * *

EXHIBIT 20

**THIS EXHIBIT HAS BEEN
REDACTED IN ITS ENTIRETY**

EXHIBIT 21



US006732147B1

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

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

(54) **LEAVING A BROADCAST CHANNEL**

OTHER PUBLICATIONS

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

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

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

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

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

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

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

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

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

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

(51) **Int. Cl.⁷** **G06F 15/16**

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

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

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

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

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

(56) **References Cited**

U.S. PATENT DOCUMENTS

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

(List continued on next page.)

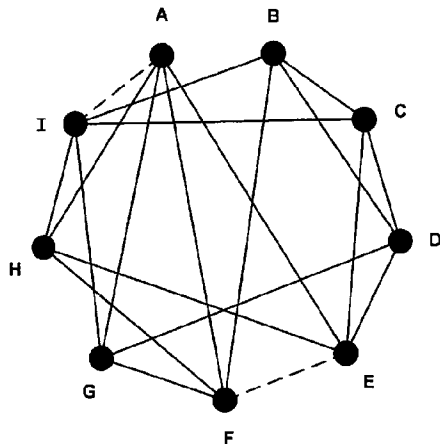
Primary Examiner—Patrice Winder
Assistant Examiner—David Lazaro
(74) *Attorney, Agent, or Firm*—Perkins Coie LLP

(57) **ABSTRACT**

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

(List continued on next page.)

16 Claims, 39 Drawing Sheets



US 6,732,147 B1

Page 2

U.S. PATENT DOCUMENTS

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

OTHER PUBLICATIONS

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

* cited by examiner

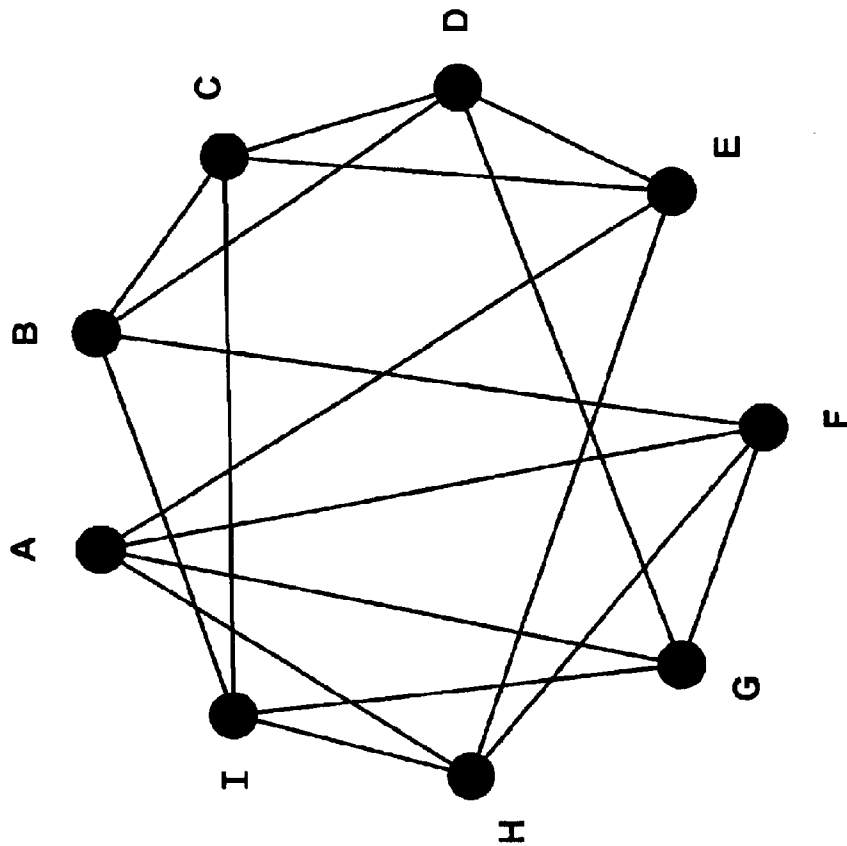


Fig. 1

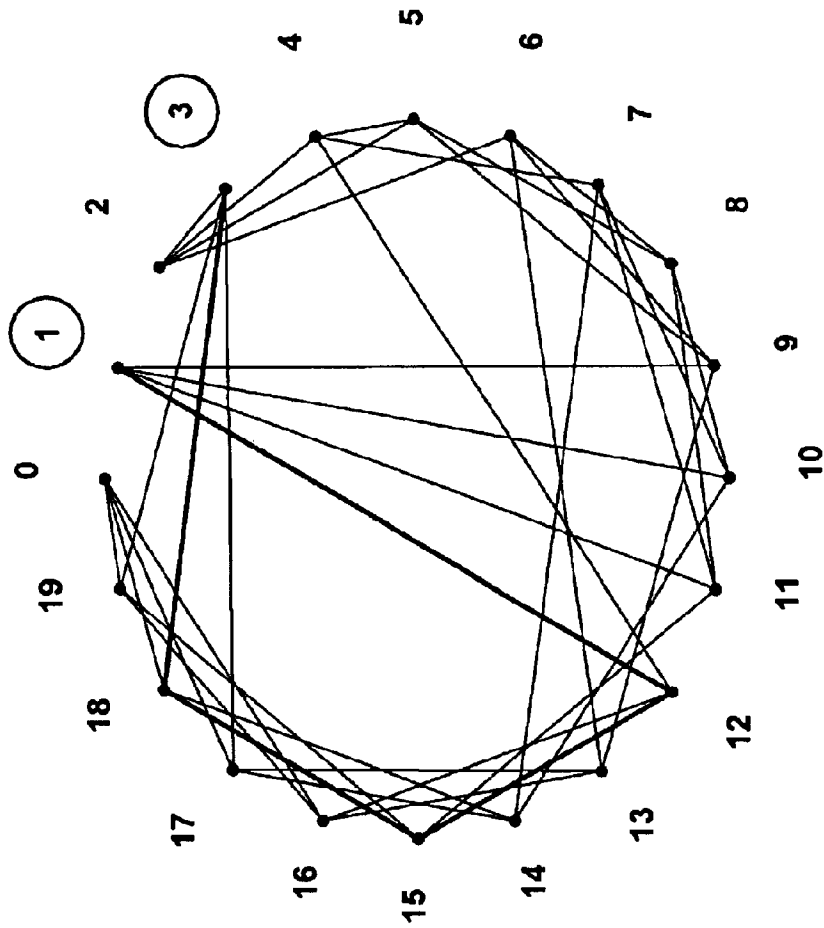


Fig. 2

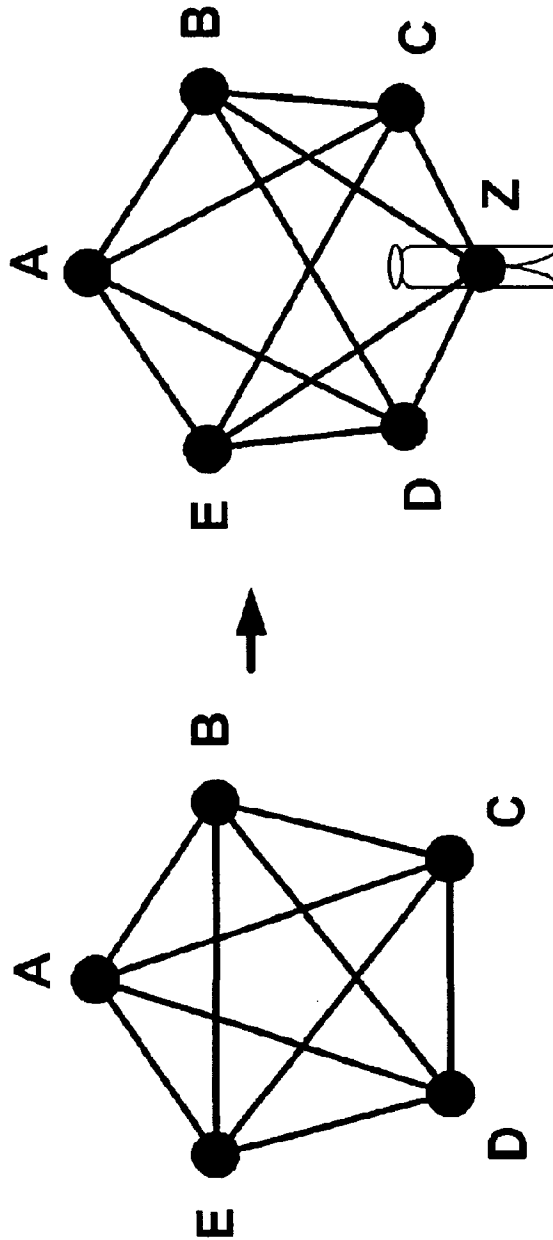


Fig. 3B

Fig. 3A

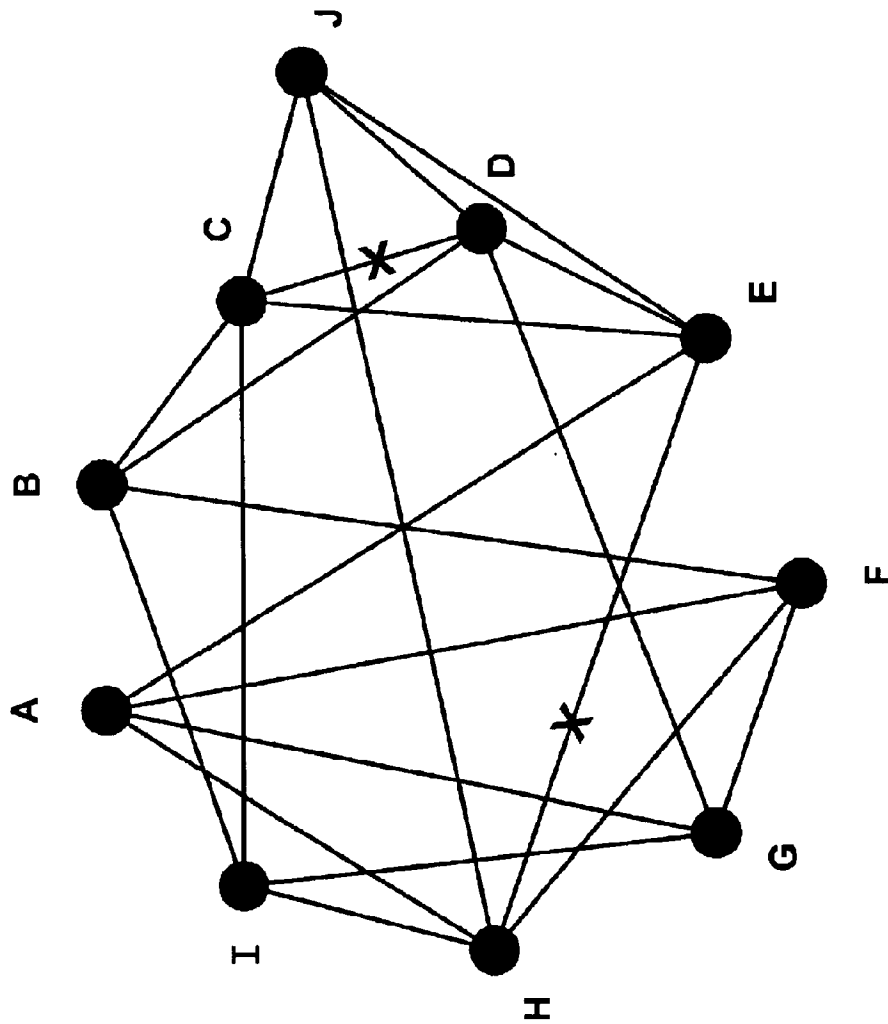


Fig. 4A

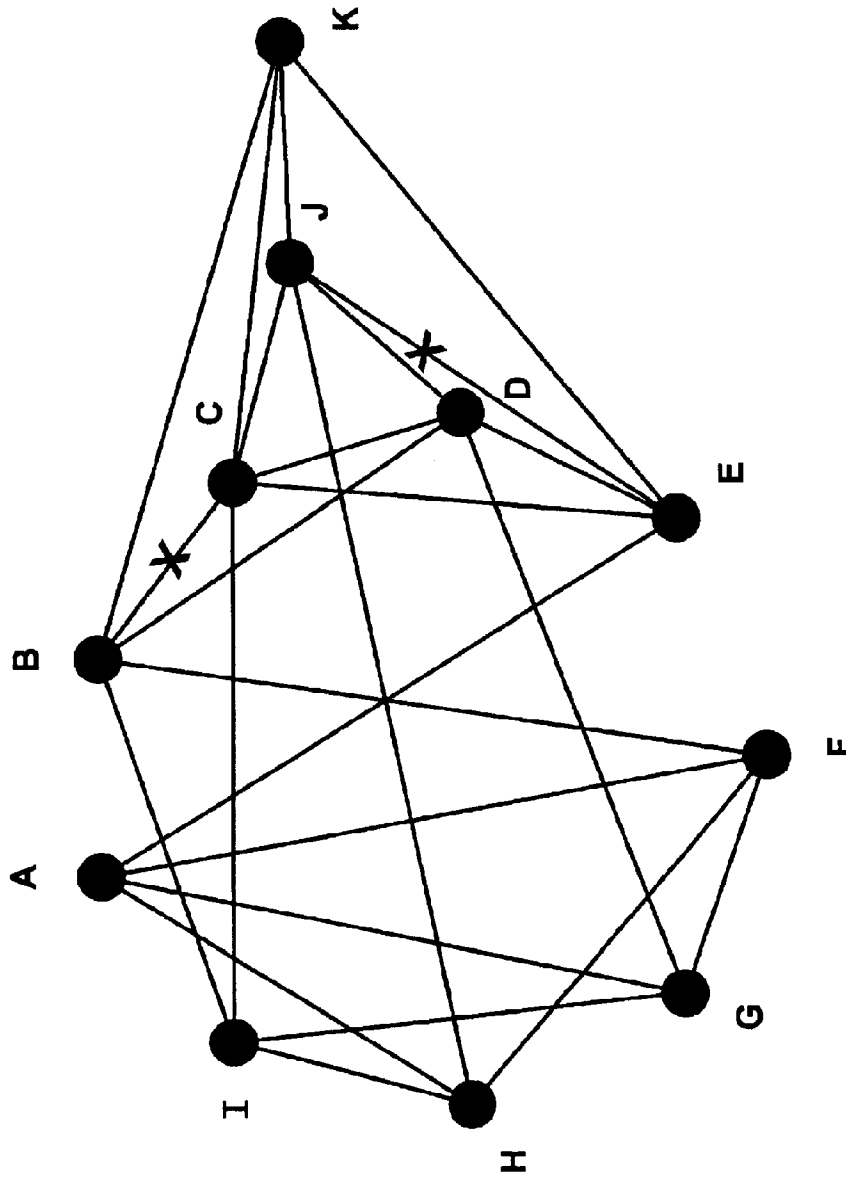


Fig. 4B

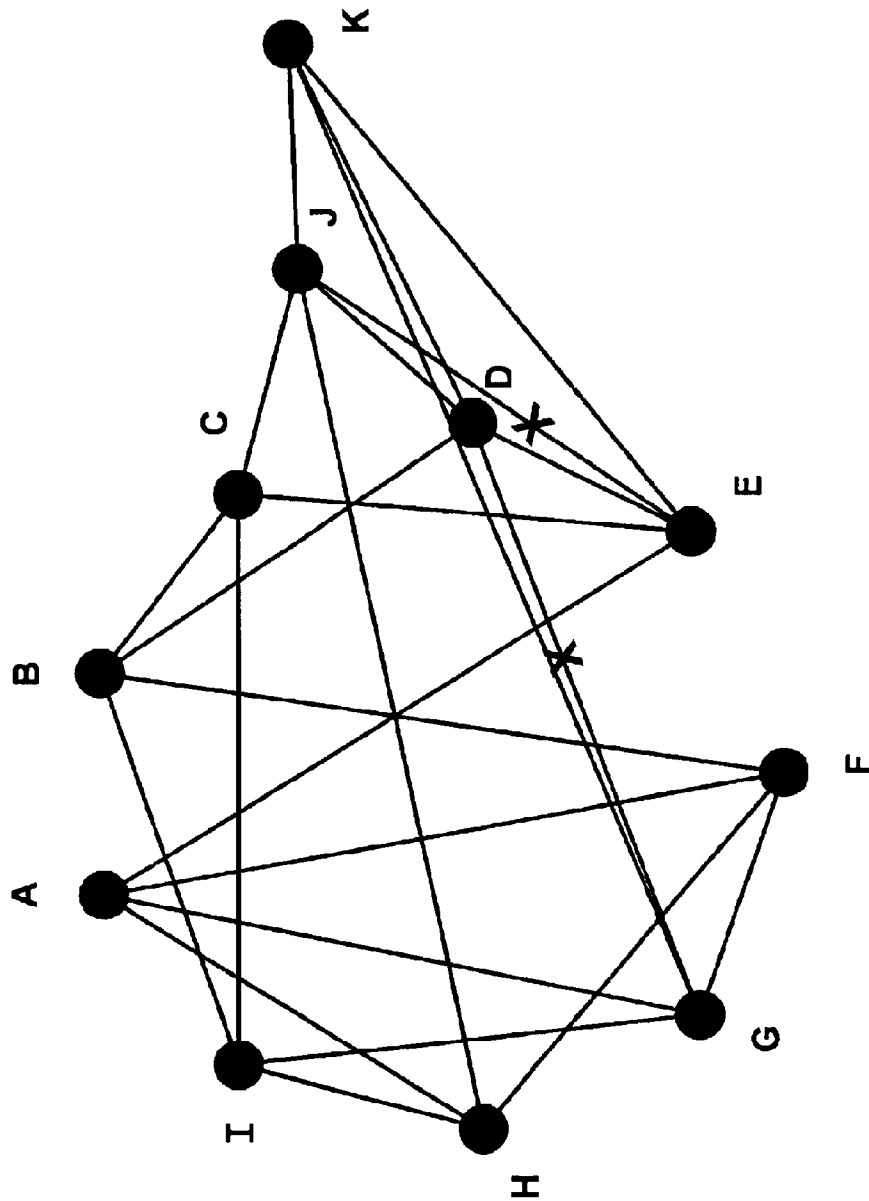


Fig. 4C

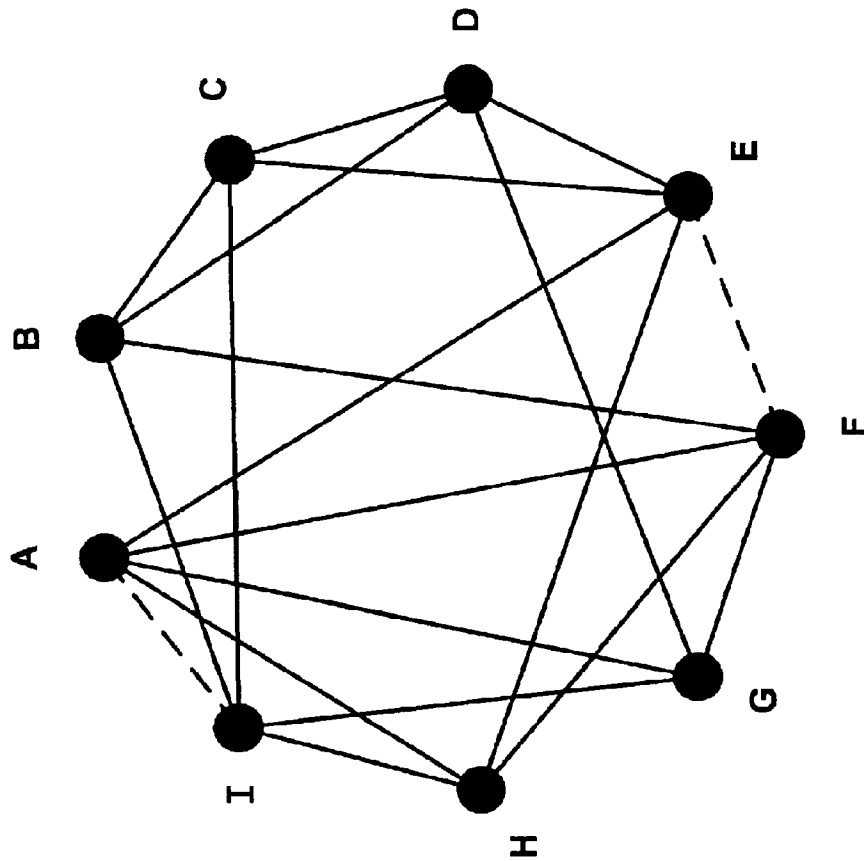


Fig. 5A

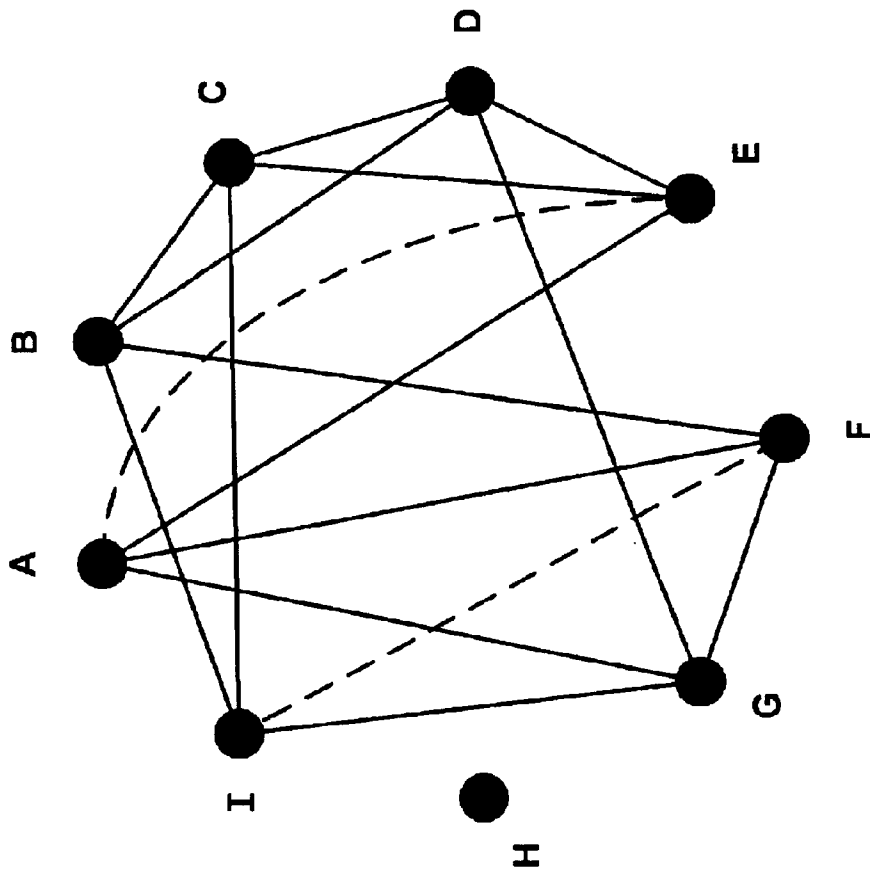


Fig. 5B

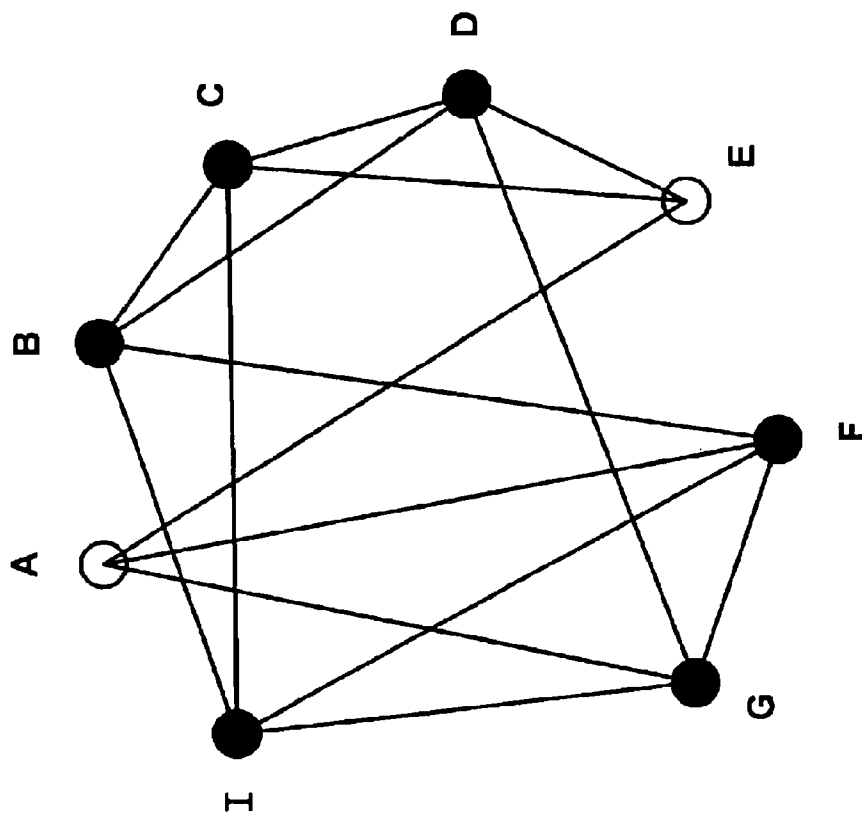


Fig. 5C

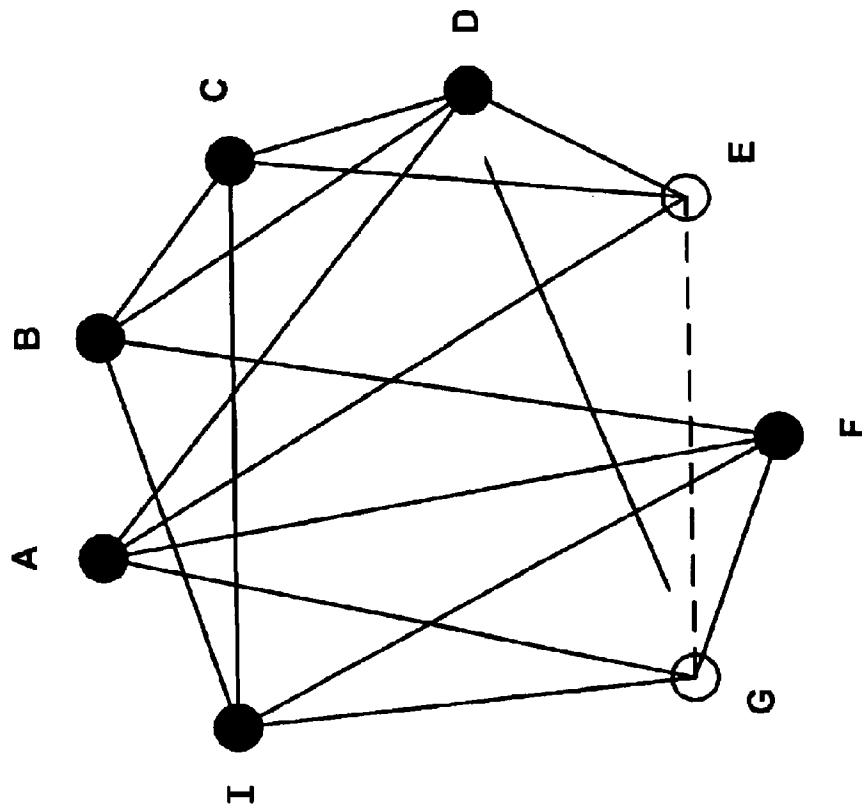


Fig. 5D

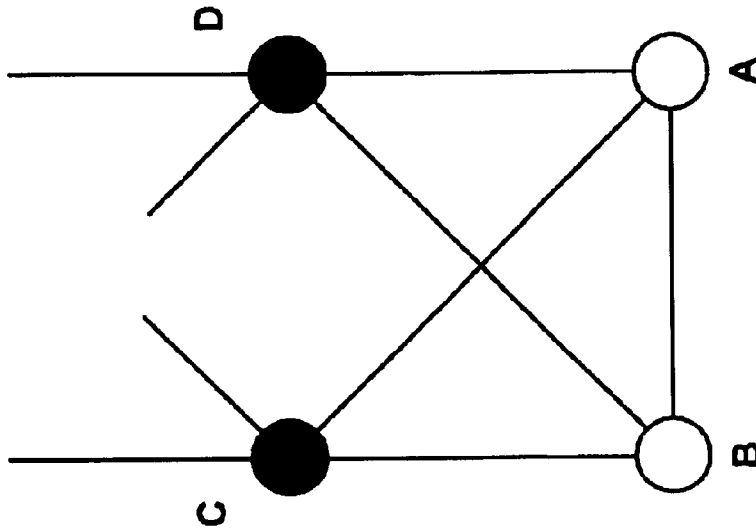


Fig. 5F

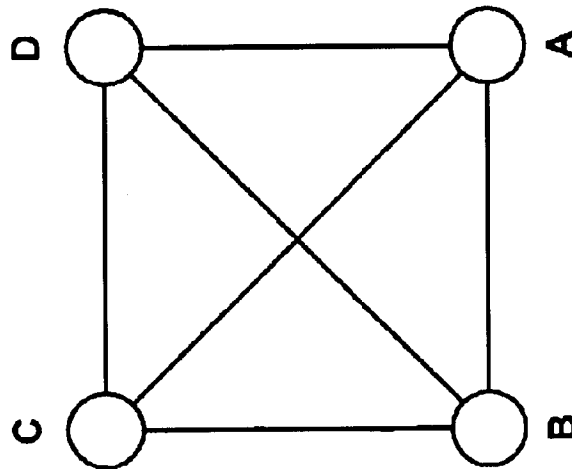


Fig. 5E

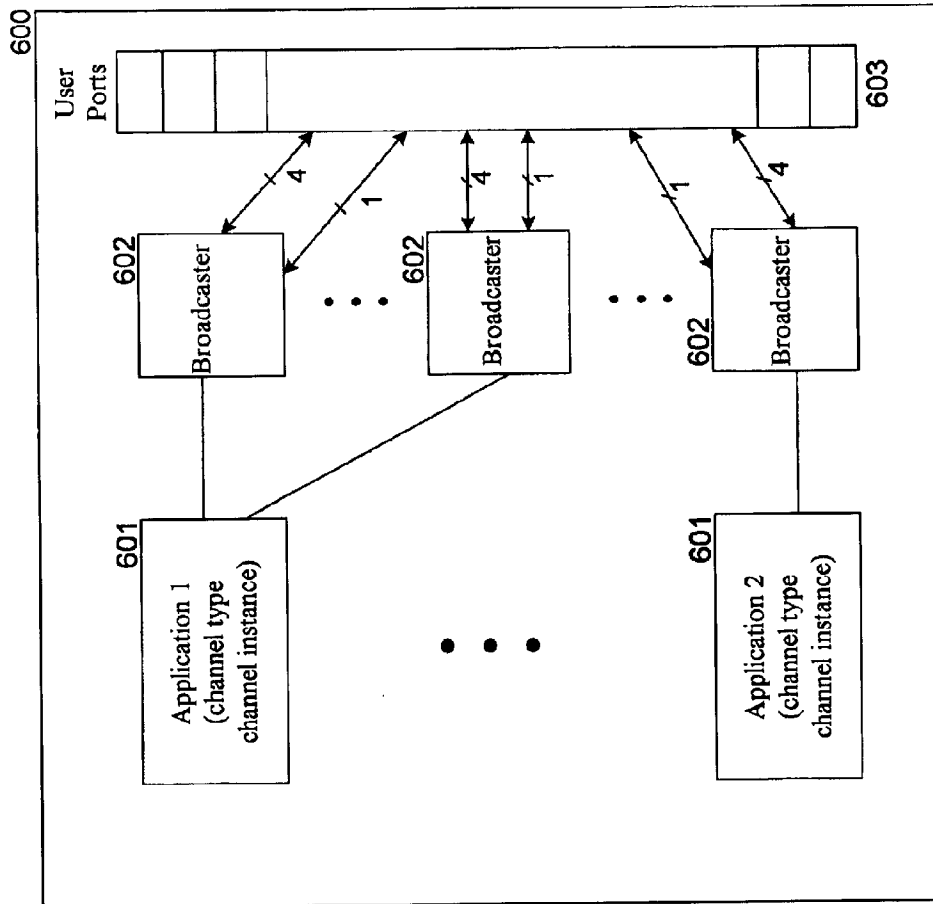


Fig. 6

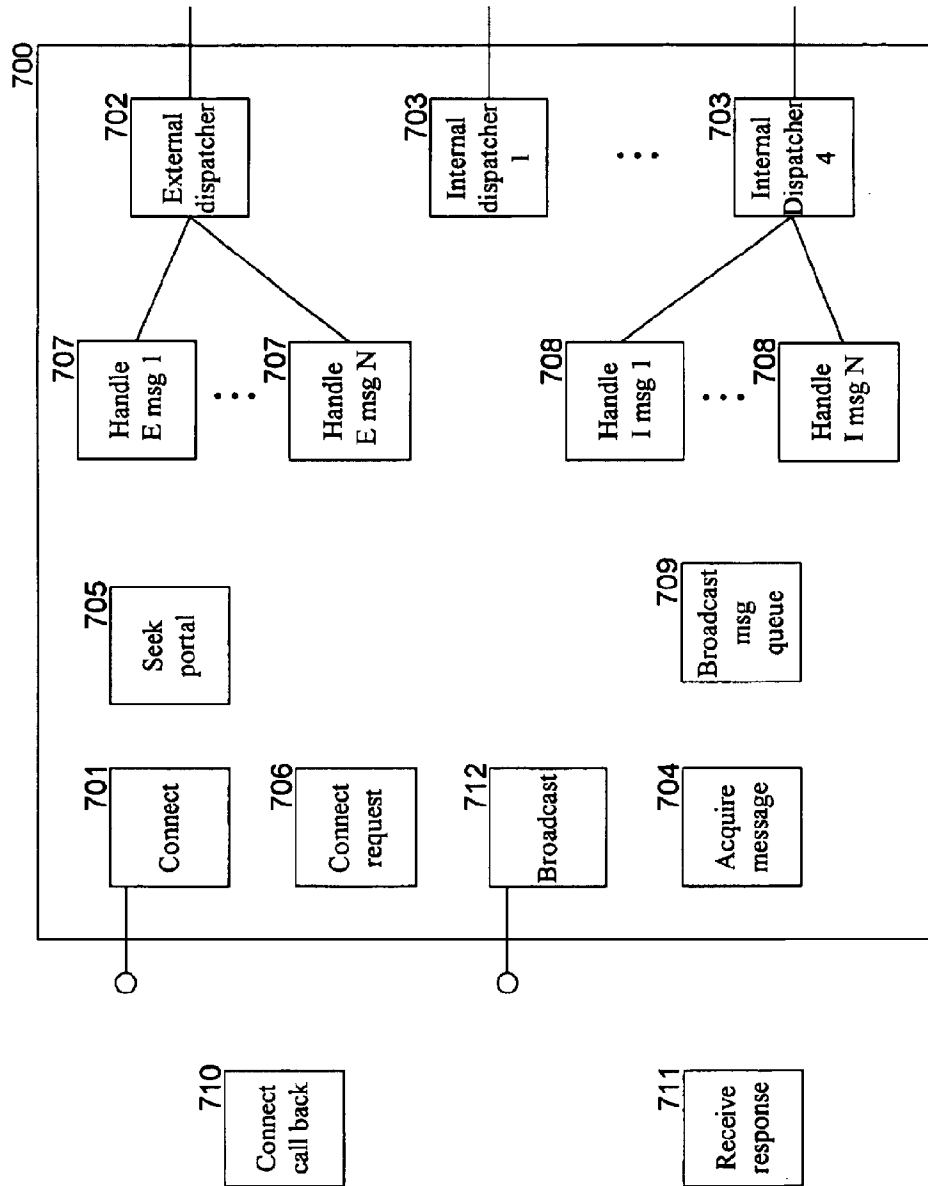
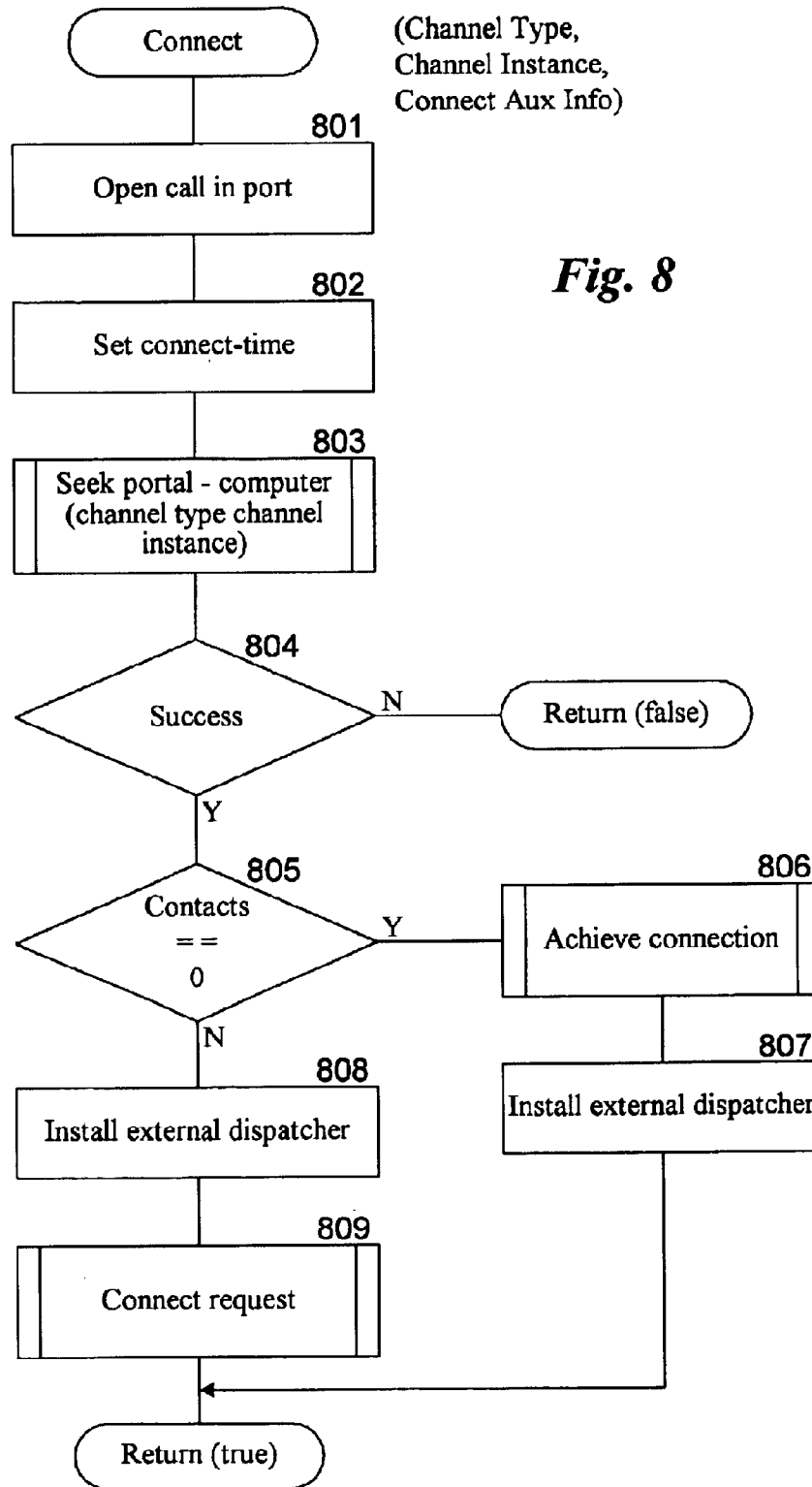


Fig. 7



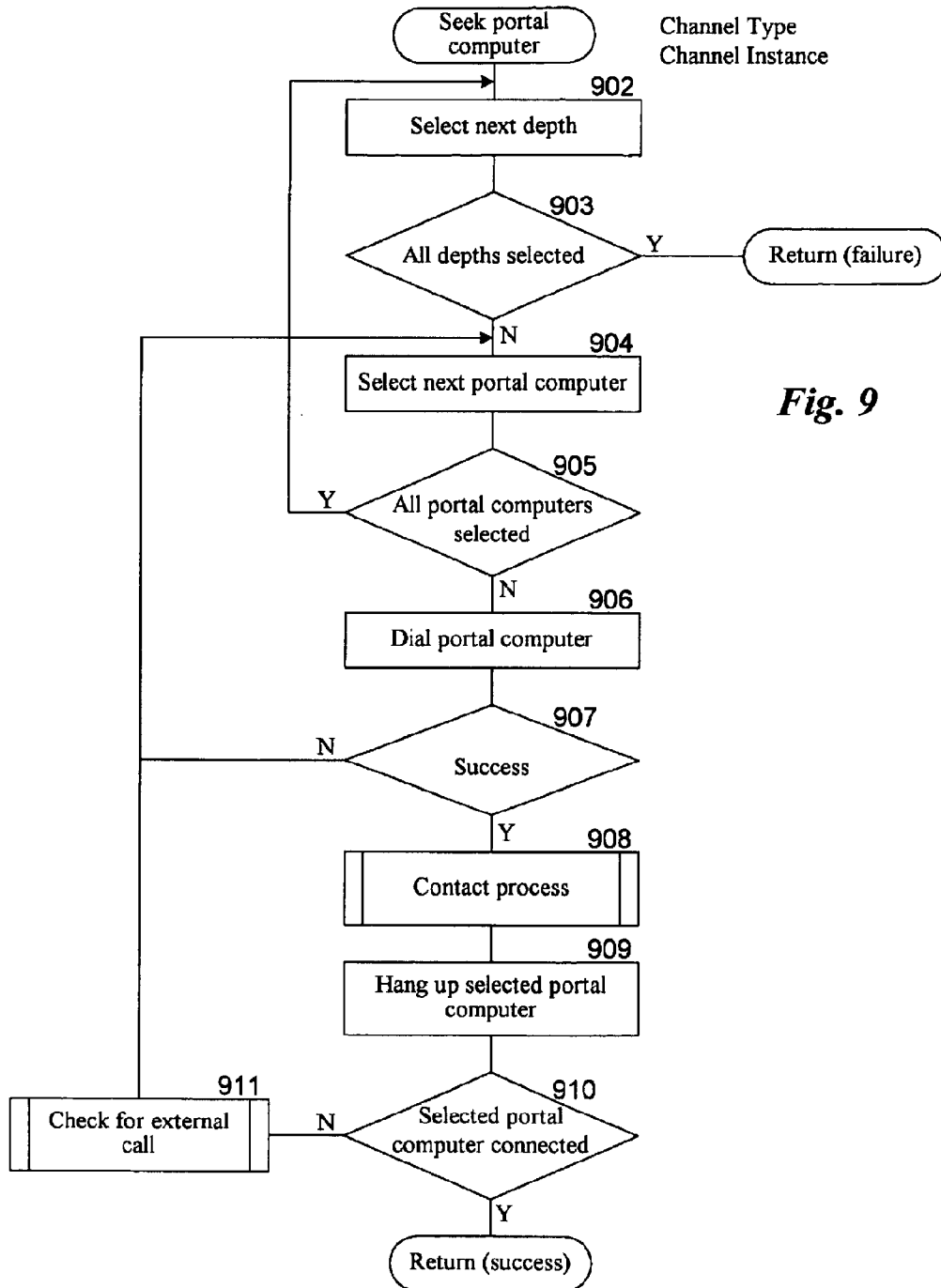


Fig. 9

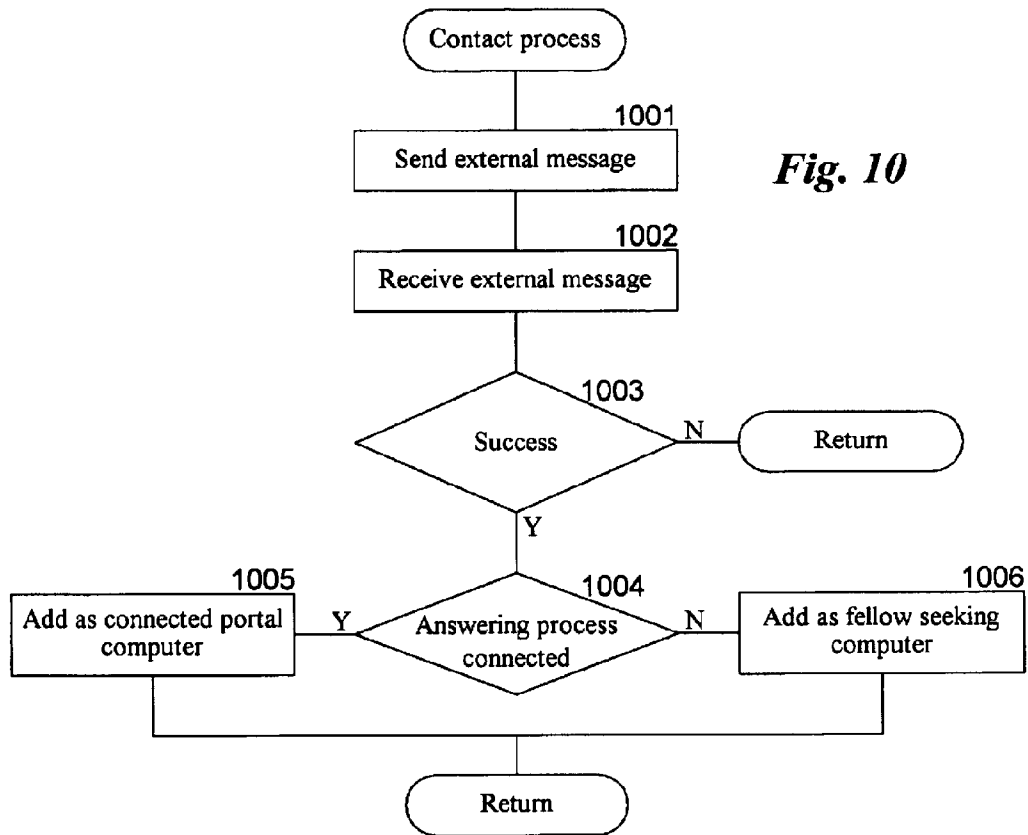
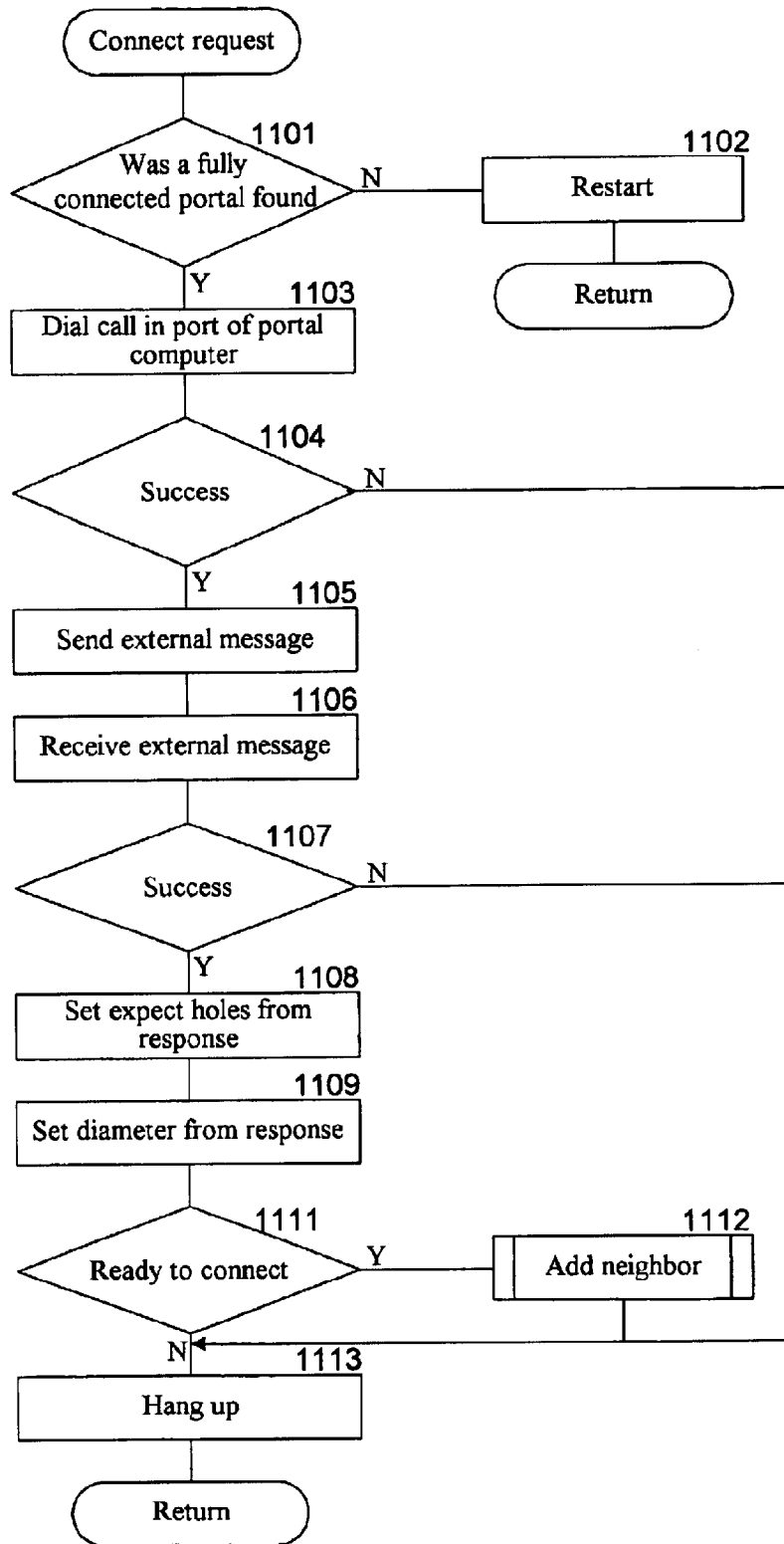


Fig. 11



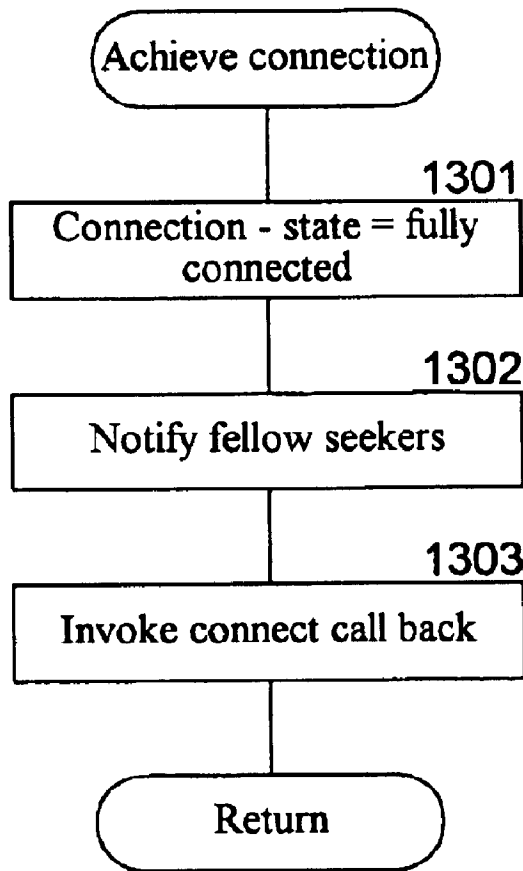


Fig. 13

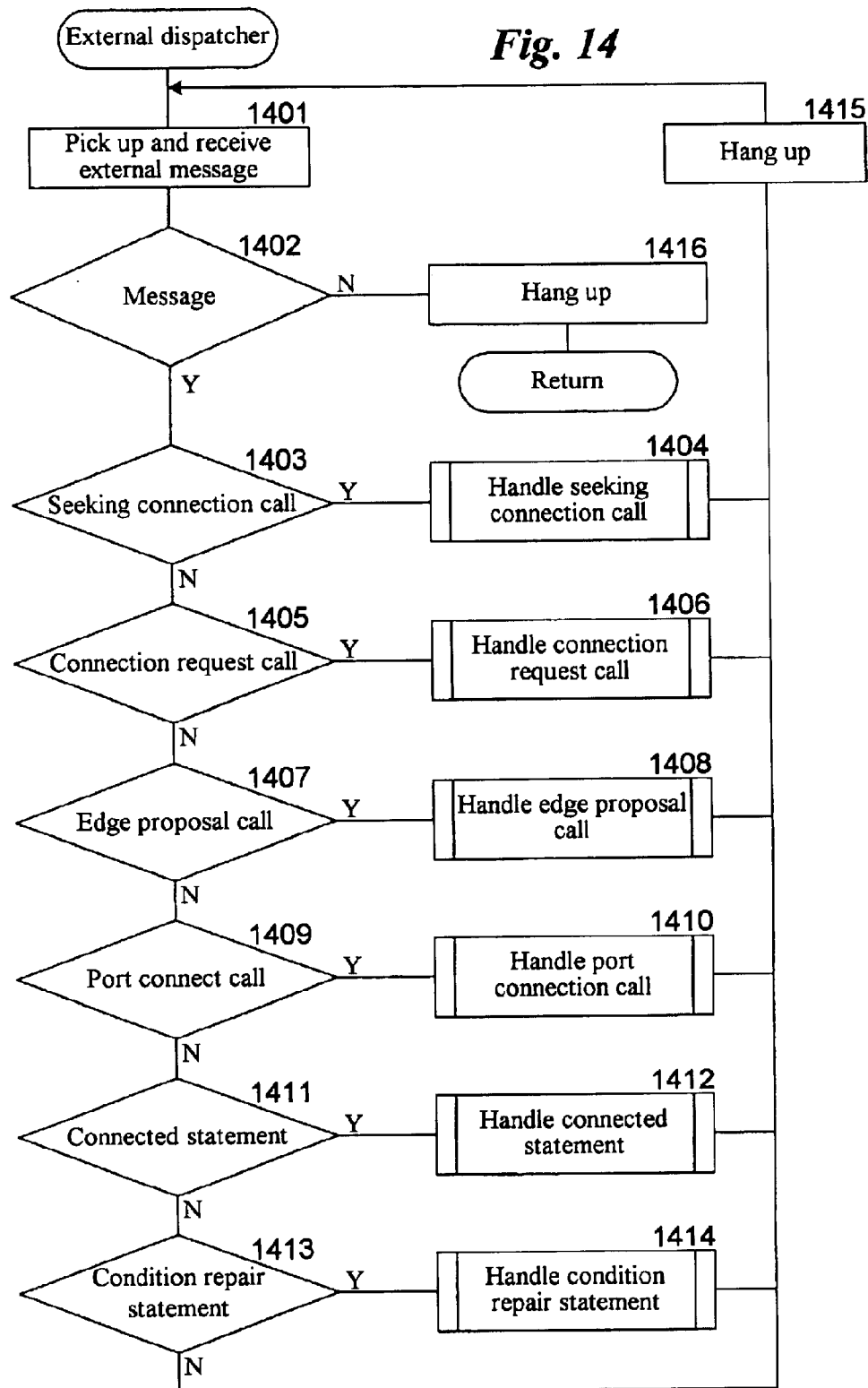
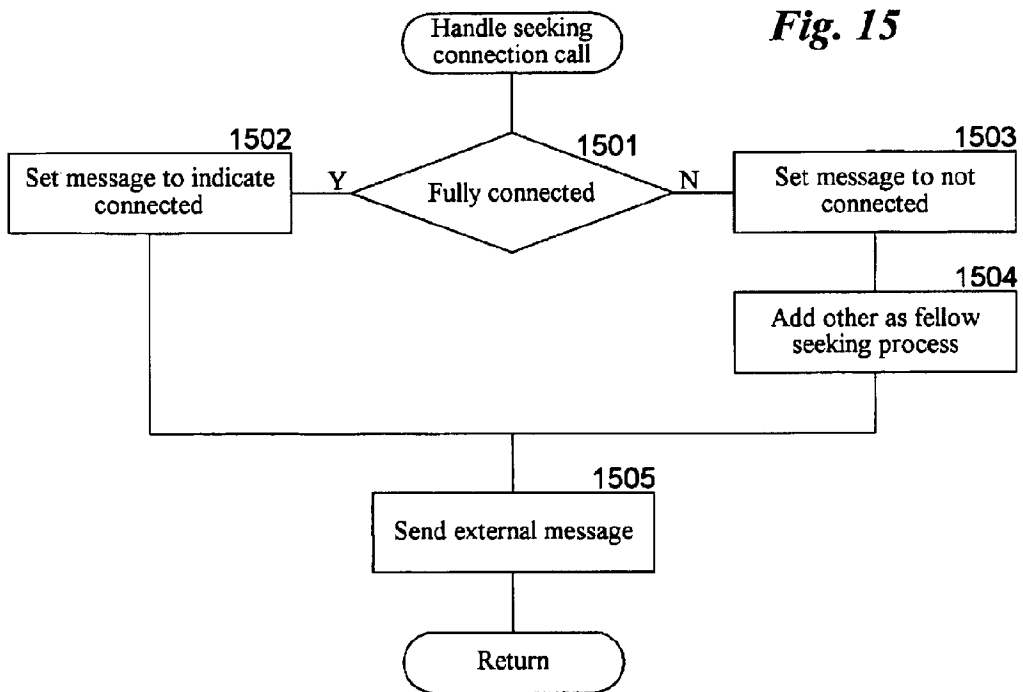


Fig. 15



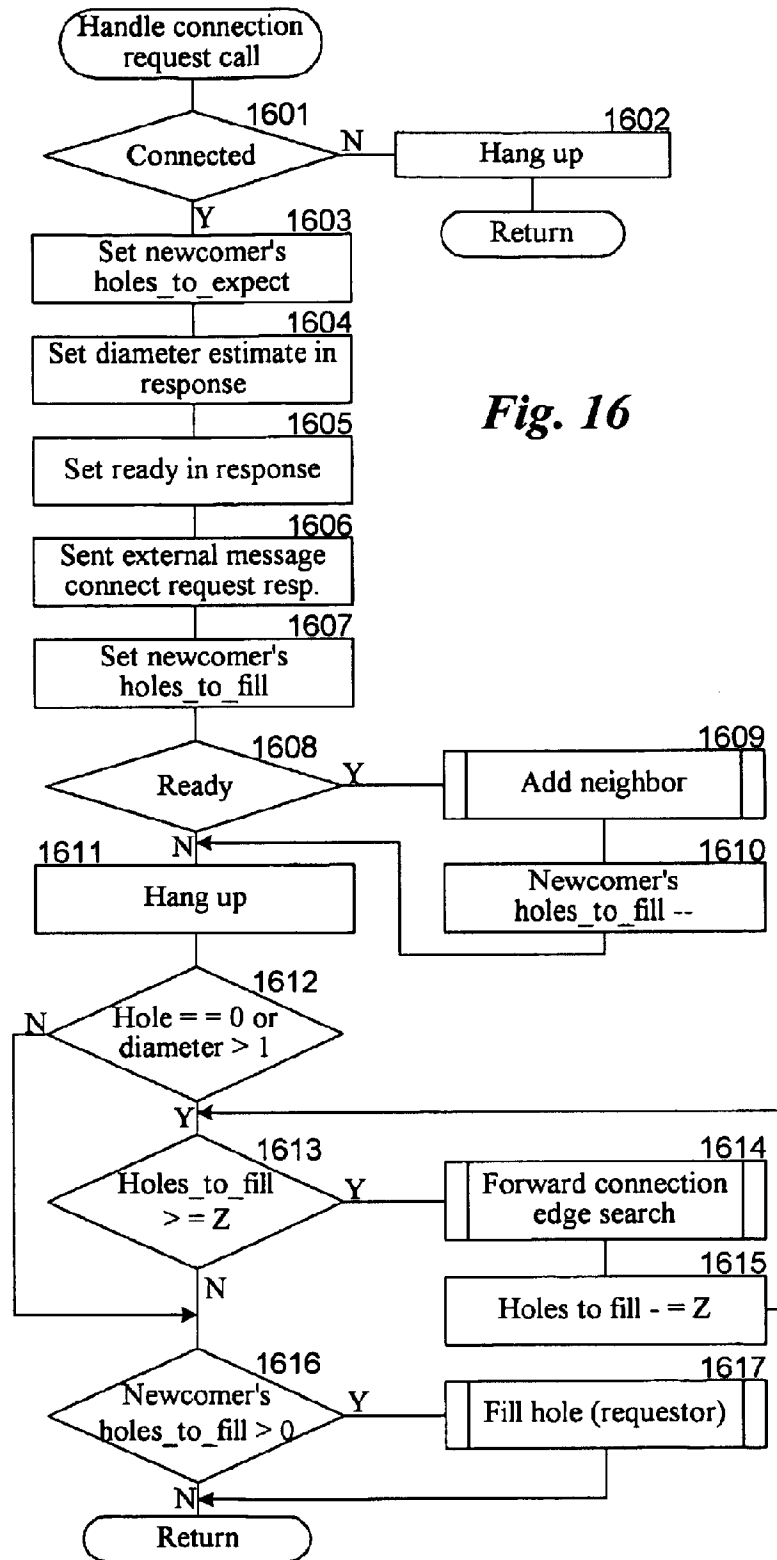


Fig. 16

Fig. 17

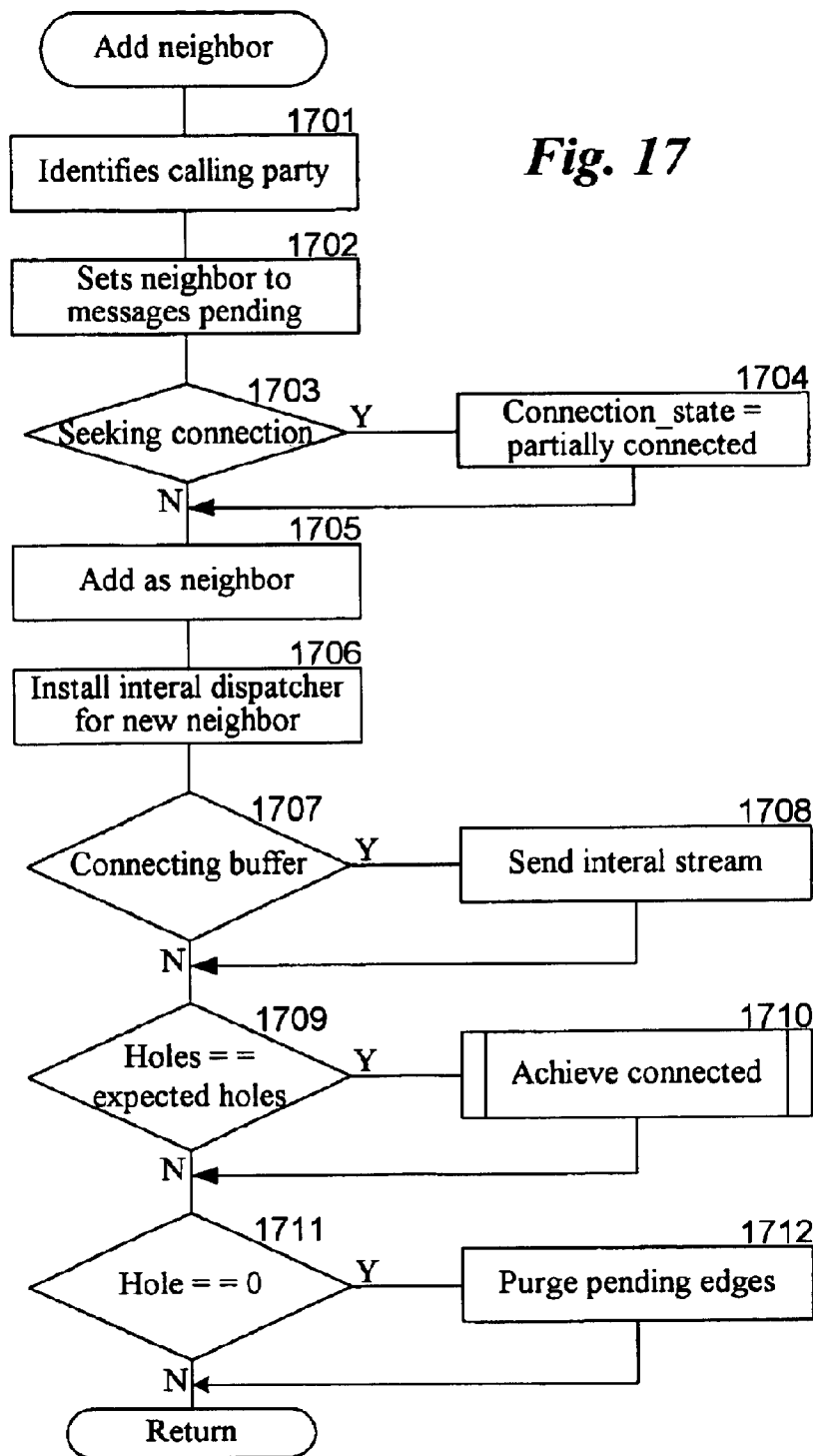
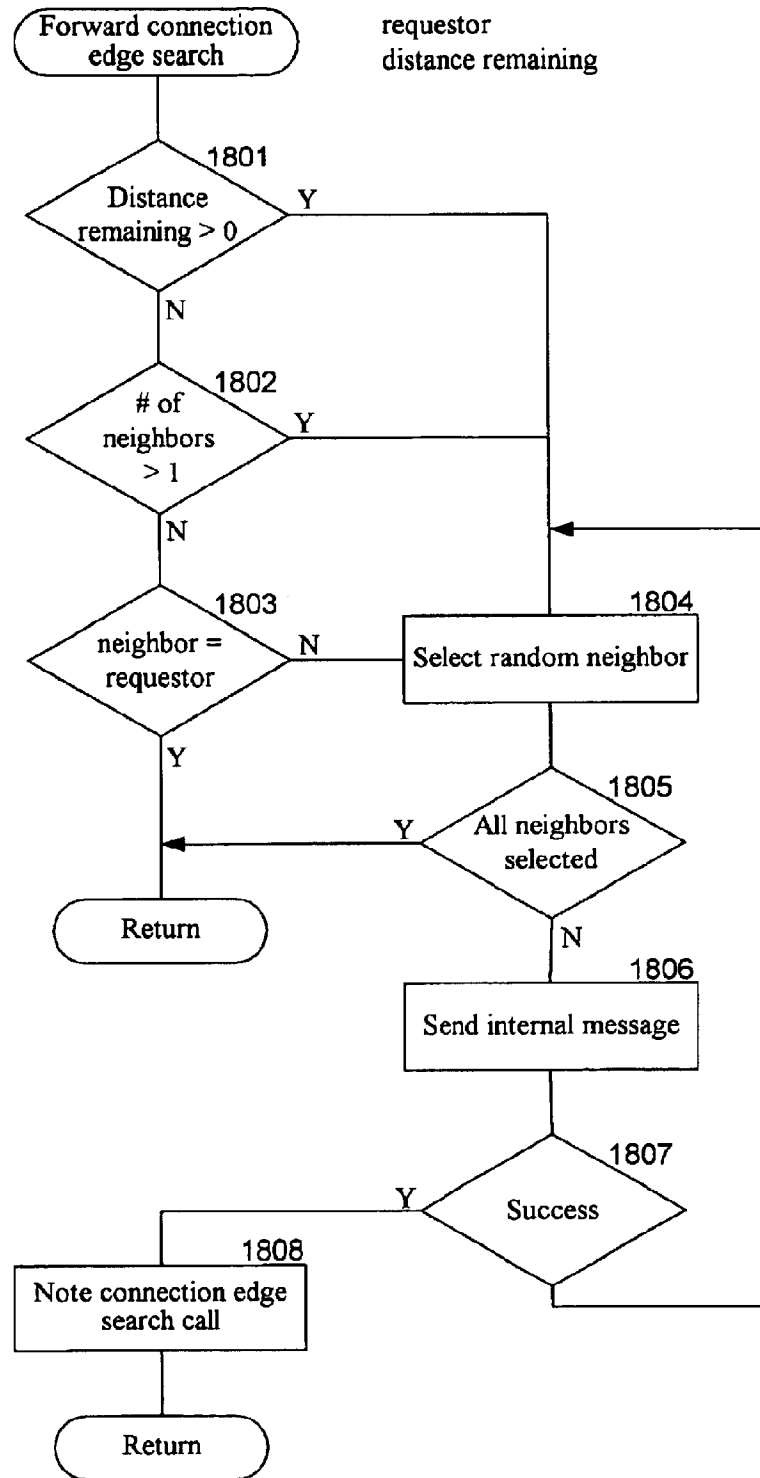


Fig. 18



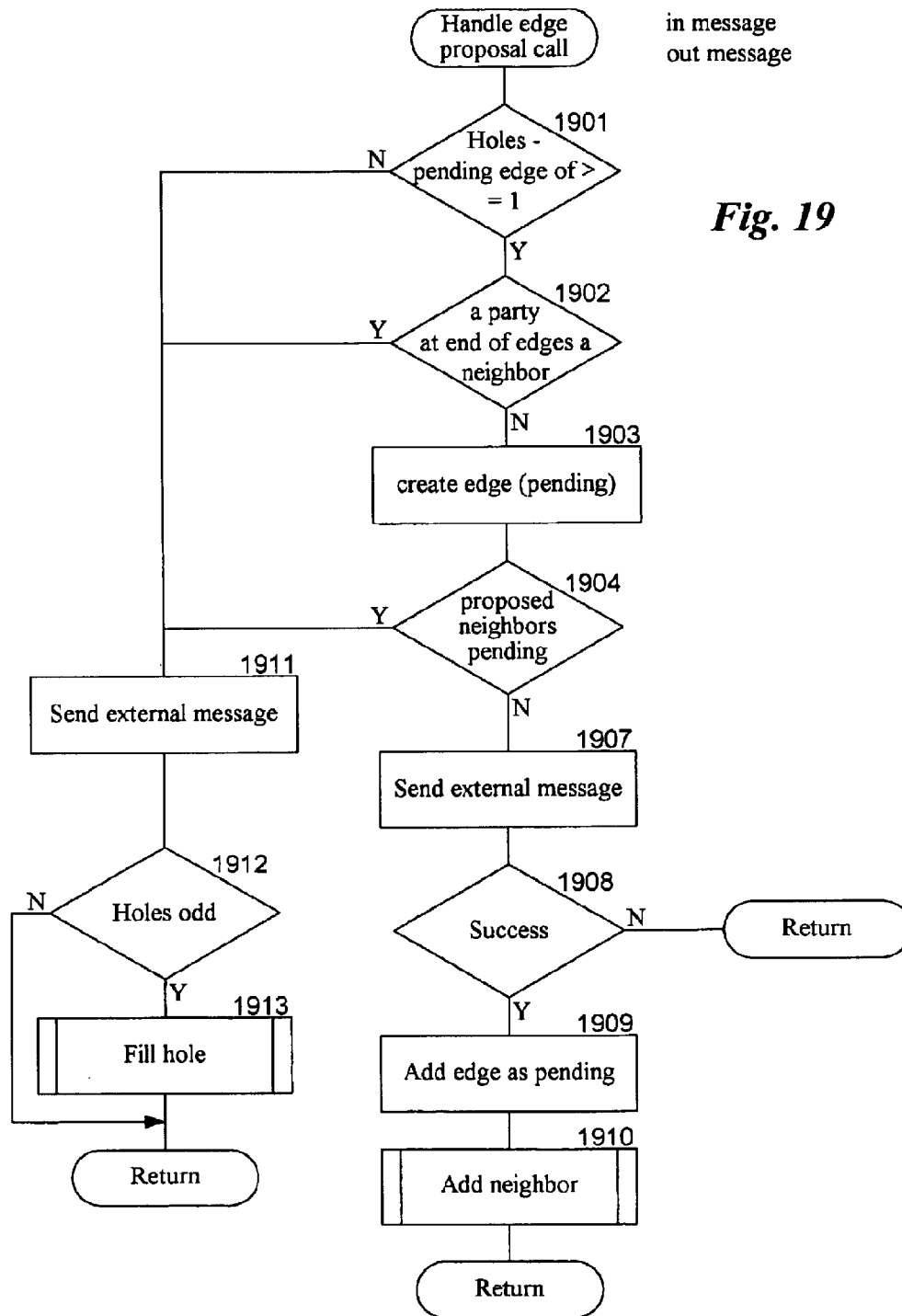


Fig. 19

Fig. 20

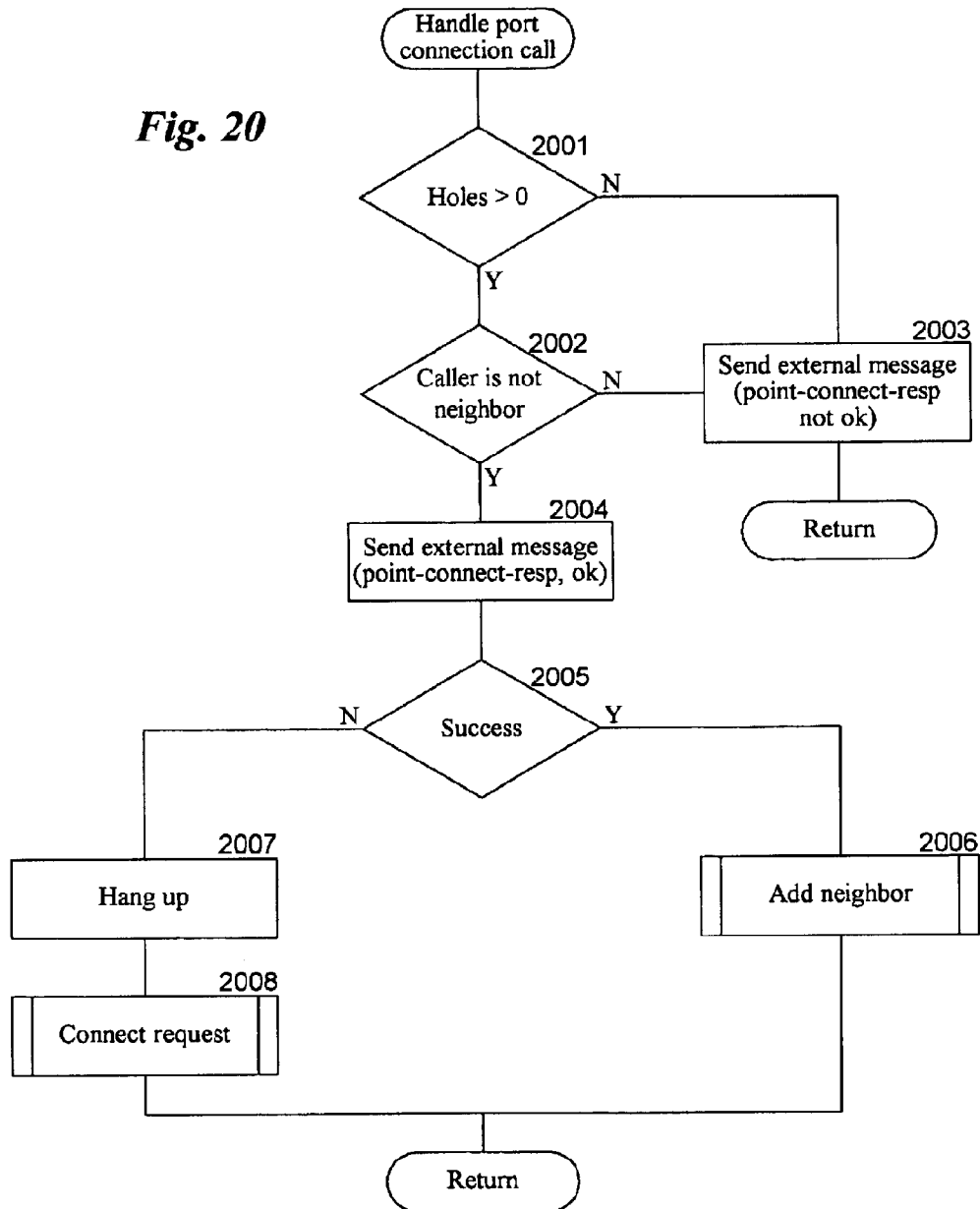


Fig. 21

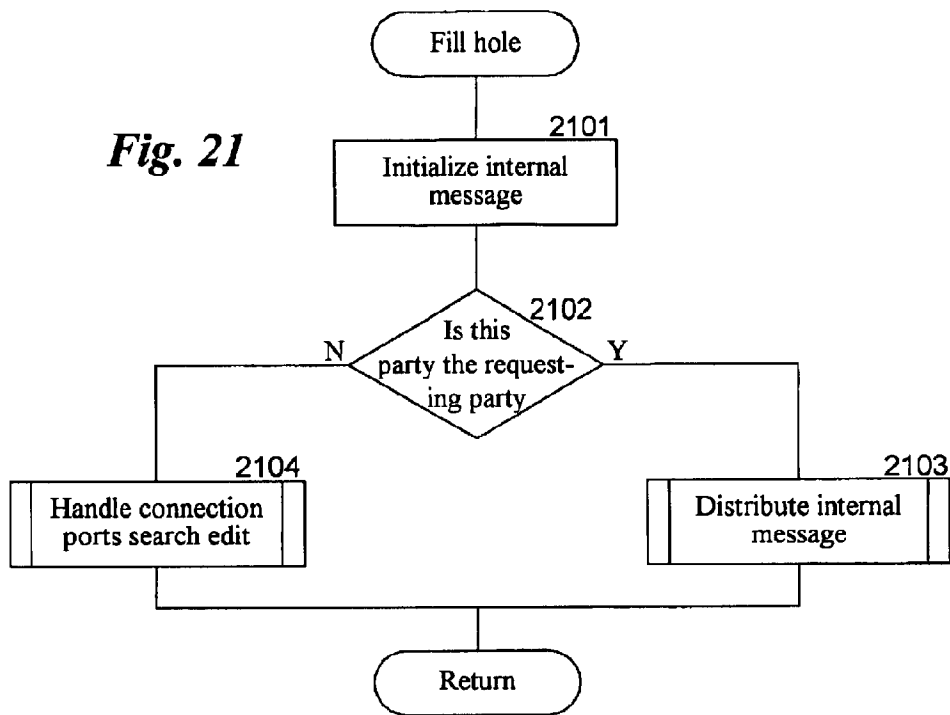


Fig. 22

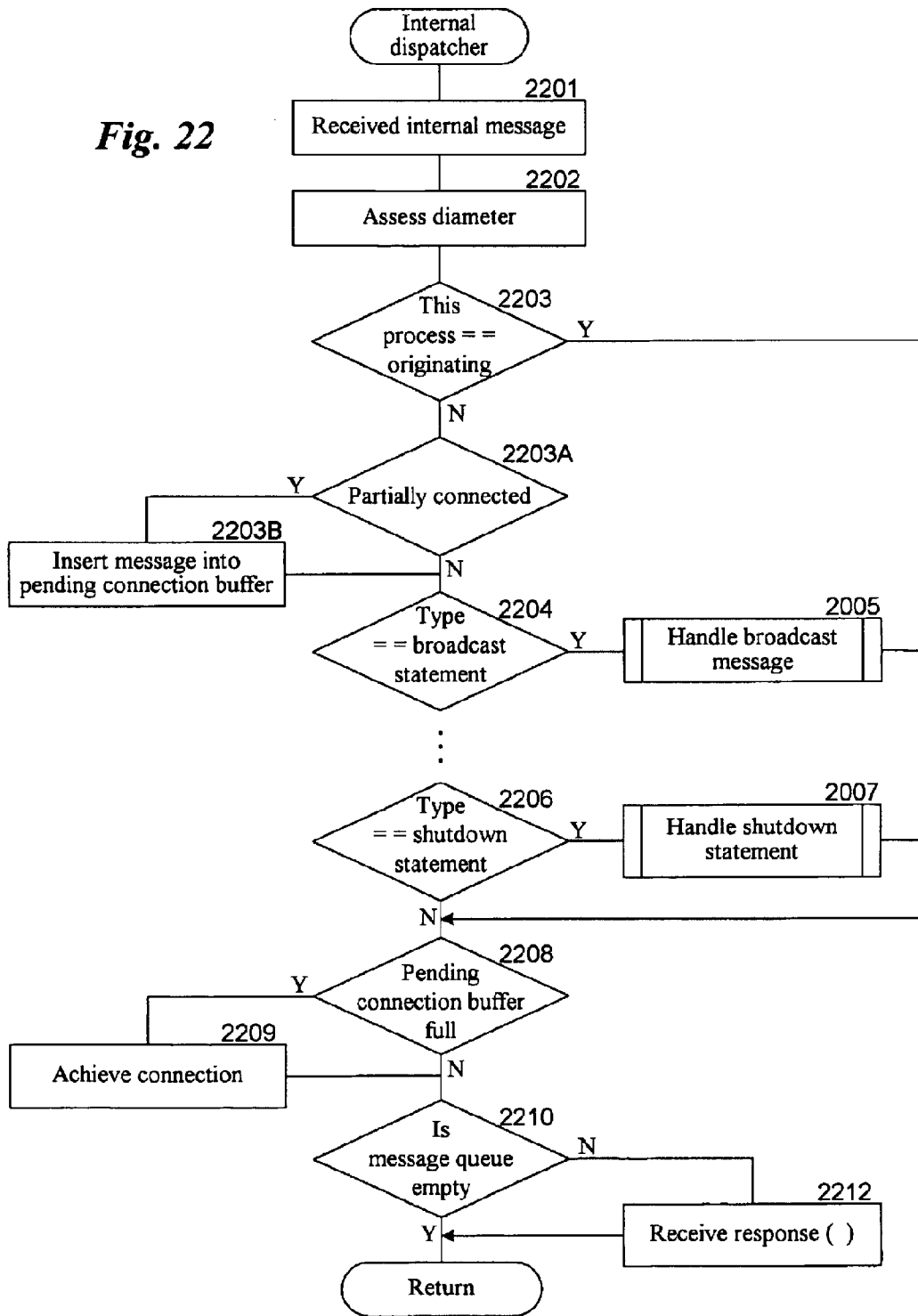


Fig. 23

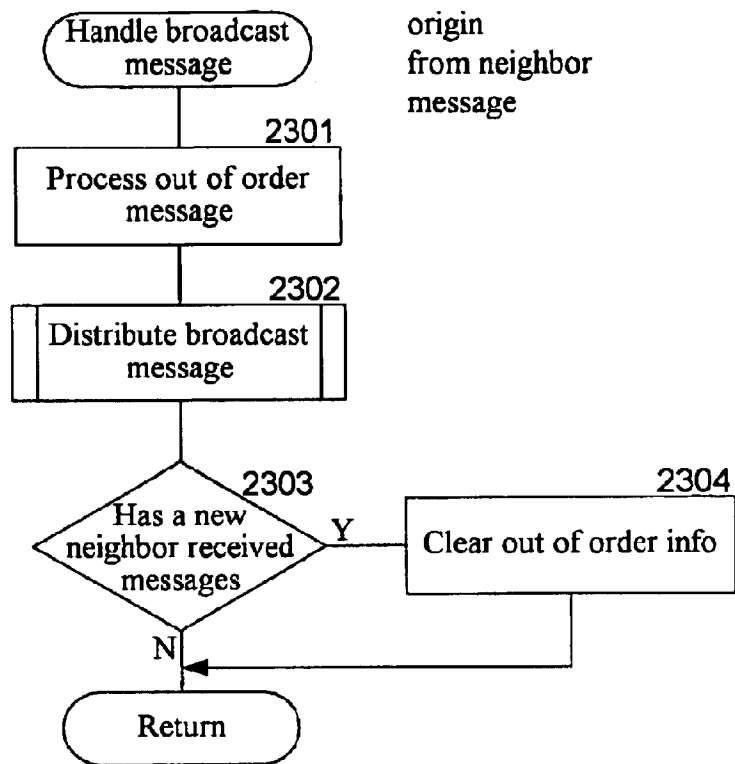
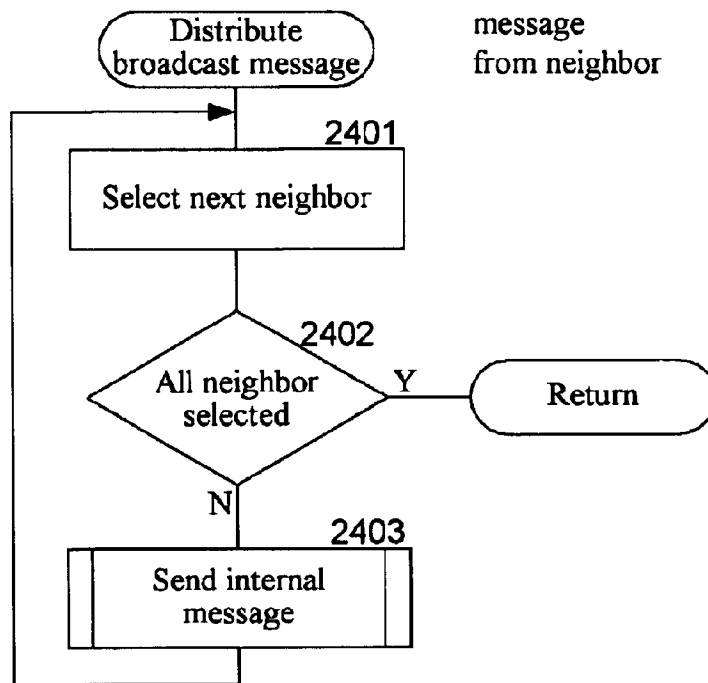


Fig. 24



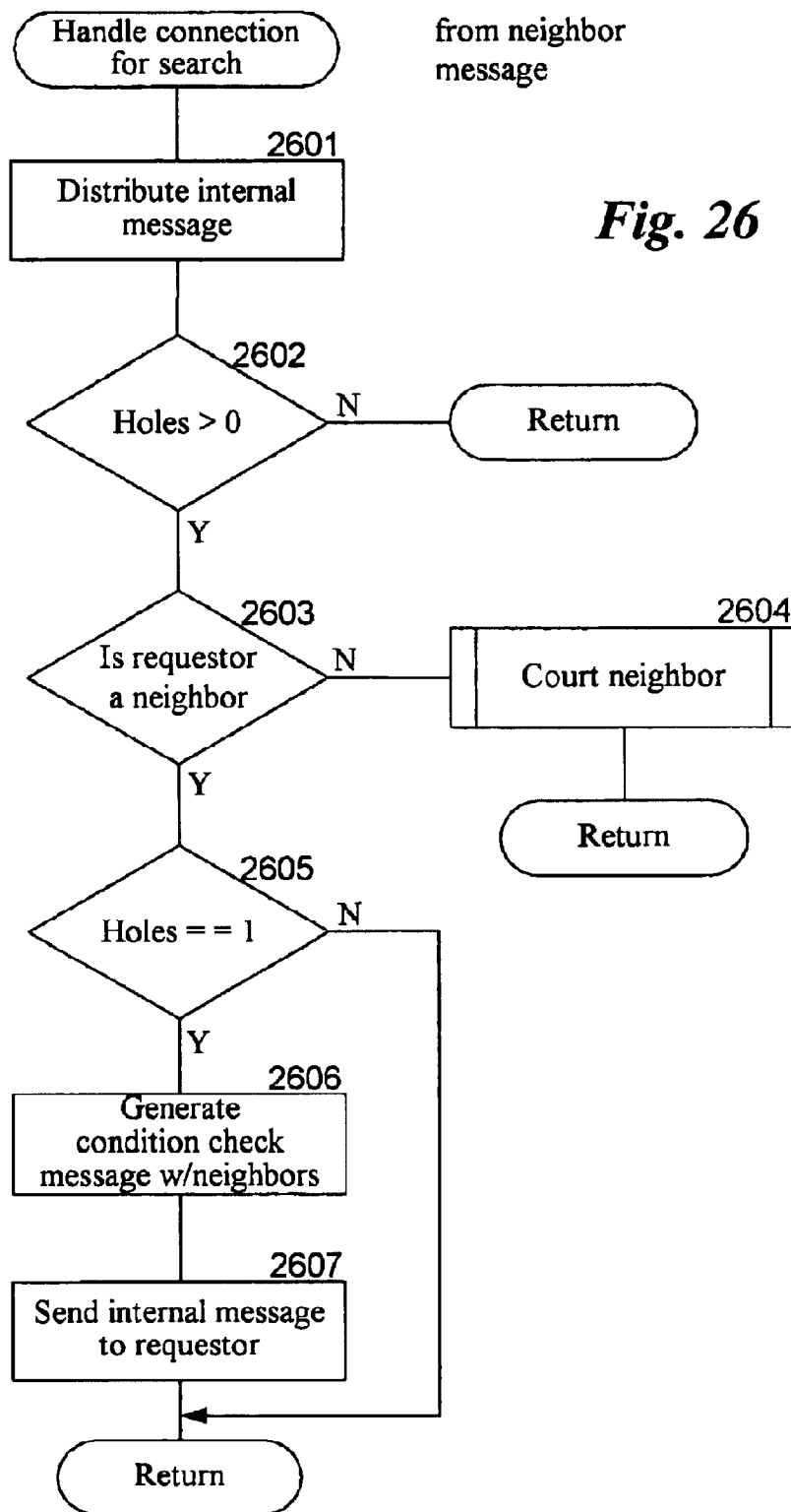


Fig. 27

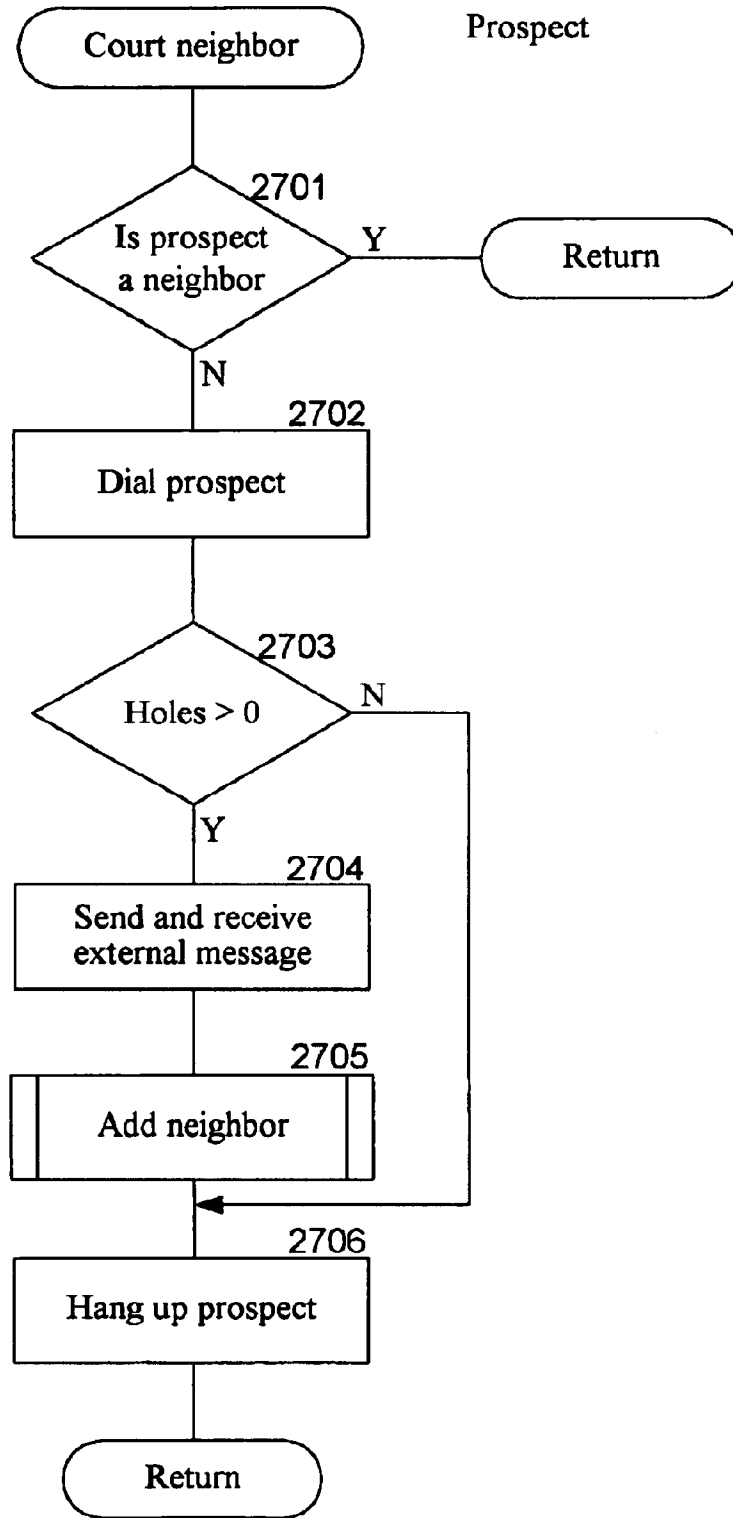


Fig. 28

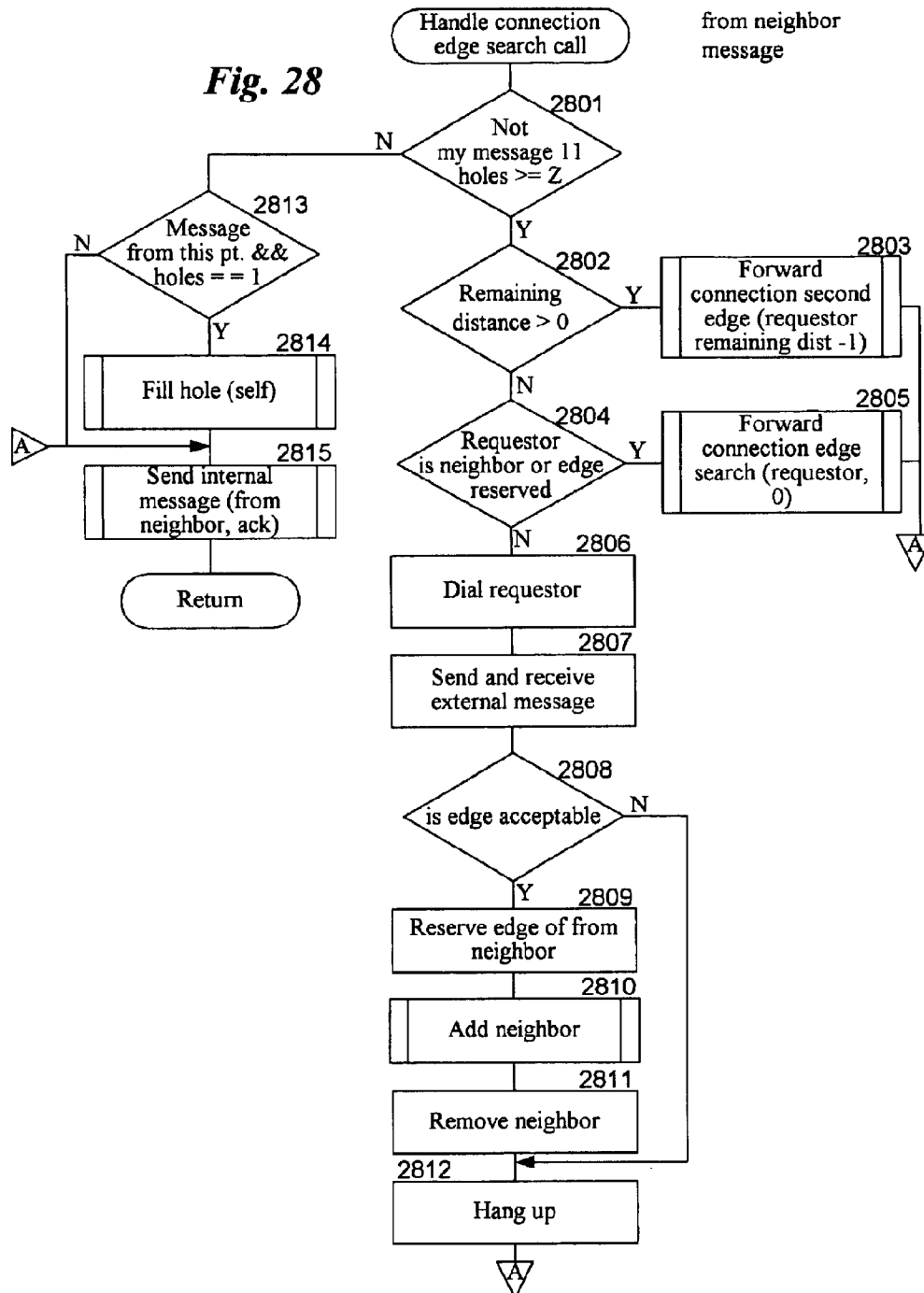


Fig. 29

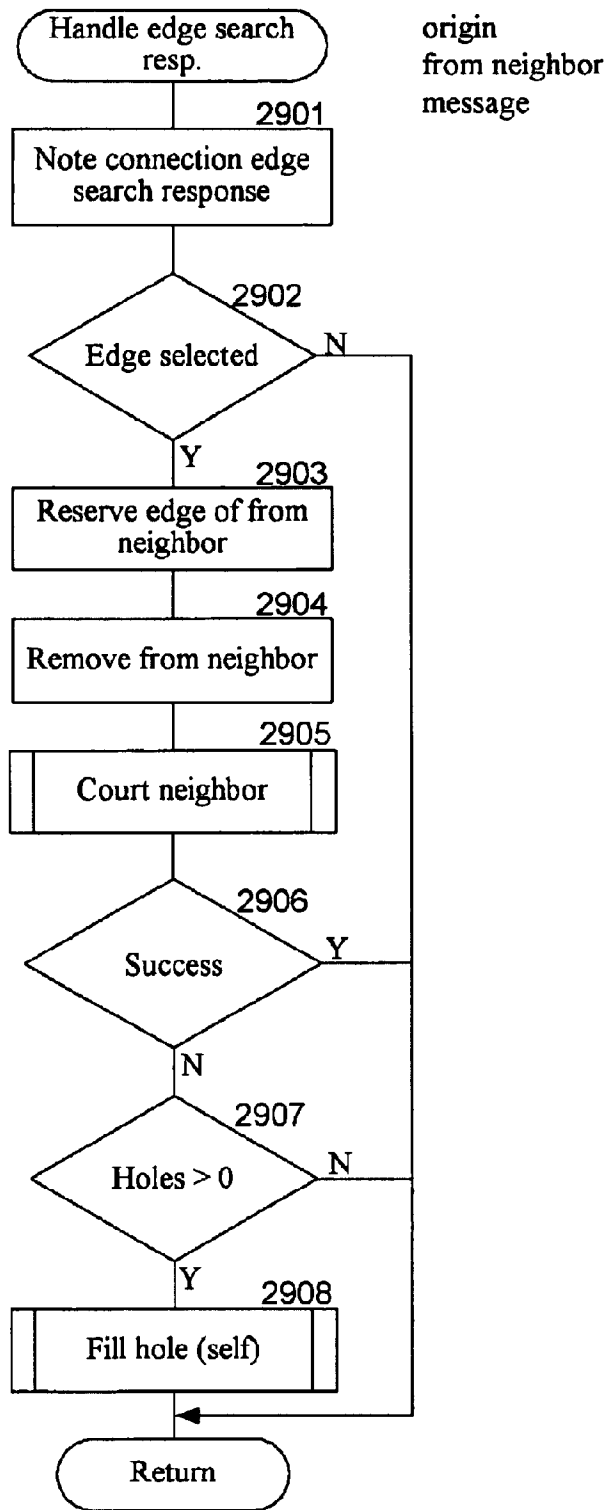


Fig. 30

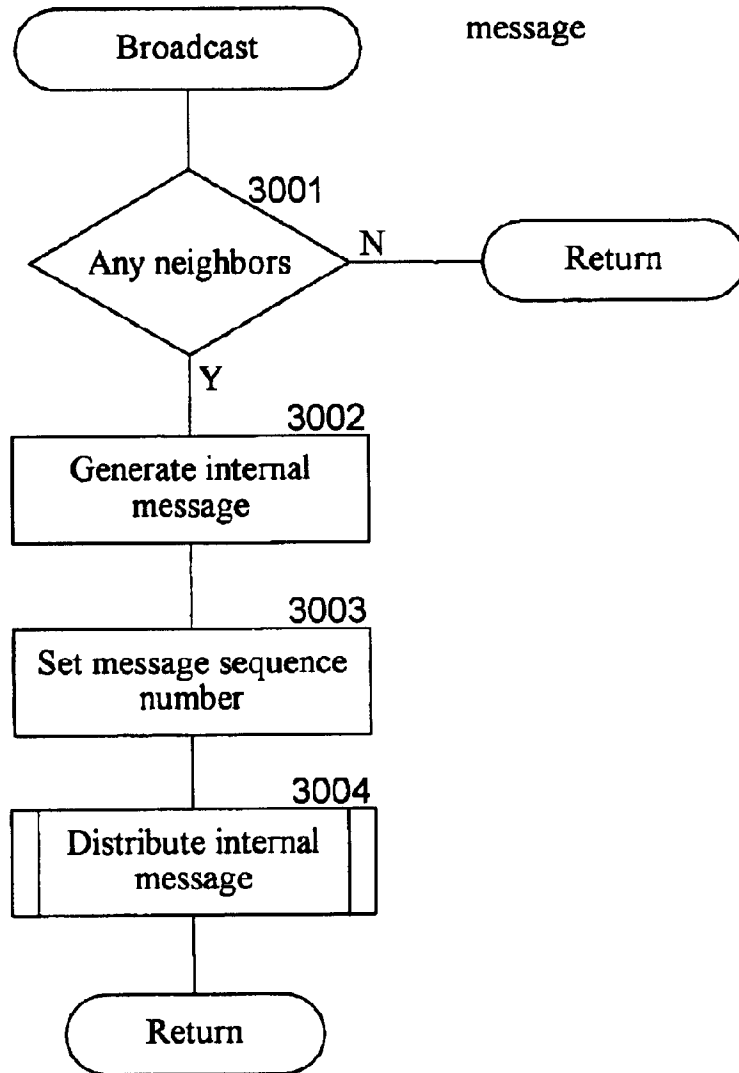


Fig. 31

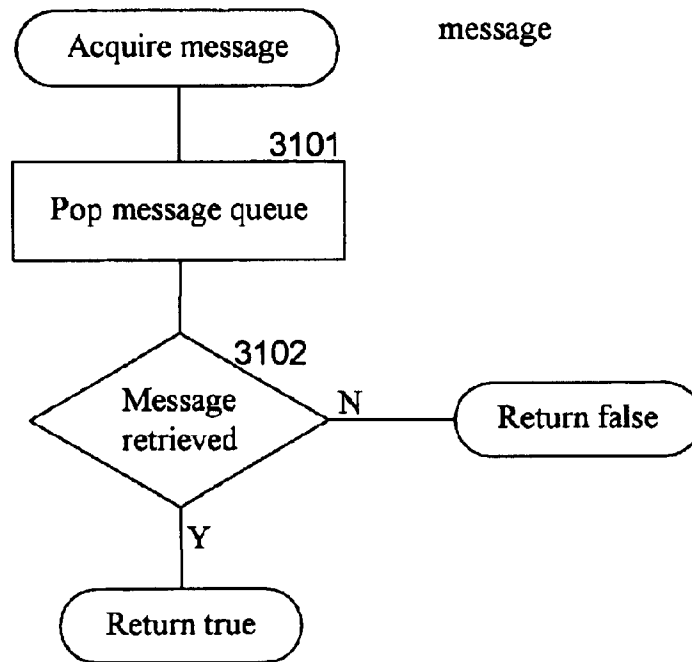


Fig. 32

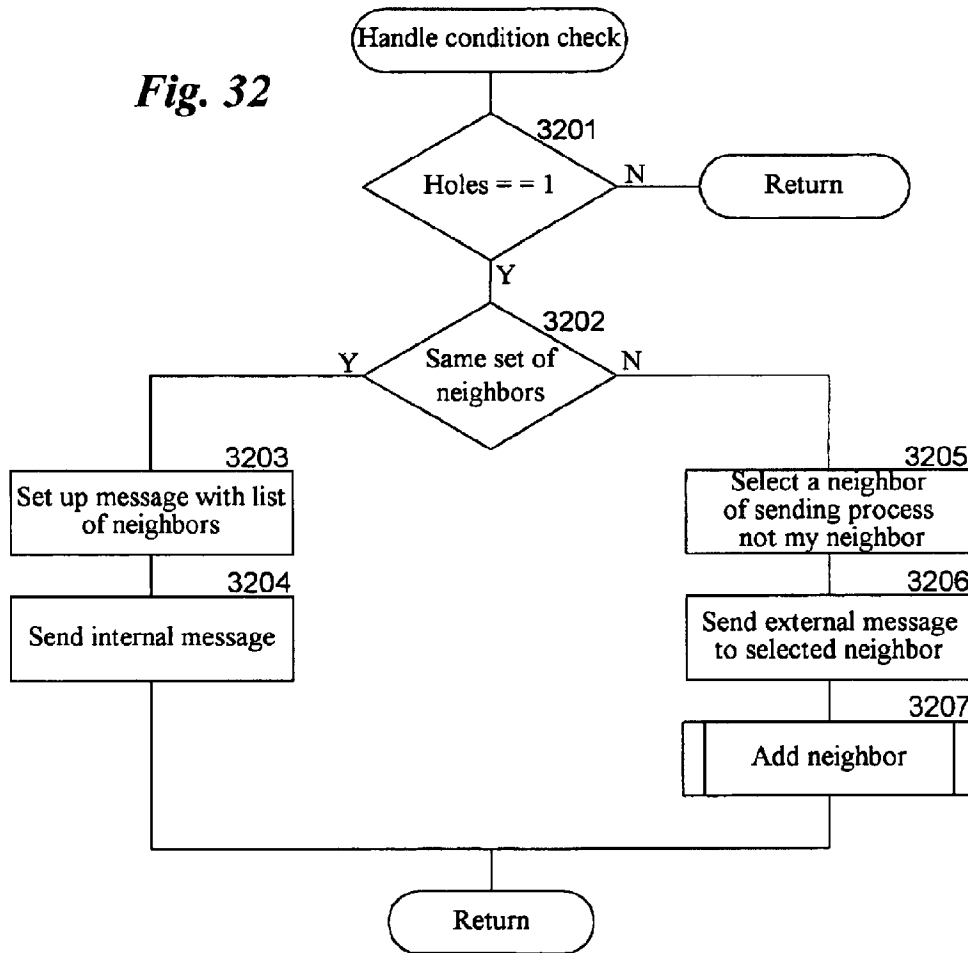


Fig. 33

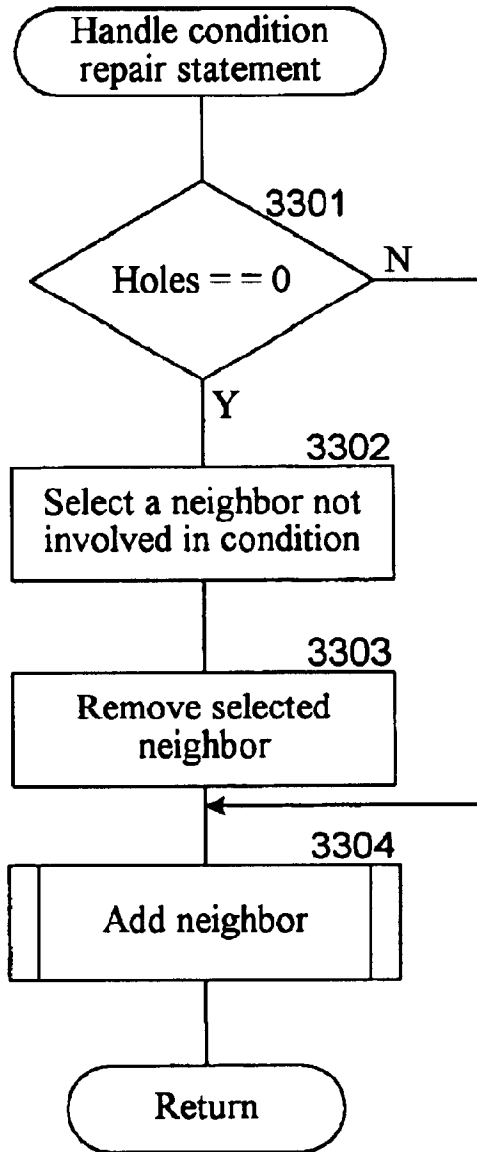
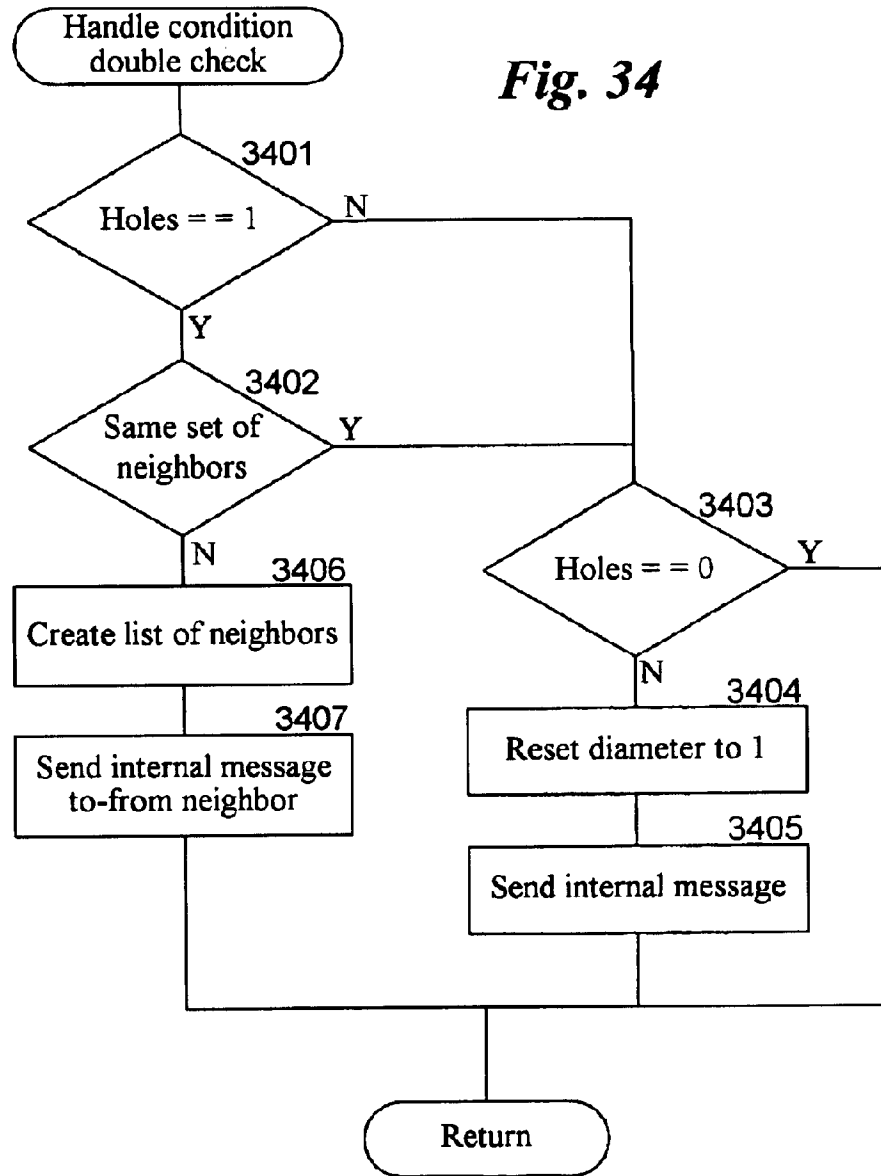


Fig. 34



US 6,732,147 B1

1

LEAVING A BROADCAST CHANNEL**CROSS-REFERENCE TO RELATED APPLICATIONS**

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

TECHNICAL FIELD

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

BACKGROUND

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

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

The client/server middleware systems provide a server that coordinates the communications between the various clients who are sharing the information. The server functions as a central authority for controlling access to shared resources. Examples of client/server middleware systems include remote procedure calls ("RPC"), database servers, and the common object request broker architecture ("CORBA"). Client/server middleware systems are not par-

2

ticularly well suited to sharing of information among many participants. In particular, when a client stores information to be shared at the server, each other client would need to poll the server to determine that new information is being shared. Such polling places a very high overhead on the communications network. Alternatively, each client may register a callback with the server, which the server then invokes when new information is available to be shared. Such a callback technique presents a performance bottleneck because a single server needs to call back to each client whenever new information is to be shared. In addition, the reliability of the entire sharing of information depends upon the reliability of the single server. Thus, a failure at a single computer (i.e., the server) would prevent communications between any of the clients.

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

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

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

BRIEF DESCRIPTION OF THE DRAWINGS

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

DETAILED DESCRIPTION

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

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

FIG. 1 illustrates a graph that is 4-regular and 4-connected which represents the broadcast channel. Each of the nine nodes A-I represents a computer that is connected to the broadcast channel, and each of the edges represents an "edge" connection between two computers of the broadcast channel. The time it takes to broadcast a message to each computer on the broadcast channel depends on the speed of the connections between the computers and the number of connections between the originating computer and each other computer on the broadcast channel. The minimum number of connections that a message would need to traverse between each pair of computers is the "distance" between the computers (i.e., the shortest path between the two nodes of the graph). For example, the distance between computers A and F is one because computer A is directly connected to computer F. The distance between computers A and B is two because there is no direct connection between computers A and B, but computer F is directly connected to

computer B. Thus, a message originating at computer A would be sent directly to computer F, and then sent from computer F to computer B. The maximum of the distances between the computers is the “diameter” of broadcast channel. The diameter of the broadcast channel represented by FIG. 1 is two. That is, a message sent by any computer would traverse no more than two connections to reach every other computer. FIG. 2 illustrates a graph representing 20 computers connected to a broadcast channel. The diameter of this broadcast channel is 4. In particular, the shortest path between computers 1 and 3 contains four connections (1–12, 12–15, 15–18, and 18–3).

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

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

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

D are the two pairs that are identified as the neighbors for the new computer Z. The connections between each of these pairs is broken, and a connection between computer Z and each of computers B, C, D, and E is established as indicated by FIG. 3B. The process of breaking the connection between two neighbors and reconnecting each of the former neighbors to another computer is referred to as “edge pinning” as the edge between two nodes may be considered to be stretched and pinned to a new node.

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

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

A seeking computer could connect to the broadcast channel by connecting to computers either directly connected to the found portal computer or directly connected to one of its neighbors. A possible problem with such a scheme for identifying the neighbors for the seeking computer is that the

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

Broadcasting Through the Graph

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

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

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

receiving computer. The second message only has to travel a distance of one. Thus, it is possible for the second message to reach the receiving computer before the first message.

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

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

Decomposing the Graph

A connected computer disconnects from the broadcast channel either in a planned or unplanned manner. When a

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

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

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

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

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

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

FIGS. 5E and 5F further illustrate the neighbors with empty ports condition. FIG. 5E illustrates the neighbors with

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

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

Port Selection

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

channel type and channel instance, it generates the same port ordering. As described below, it is possible for a computer to be connected to multiple broadcast channels that are uniquely identified by channel type and channel instance. The algorithm may be "seeded" with channel type and channel instance in order to generate a unique ordering of port numbers for each broadcast channel. Thus, a seeking computer will dial the ports of a portal computer in the same order as the portal computer used when allocating its call-in port.

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

Locating a Portal Computer

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

When a seeking computer locates a portal computer that is itself not fully connected, the two computers do not connect when they first locate each other because the

broadcast channel may already be established and accessible through a higher-ordered port number on another portal computer. If the two seeking computers were to connect to each other, then two disjoint broadcast channels would be formed. Each seeking computer can share its experience in trying to locate a portal computer with the other seeking computer. In particular, if one seeking computer has searched all the portal computers to a depth of eight, then the one seeking computer can share that it has searched to a depth of eight with another seeking computer. If that other seeking computer has searched to a depth of, for example, only four, it can skip searching through depths five through eight and that other seeking computer can advance its searching to a depth of nine.

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

Identifying Neighbors for a Seeking Computer

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

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

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

the forwarding computer toggles the new distance to travel between zero and one to help prevent two computers from sending the message back and forth between each other.

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

External Data Representation

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

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

M-Regular.

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

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

Components

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

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

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

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

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

The following tables list messages sent by the broadcaster components.

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

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

-continued

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

Flow Diagrams

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

routine continues at block 808. In block 806, the routine invokes the achieve connection routine to change the state of this process to fully connected. In block 807, the routine installs the external dispatcher for processing messages received through this process' external port for the passed channel type and channel instance. When a message is received through that external port, the external dispatcher is invoked. The routine then returns. In block 808, the routine installs an external dispatcher. In block 809, the routine invokes the connect request routine to initiate the process of identifying neighbors for the seeking computer. The routine then returns.

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

FIG. 10 is a flow diagram illustrating the processing of the contact process routine in one embodiment. This routine determines whether the process of the selected portal computer that answered the call-in to the selected port is fully connected to the broadcast channel. In block 1001, the routine sends an external message (i.e., seeking_connection_call) to the answering process indicating that a seeking process wants to know whether the answering process is fully connected to the broadcast channel. In block 1002, the routine receives the external response message

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

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

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

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

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

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

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

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

filled is greater than or equal to two, then the routine continues at block 1614, else the routine continues at block 1616. In block 1614, the routine invokes the forward connection edge search routine. The invoked routine is passed to an indication of the calling process and the random walk distance. In one embodiment, the distance is twice in the estimated diameter of the broadcast channel. In block 1614, the routine decrements the holes left to fill by two and loops to block 1613. In decision block 1616, if there is still a hole to fill, then the routine continues at block 1617, else the routine returns. In block 1617, the routine invokes the fill hole routine passing the identification of the calling process. The fill hole routine broadcasts a connection port search statement (i.e., connection_port_search_stmt) for a hole of a connected process through which the calling process can connect to the broadcast channel. The routine then returns.

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

FIG. 18 is a flow diagram illustrating the processing of the forward connection edge search routine in one embodiment. This routine is responsible for passing along a request to connect a requesting process to a randomly selected neighbor of this process through the internal port of the selected neighbor, that is part of the random walk. In decision block 1801, if the forwarding distance remaining is greater than zero, then the routine continues at block 1804, else the routine continues at block 1802. In decision block 1802, if the number of neighbors of this process is greater than one, then the routine continues at block 1804, else this broadcast

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

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

FIG. 20 is a flow diagram illustrating the processing of the handle port connection call routine in one embodiment. This routine is invoked when an external message is received then indicates that the sending process wants to connect to one hole of this process. In decision block 2001, if the number of holes of this process is greater than zero, then the

routine continues at block 2002, else the routine continues at block 2003. In decision block 2002, if the sending process is not a neighbor, then the routine continues at block 2004, else the routine continues to block 2003. In block 2003, the routine sends a port connection response external message (i.e., `port_connection_resp`) to the sending process that indicates that it is not okay to connect to this process. The routine then returns. In block 2004, the routine sends a port connection response external message to the sending process that indicates that it is okay to connect to this process. In decision block 2005, if the sending of the message was successful, then the routine continues at block 2006, else the routine continues at block 2007. In block 2006, the routine invokes the add neighbor routine to add the sending process as a neighbor of this process and then returns. In block 2007, the routine hangs up the external connection. In block 2008, the routine invokes the connect request routine to request that a process connect to one of the holes of this process. The routine then returns.

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

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

the routine invokes the achieve connection routine and then continues in block 2210. In decision block 2210, if the application program message queue is empty, then the routine returns, else the routine continues at block 2212. In block 2212, the routine invokes the receive response routine passing the acquired message and then returns. The received response routine is a callback routine of the application program.

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

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

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

FIG. 27 is a flow diagram illustrating the processing of the court neighbor routine in one embodiment. This routine is passed an indication of the prospective neighbor for this process. If this process can connect to the prospective neighbor, then it sends a port connection call external message to the prospective neighbor and adds the prospective neighbor as a neighbor. In decision block 2701, if the prospective neighbor is already a neighbor, then the routine returns, else the routine continues at block 2702. In block 2702, the routine dials the prospective neighbor. In decision block 2703, if the number of holes of this process is greater than zero, then the routine continues at block 2704, else the

routine continues at block 2706. In block 2704, the routine sends a port connection call external message (i.e., port_connection_call) to the prospective neighbor and receives its response (i.e., port_connection_rsp). Assuming the response is successfully received, in block 2705, the routine adds the prospective neighbor as a neighbor of this process by invoking the add neighbor routine. In block 2706, the routine hangs up with the prospect and then returns.

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

FIG. 29 is a flow diagram illustrating the processing of the handle connection edge search response routine in one embodiment. This routine is passed as indication of the requesting process, the sending neighbor, and the message. In block 2901, the routine notes that the connection edge

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

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

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

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

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

ment. This routine removes an existing neighbor and connects to the process that sent the message. In decision block 3301, if this process has no holes, then the routine continues at block 3302, else the routine continues at block 3304. In block 3302, the routine selects a neighbor that is not involved in the neighbors with empty ports condition. In block 3303, the routine removes the selected neighbor as a neighbor of this process. Thus, this process that is executing the routine now has at least one hole. In block 3304, the routine invokes the add neighbor routine to add the process that sent the message as a neighbor of this process. The routine then returns.

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

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

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

We claim:

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

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

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

29

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

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

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

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

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

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

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

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

7. The method of claim 6 including:

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

8. The method of claim 7 including:

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

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

30

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

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

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

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

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

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

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

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

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

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

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

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

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

Page 1 of 2

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

Column 5,

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

Column 6,

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

Column 8,

Line 26, delete comma between "newly";

Column 11,

Line 60, "port-number" should be -- port number --;

Line 63, "port-order" should be -- port order --;

Column 13,

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

Column 14,

Line 51, delete period after "Regular";

Column 22,

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

Column 23,

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

Column 25,

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

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

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

Page 2 of 2

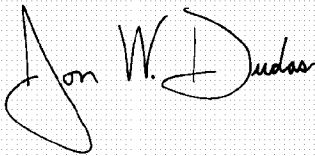
It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 25 (cont'd),

Line 46, delete comma between "2604" and "In";

Signed and Sealed this

Twenty-seventh Day of July, 2004

A handwritten signature in black ink on a light gray dotted background. The signature reads "Jon W. Dudas" in a cursive style.

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

(12) **INTER PARTES REVIEW CERTIFICATE** (1163rd)

United States Patent
Holt et al.

(10) **Number:** US **6,732,147 K1**
(45) **Certificate Issued:** **May 1, 2019**

(54) **LEAVING A BROADCAST CHANNEL**

(75) **Inventors:** **Fred B. Holt; Virgil E. Bourassa**

(73) **Assignee:** **ACCELERATION BAY, LLC**

Trial Number:

IPR2016-00747 filed Mar. 12, 2016

Inter Partes Review Certificate for:

Patent No.: **6,732,147**
Issued: **May 4, 2004**
Appl. No.: **09/629,577**
Filed: **Jul. 31, 2000**

The results of IPR2016-00747 are reflected in this inter partes review certificate under 35 U.S.C. 318(b).

INTER PARTES REVIEW CERTIFICATE

U.S. Patent 6,732,147 K1

Trial No. IPR2016-00747

Certificate Issued May 1, 2019

1

2

AS A RESULT OF THE INTER PARTES
REVIEW PROCEEDING, IT HAS BEEN
DETERMINED THAT:

Claims **6-10** are found patentable.

5

* * * * *

EXHIBIT 22



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

(10) **Patent No.:** **US 6,910,069 B1**
 (45) **Date of Patent:** **Jun. 21, 2005**

- | | | | |
|---|---------------|---------|---------------------------|
| (54) JOINING A BROADCAST CHANNEL | 5,696,903 A | 12/1997 | Mahany |
| (75) Inventors: Fred B. Holt , Seattle, WA (US); Virgil E. Bourassa , Bellevue, WA (US) | 5,732,074 A | 3/1998 | Spaur et al. |
| | 5,732,086 A * | 3/1998 | Liang et al. 370/410 |
| | 5,732,219 A | 3/1998 | Blumer et al. |
| | 5,734,865 A | 3/1998 | Yu |
| (73) Assignee: The Boeing Company , Seattle, WA (US) | 5,737,526 A | 4/1998 | Periasamy et al. |
| | 5,754,830 A | 5/1998 | Butts et al. |

(Continued)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 708 days.

(21) Appl. No.: **09/629,570**

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

(51) **Int. Cl.**⁷ **G06F 15/177**

(52) **U.S. Cl.** **709/221; 709/252; 709/243; 709/227**

(58) **Field of Search** **709/221, 220, 709/252, 243, 227, 223, 204, 238; 370/225, 260, 400; 455/428**

(56) **References Cited**

U.S. PATENT DOCUMENTS

- | | | | |
|---------------|---------|--------------------|----------|
| 4,912,656 A | 3/1990 | Cain et al. | |
| 5,056,085 A | 10/1991 | Vu | |
| 5,058,105 A | 10/1991 | Mansour et al. | |
| 5,079,767 A | 1/1992 | Perlman | |
| 5,099,235 A * | 3/1992 | Crookshanks | 455/13.1 |
| 5,101,480 A * | 3/1992 | Shin et al. | 710/317 |
| 5,117,422 A * | 5/1992 | Hauptschein et al. | 370/255 |
| 5,309,437 A | 5/1994 | Perlman et al. | |
| 5,345,558 A | 9/1994 | Opher et al. | |
| 5,426,637 A | 6/1995 | Derby et al. | |
| 5,459,725 A | 10/1995 | Bodner et al. | |
| 5,471,623 A * | 11/1995 | Napolitano, Jr. | 709/243 |
| 5,511,168 A | 4/1996 | Perlman et al. | |
| 5,535,199 A | 7/1996 | Amri et al. | |
| 5,568,487 A | 10/1996 | Sitbon et al. | |
| 5,636,371 A | 6/1997 | Yu | |
| 5,644,714 A | 7/1997 | Kikinis | |
| 5,673,265 A | 9/1997 | Gupta et al. | |

OTHER PUBLICATIONS

Cho et al., "A Flood Routing Method for Data Networks," Sep. 1997, Proceedings of 1997 International Conference on Information, Communications and Signal Processing, vol. 3, pp. 1418-1422.*
 Bandyopadhyay et al., "A Flexible Architecture for Multi-Hop Optical Networks," Oct. 1998, 7th International Conference on Computer Communications and Networks, 1998, pp. 472-478.*

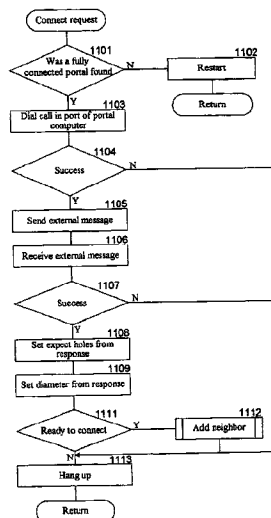
(Continued)

Primary Examiner—Glenton B. Burgess
Assistant Examiner—Bradley Edelman
 (74) *Attorney, Agent, or Firm*—Perkins Coie LLP

(57) **ABSTRACT**

A technique for adding a participant to a network is provided. This technique allows for the simultaneous sharing of information among many participants in a network without the placement of a high overhead on the underlying communication network. To connect to the broadcast channel, a seeking computer first locates a computer that is fully connected to the broadcast channel. The seeking computer then establishes a connection with a number of the computers that are already connected to the broadcast channel. The technique for adding a participant to a network includes identifying a pair of participants that are connected to the network, disconnecting the participants of the identified pair from each other, and connecting each participant of the identified pair of participants to the added participant.

17 Claims, 39 Drawing Sheets



US 6,910,069 B1

Page 2

U.S. PATENT DOCUMENTS

5,757,795 A 5/1998 Schnell
5,761,425 A 6/1998 Miller
5,764,756 A 6/1998 Onweller
5,790,548 A 8/1998 Sistanizadeh et al.
5,790,553 A 8/1998 Deaton, Jr. et al.
5,799,016 A 8/1998 Onweller
5,802,285 A 9/1998 Hirviniemi
5,850,592 A 12/1998 Ramanathan
5,864,711 A 1/1999 Mairs et al.
5,867,660 A 2/1999 Schmidt et al.
5,867,667 A 2/1999 Butman et al.
5,870,605 A 2/1999 Bracho et al.
5,874,960 A 2/1999 Mairs et al.
5,899,980 A 5/1999 Wilf et al.
5,907,610 A 5/1999 Onweller
5,925,097 A 7/1999 Gopinath et al.
5,928,335 A 7/1999 Morita
5,935,215 A 8/1999 Bell et al.
5,946,316 A 8/1999 Chen et al.
5,948,054 A 9/1999 Nielsen
5,949,975 A 9/1999 Batty et al.
5,953,318 A 9/1999 Nattkemper et al.
5,956,484 A 9/1999 Rosenberg et al.
5,970,232 A 10/1999 Passint et al.
5,974,043 A 10/1999 Solomon
5,987,506 A 11/1999 Carter et al.
6,003,088 A 12/1999 Houston et al.
6,013,107 A 1/2000 Blackshear et al.
6,023,734 A 2/2000 Ratcliff et al.
6,029,171 A 2/2000 Smiga et al.
6,032,188 A 2/2000 Mairs et al.
6,038,602 A 3/2000 Ishikawa
6,047,289 A 4/2000 Thorne et al.
6,065,063 A * 5/2000 Abali 709/242
6,073,177 A 6/2000 Hebel et al.
6,094,676 A 7/2000 Gray et al.
6,115,580 A 9/2000 Chuprun et al.
6,151,633 A 11/2000 Hurst
6,167,432 A 12/2000 Jiang
6,173,314 B1 1/2001 Kurashima et al.
6,195,366 B1 2/2001 Kayashima
6,199,116 B1 3/2001 May et al.
6,216,177 B1 4/2001 Mairs et al.
6,223,212 B1 4/2001 Batty et al.
6,243,691 B1 6/2001 Fisher et al.
6,252,884 B1 6/2001 Hunter
6,268,855 B1 7/2001 Mairs et al.
6,269,080 B1 7/2001 Kumar
6,271,839 B1 8/2001 Mairs et al.
6,272,548 B1 8/2001 Cotter et al.
6,285,363 B1 9/2001 Mairs et al.
6,304,928 B1 10/2001 Mairs et al.
6,321,270 B1 11/2001 Crawley
6,353,599 B1 3/2002 Bi et al.
6,415,270 B1 7/2002 Rackson et al.
6,434,622 B1 8/2002 Monteiro et al.
6,463,078 B1 10/2002 Engstrom et al.
6,490,247 B1 * 12/2002 Gilbert et al. 370/222
6,499,251 B2 12/2002 Weder
6,505,289 B1 * 1/2003 Han et al. 712/11
6,524,189 B1 2/2003 Rautila
6,553,020 B1 * 4/2003 Hughes et al. 370/347
6,603,742 B1 * 8/2003 Steele et al. 370/254
6,611,872 B1 8/2003 McCanne
6,618,752 B1 9/2003 Moore et al.
6,701,344 B1 3/2004 Holt et al.
2002/0027896 A1 3/2002 Hughes et al.

OTHER PUBLICATIONS

Hsu, "On Four-Connecting a Triconnected Graph," Oct. 1992, Annual Symposium on Foundations of Computer Science, 1992, pp. 70-79.*
Shiokawa et al., "Performance Analysis of Network Connective Probability of Multihop Network under Correlated Breakage," Jun. 1996, 1996 IEEE International Conference on Communications, vol. 3, pp. 1581-1585.*
Komine et al., "A Distributed Restoration Algorithm for Multiple-Link and Node Failures of Transport Networks," Dec. 199 IEEE Globecom '90, 'Communications: Connecting the Future,' vol. 1, pp. 459-463.*
U.S. Appl. No. 09/629,576, filed Jul. 31, 2000, Bourassa et al.
U.S. Appl. No. 09/629,577, filed Jul. 31, 2000, Bourassa et al.
U.S. Appl. No. 09/629,575, filed Jul. 31, 2000, Bourassa et al.
U.S. Appl. No. 09/629,572, filed Jul. 31, 2000, Bourassa et al.
U.S. Appl. No. 09/629,023, filed Jul. 31, 2000, Bourassa et al.
U.S. Appl. No. 09/629,043, filed Jul. 31, 2000, Bourassa et al.
U.S. Appl. No. 09/629,024, filed Jul. 31, 2000, Bourassa et al.
U.S. Appl. No. 09/629,042, filed Jul. 31, 2000, Bourassa et al.
Murphy, Patricia, A., "The Next Generation Networking Paradigm: Producer/Consumer Model," Dedicated Systems Magazine—2000 (pp. 26-28).
The Gamer's Guide, "First-Person Shooters," Oct. 20, 1998 (4 pages).
The O'Reilly Network, "Gnutella: Alive, Well, and Changing Fast," Jan. 25, 2001 (5 pages) <http://www.open2p.com/1pt/> . . . [Accessed Jan. 29, 2002].
Oram, Andy, "Gnutella and Freenet Represents True Technological Innovation," May 12, 2000 (7 pages) The O'Reilly Network <http://www.oreillynet.com/1pt/> . . . [Accessed Jan. 29, 2003].
Internetworking Technologies Handbook, Chapter 43 (pp. 43-1-43-16).
Oram, Andy, "Peer-to-Peer Makes the Internet Interesting Again," Sep. 22, 2000 (7 pages) The O'Reilly Network <http://linux.oreillynet.com/1pt/> . . . [Accessed Jan. 29, 2002].
Monte, Richard, "The Random Walk for Dummies," MIT Undergraduate Journal of Mathematics (pp. 143-148).
Srinivasan, R., "XDR: External Data Representation Standard," Sun Microsystems, Aug. 1995 (20 pages) Internet RFC/STD/FYI/BCP Archives <http://www.faqs.org/rfcs/rfc1832.html> [Accessed Jan. 29, 2002].
ADatabeam Corporate White Paper, "A Primer on the T.120 Series Standards," Copyright 1995 (pp. 1-16).
Kessler, Gary, C., "An Overview of TCP/IP Protocols and the Internet," Apr. 23, 1999 (23 pages) Hill Associates, Inc. <http://www.hill.com/library/publications/t/> . . . [Accessed Jan. 29, 2002].
Bondy, J.A., and Murty, U.S.R., "Graph Theory with Applications," Chapters 1-3 (pp. 1-47), 1976 American Elsevier Publishing Co., Inc., New York, New York.
Cormen, Thomas, H. et al., Introduction to Algorithms, Chapter 5.3 (pp. 84-91), Chapter 12 (pp. 218-243), Chapter 13 (p. 245), 1990, The MIT Press, Cambridge, Massachusetts, McGraw-Hill Book Company, New York.

US 6,910,069 B1

Page 3

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

The University of Warwick, Computer Science Open Days, "Demonstration on the Problems of Distributed Systems," <http://www.dcs.warwick.ac.uk> . . . [Accessed Jan. 29, 2002].
Alagar, S. and Venkatesan, S., "Reliable Broadcast in Mobile Wireless Networks," Department of Computer Science, University of Texas at Dallas, Military Communications Conference, 1995, MILCOM '95 Conference Record, IEEE San Diego, California, Nov. 5-8, 1995 (pp. 236-240).
International Search Report for The Boeing Company, International Patent Application No. PCT/US01/24240, Jun. 5, 2002 (7 pages).

Yavatkar et al., "A reliable Dissemination Protocol for Interactive Collaborative Applications," Proc. ACM Multimedia, 1995, p. 333-344; <http://citeseer.nj.nec.com/article/yavatkar95reliable.htm>.

Business Wire, "Boeing Panthesis Complete SWAN Transaction," Jul. 22, 2002, pp 1ff.

PR Newswire, "Microsoft Annouces Launch Date for UltraCrops, Its Second Premium Title for the Internet Gaming Zone," Mar. 27, 1998, pp1 ff.

PR Newswire, "Microsoft Boosts Accessibility to Internet Gaming Zone with Latest Release," Apr. 27, 1998, pp 1ff.

Peercy et al., "Distributed Algorithms for Shortest-Path, Deadlock-Free Routing and Broadcasting in Arbitrarily Faulty Hypercubes," Jun. 1990, 20th International Symposium on Fault-Tolerant Computing, 1990, pp-218-225.

Azar et al., "Routing Strategies for Fast Networks," May 1992, INFOCOM '92 Eleventh Annual Joint Conference of the IEEE Computer Communications Societies, vol. 1, 170-179###.

* cited by examiner

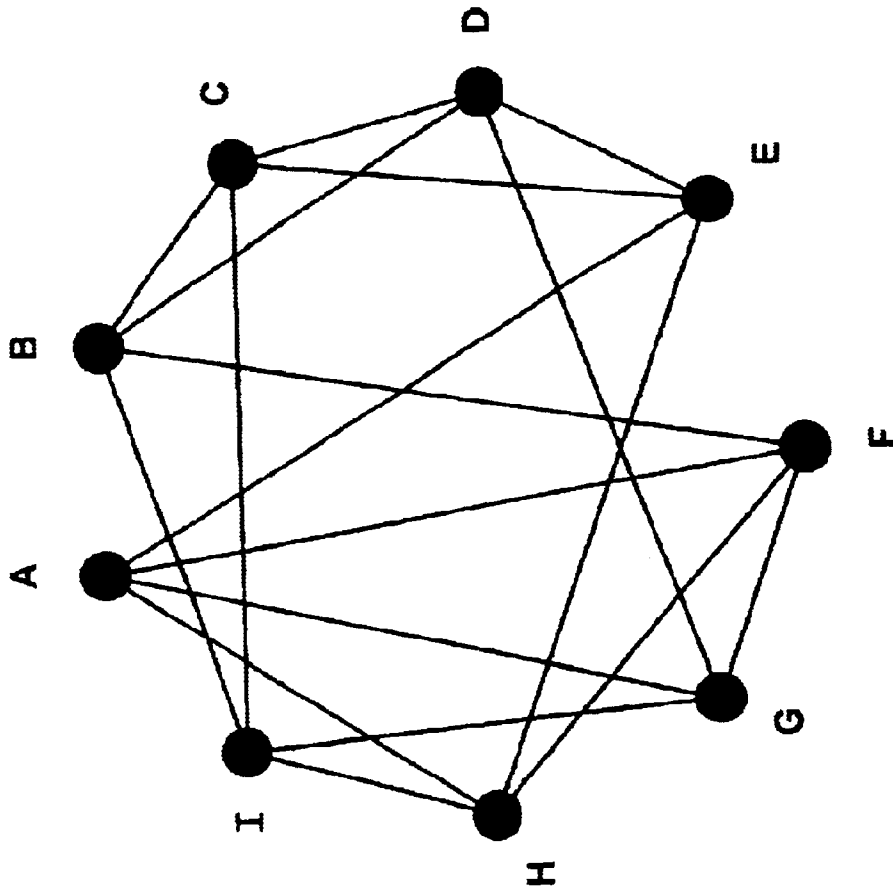


Fig. 1

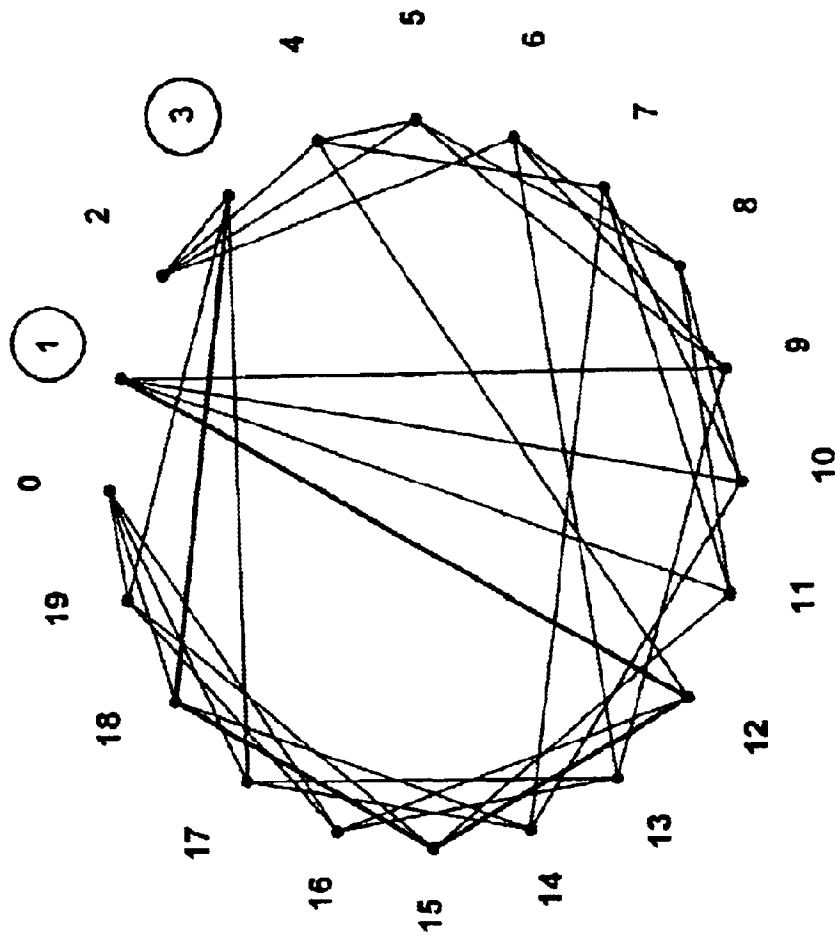


Fig. 2

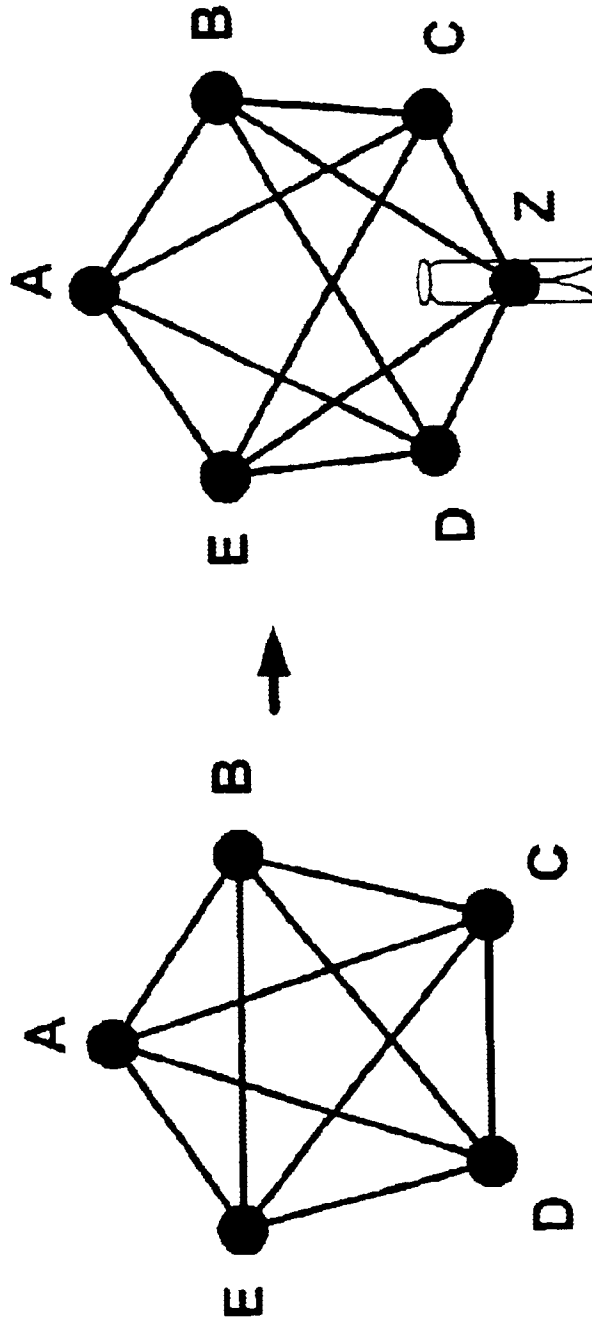


Fig. 3B

Fig. 3A

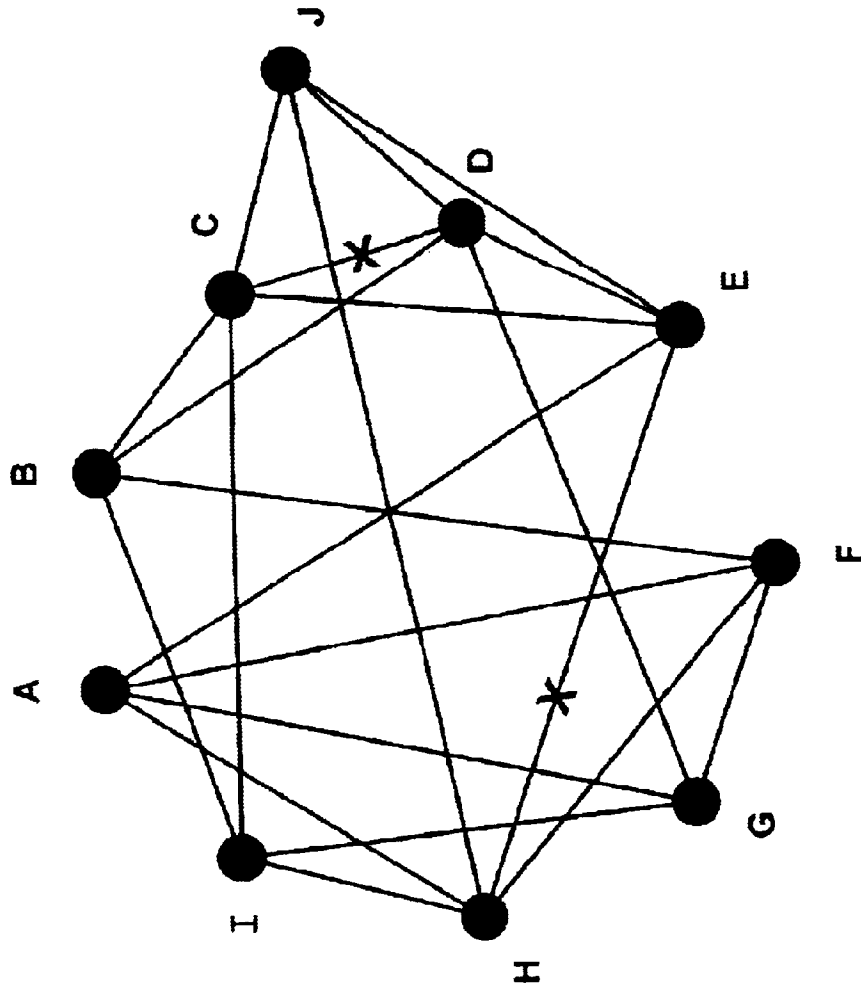


Fig. 4A

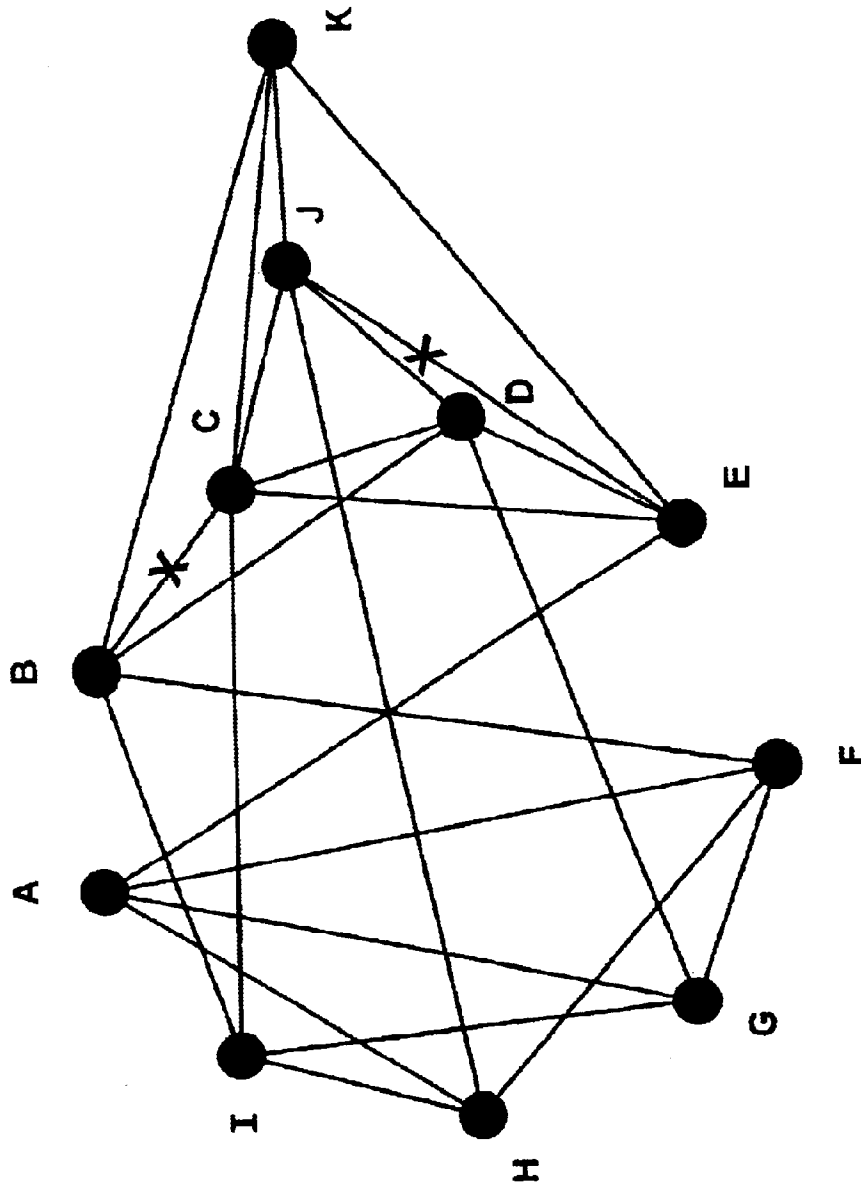


Fig. 4B

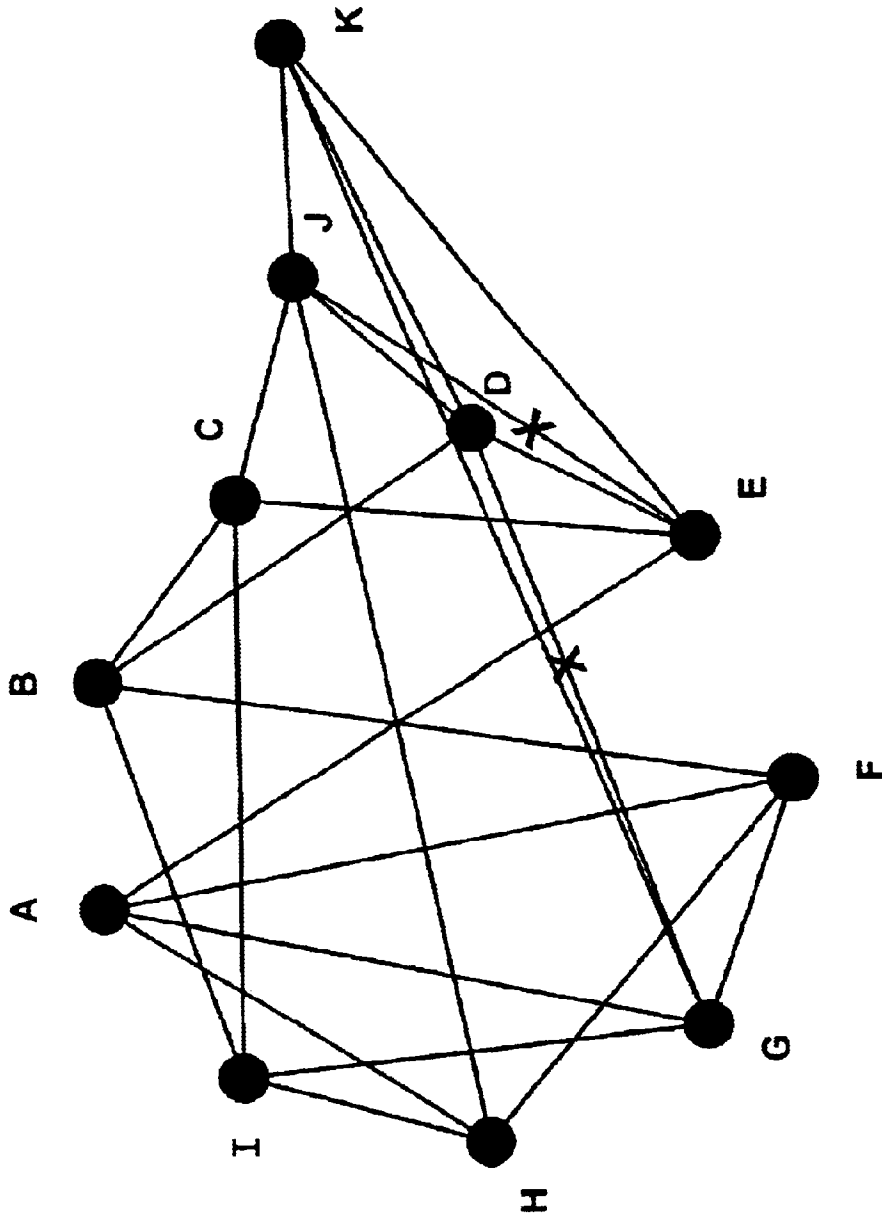


Fig. 4C

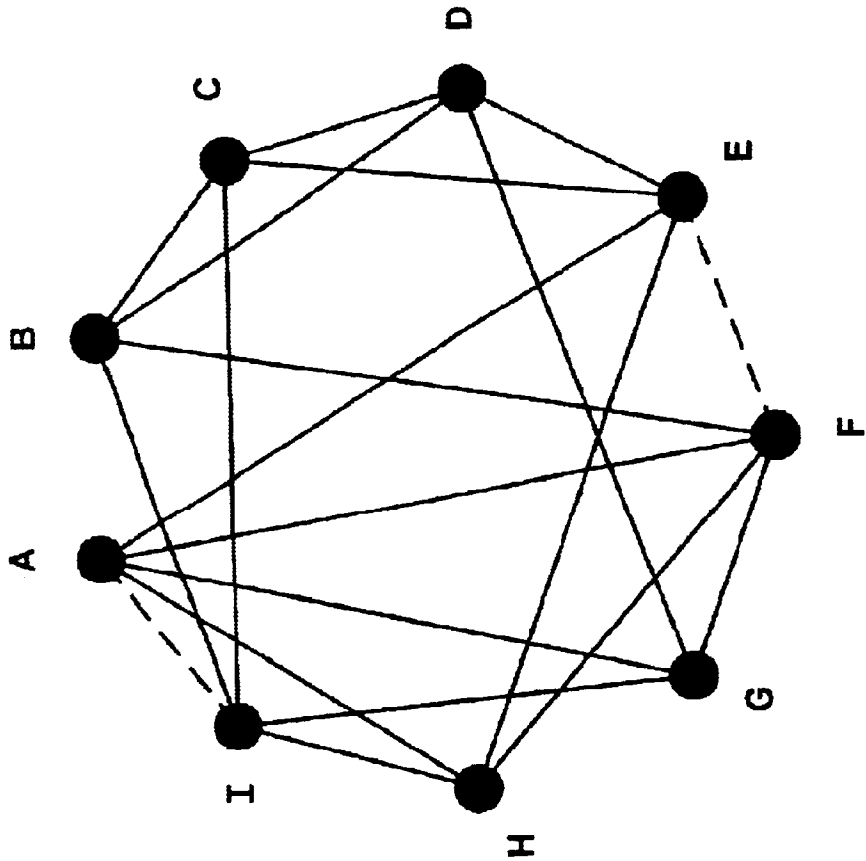


Fig. 5A

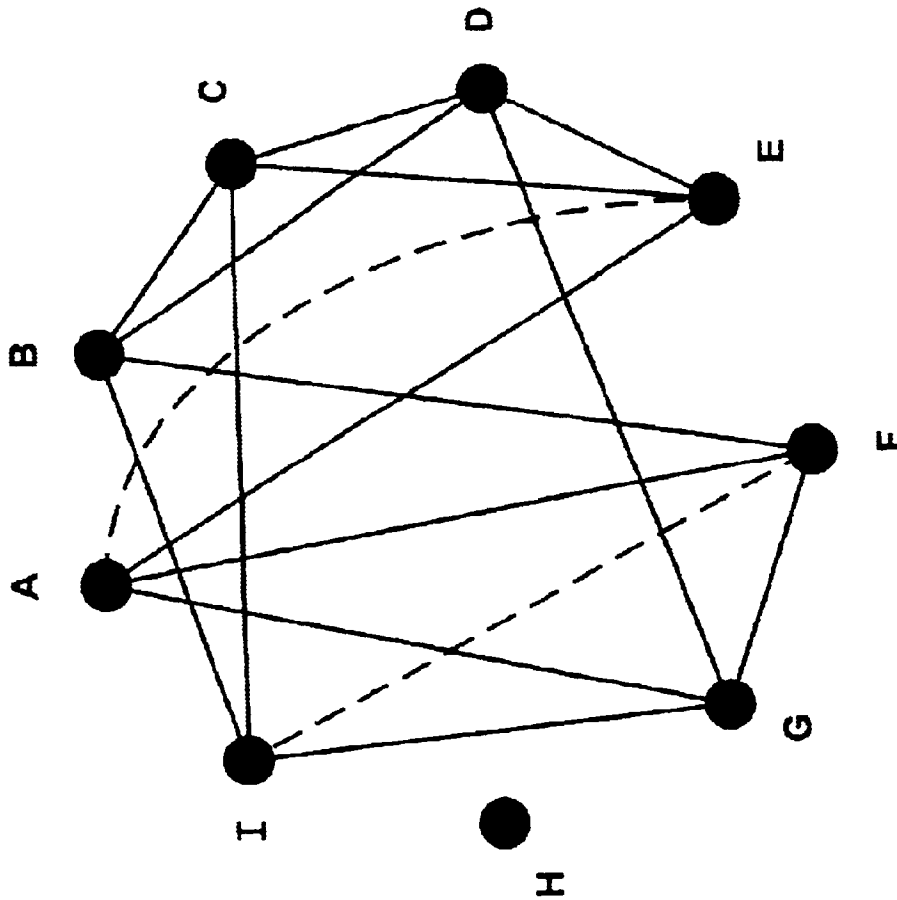


Fig. 5B

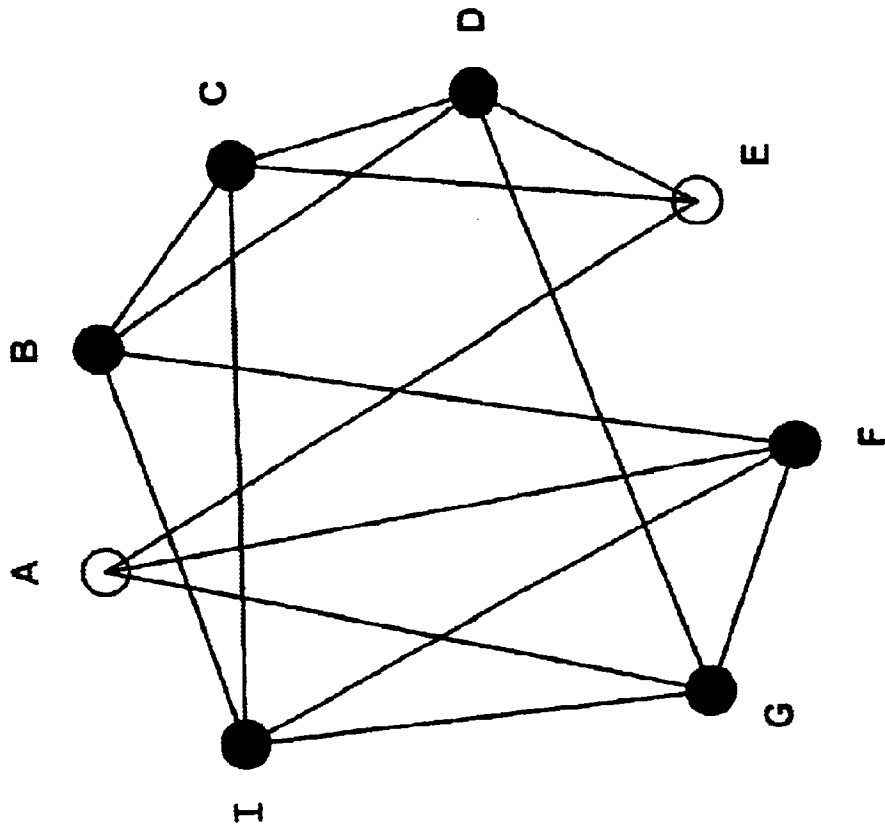


Fig. 5C

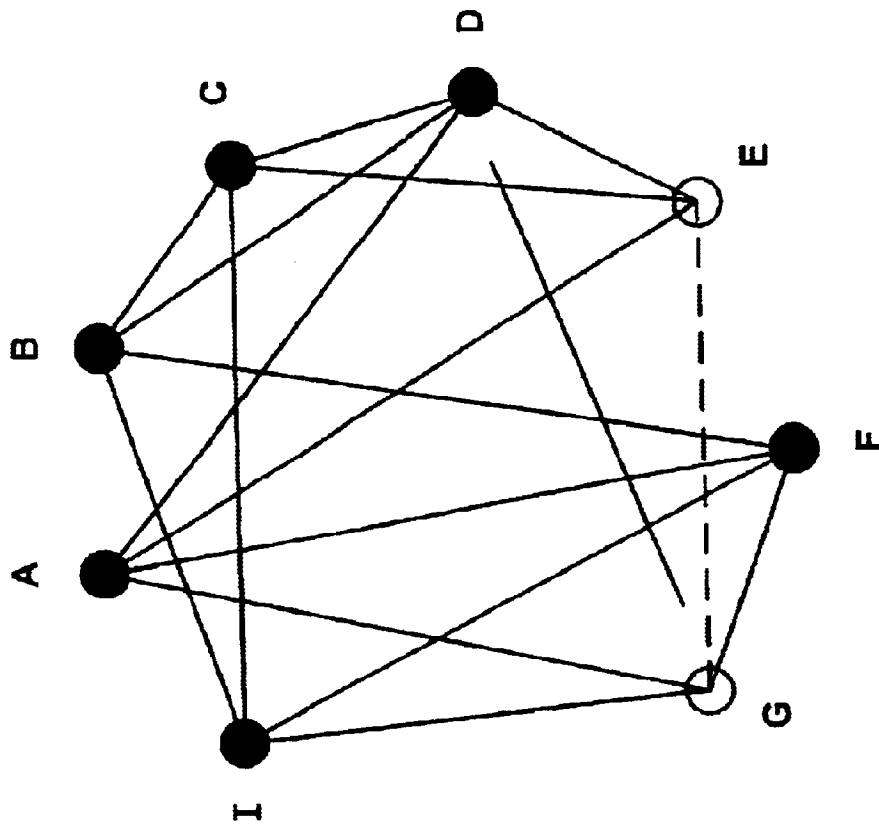


Fig. 5D

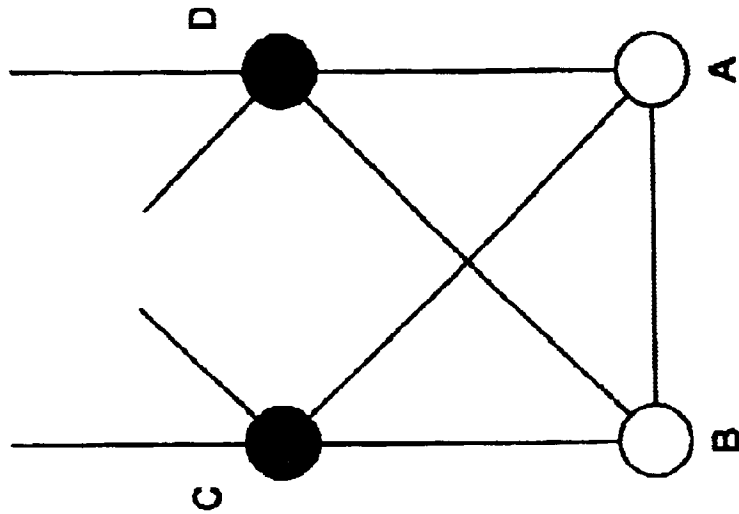


Fig. 5F

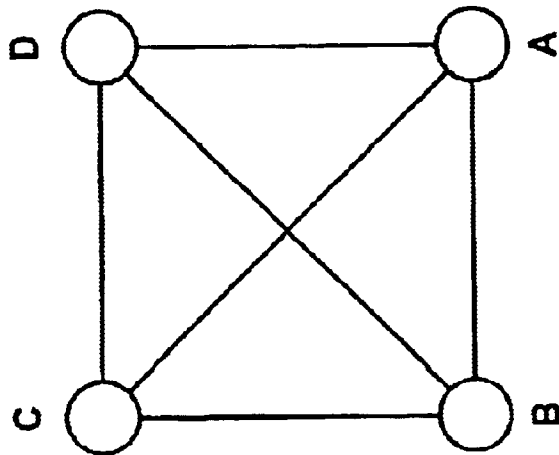


Fig. 5E

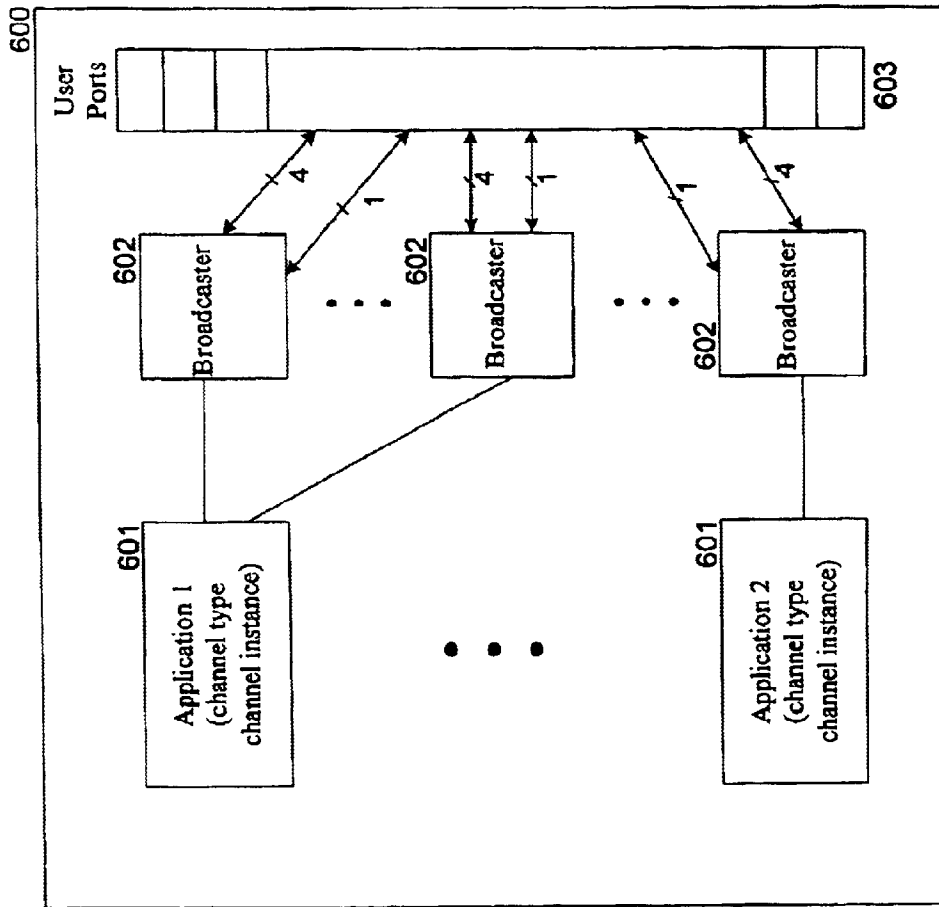


Fig. 6

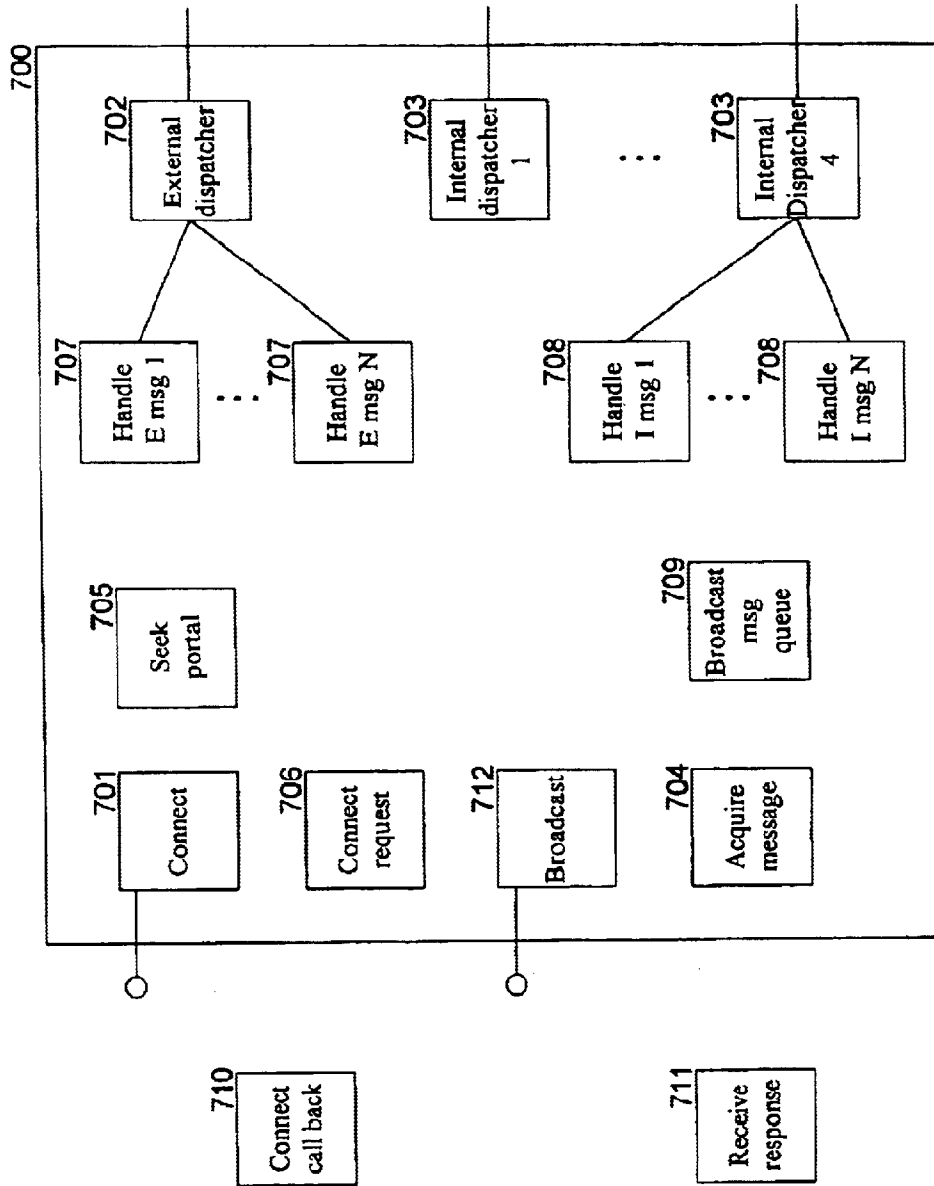
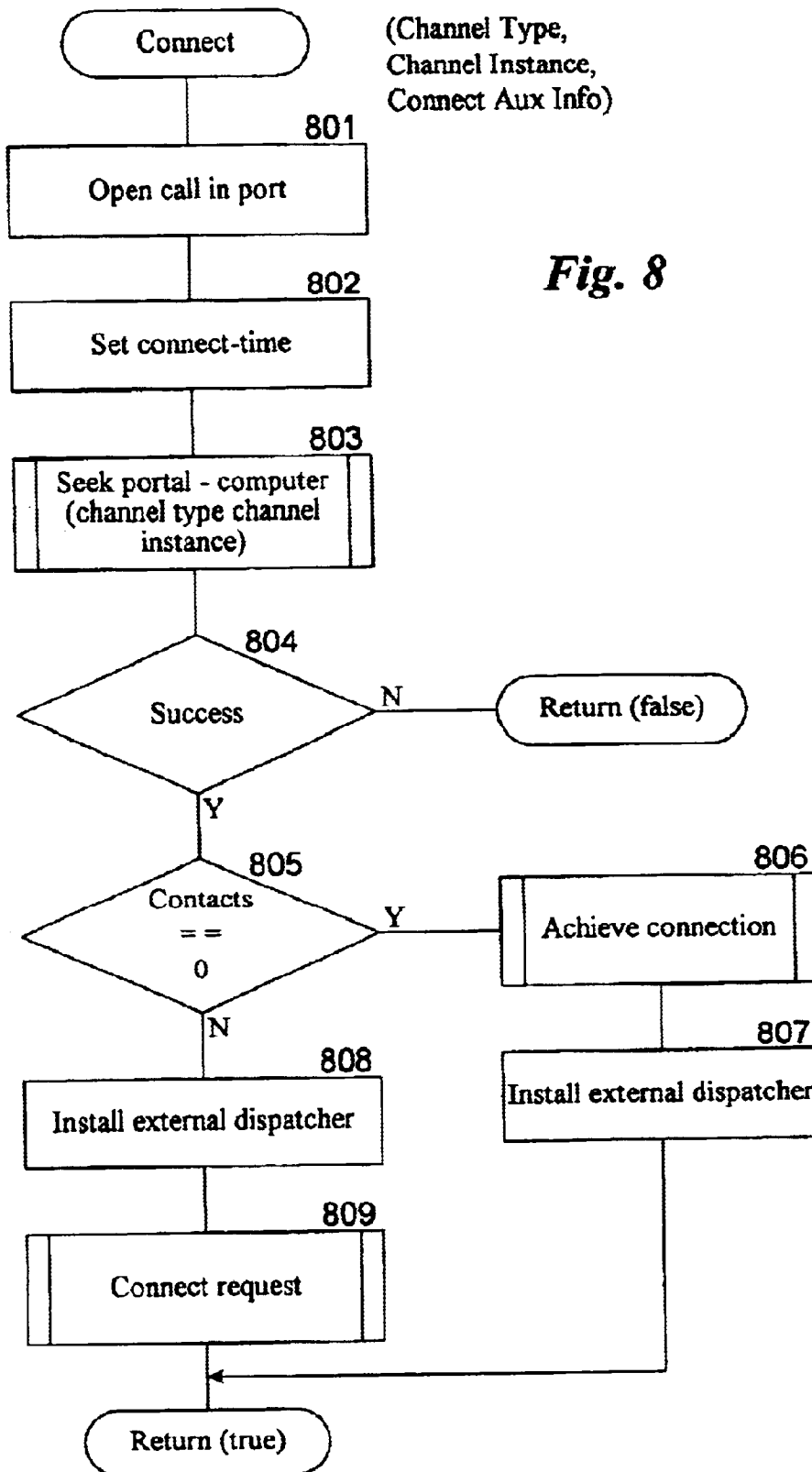
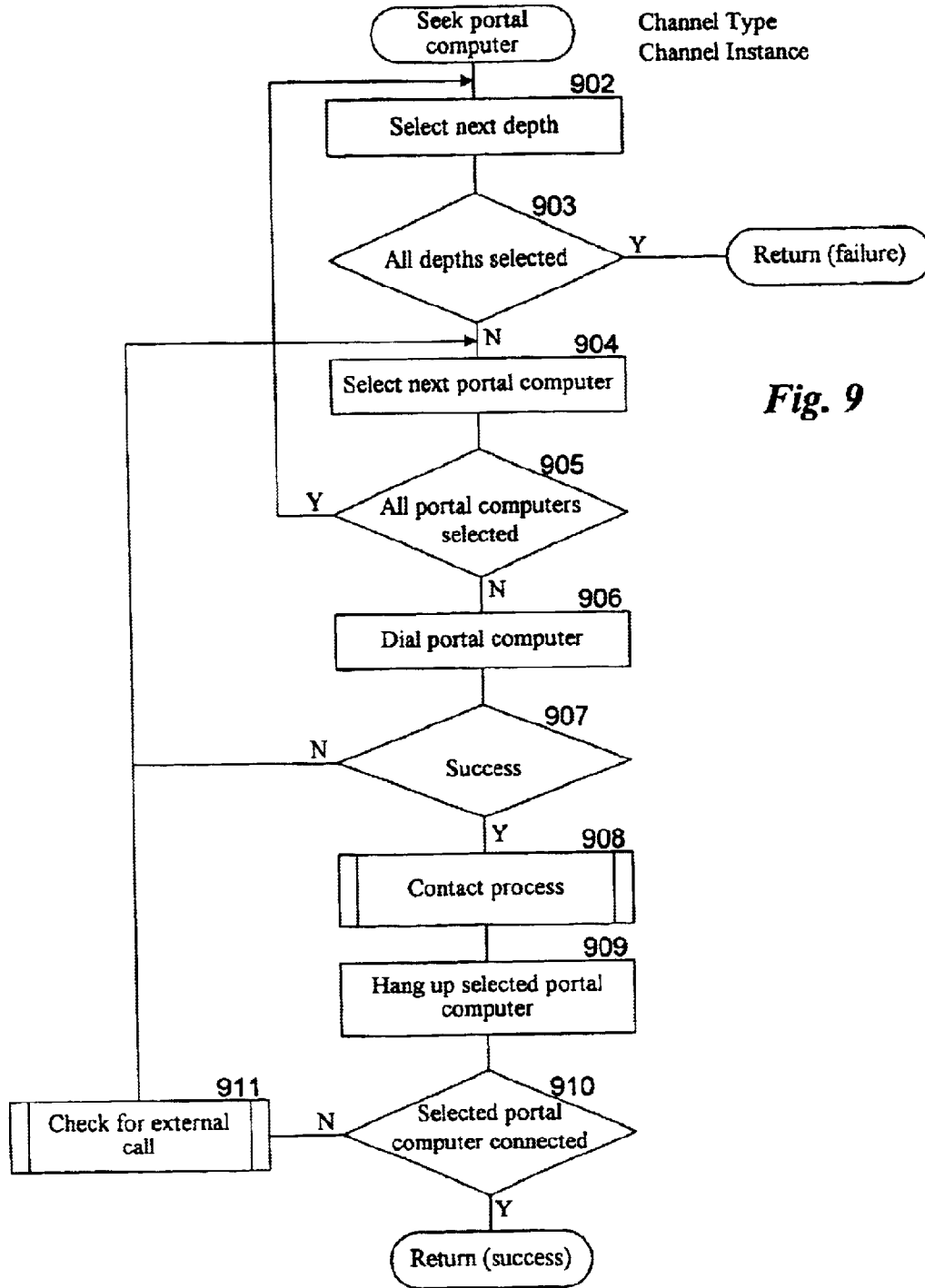


Fig. 7





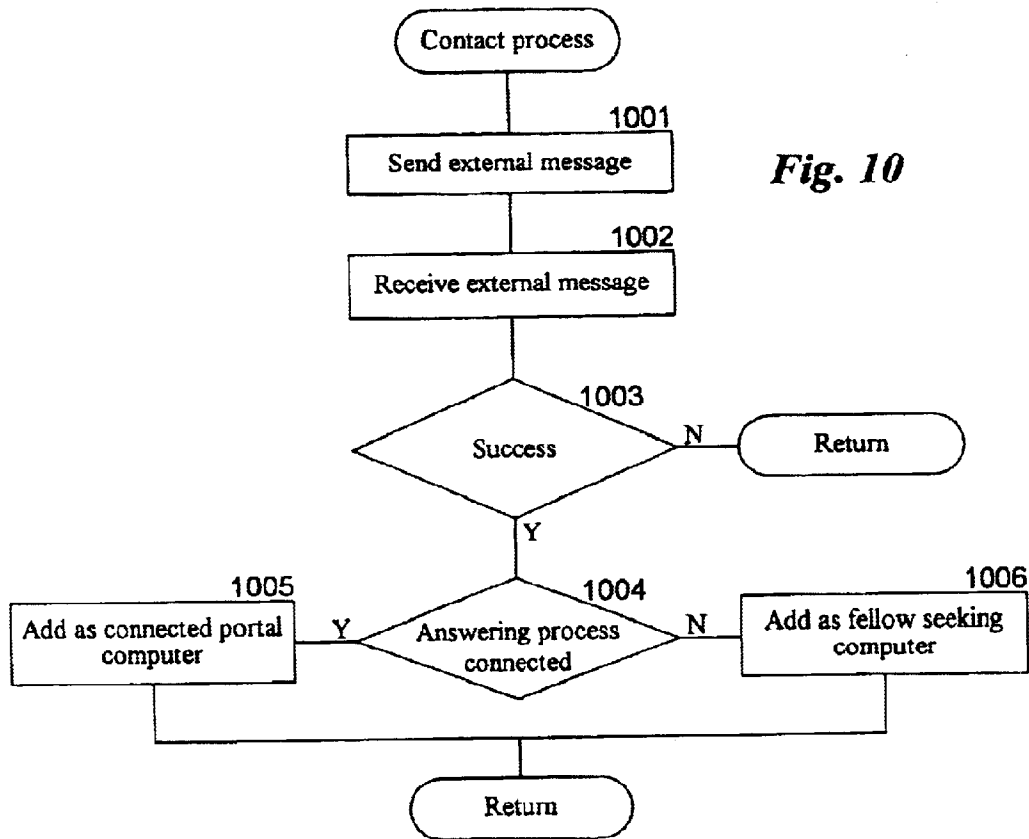


Fig. 11

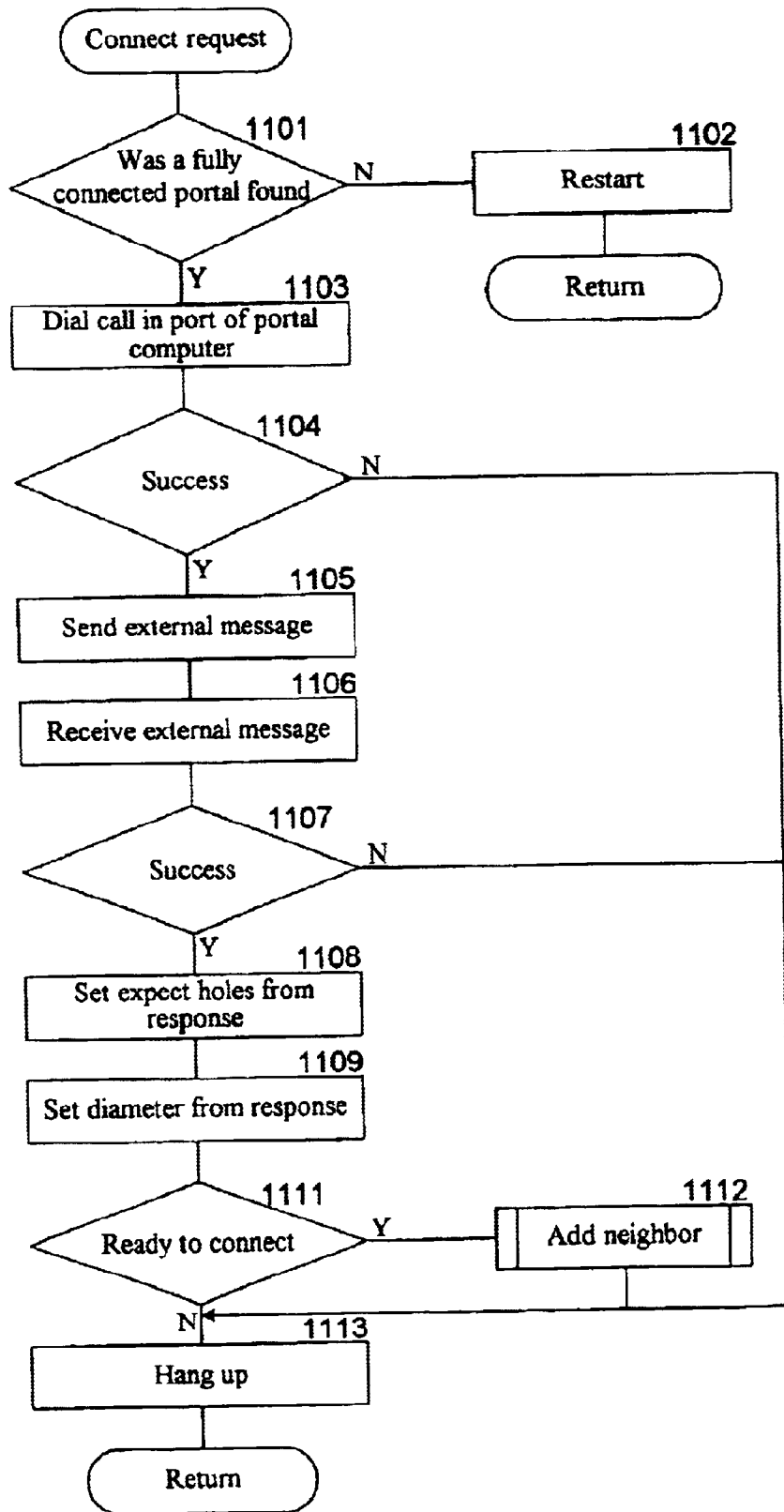
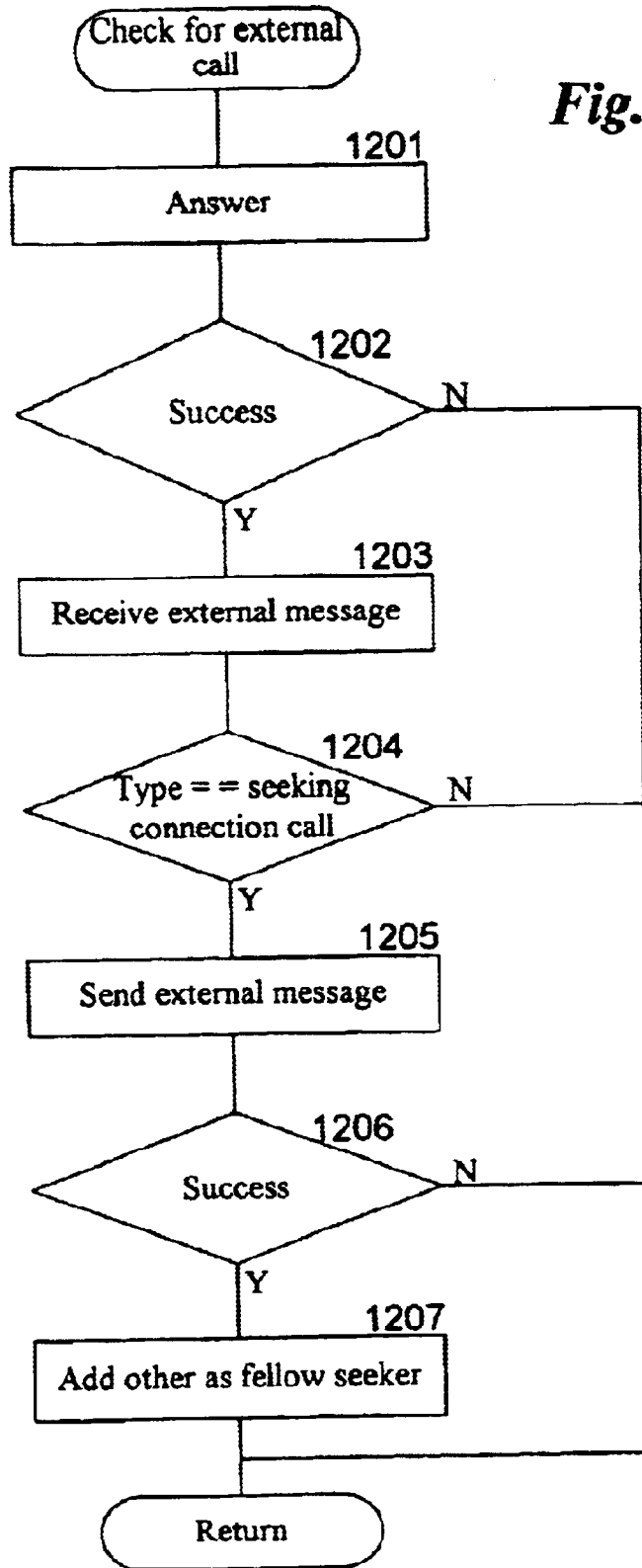


Fig. 12



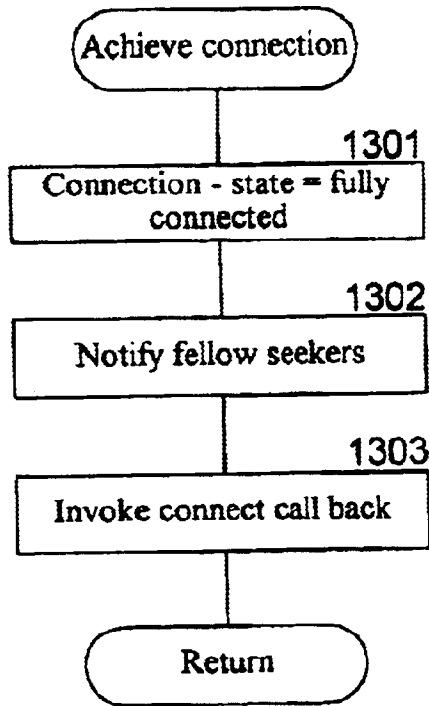
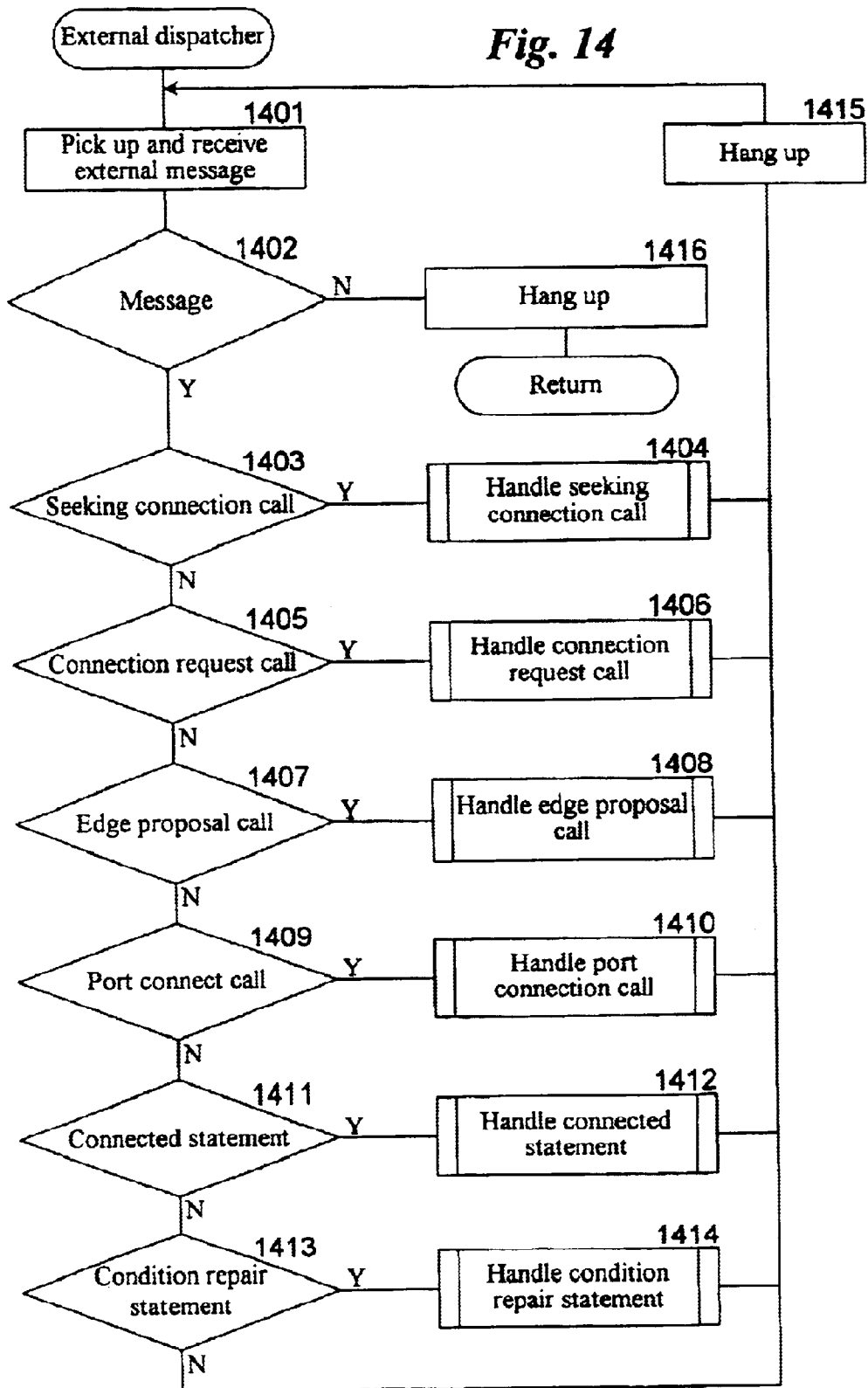
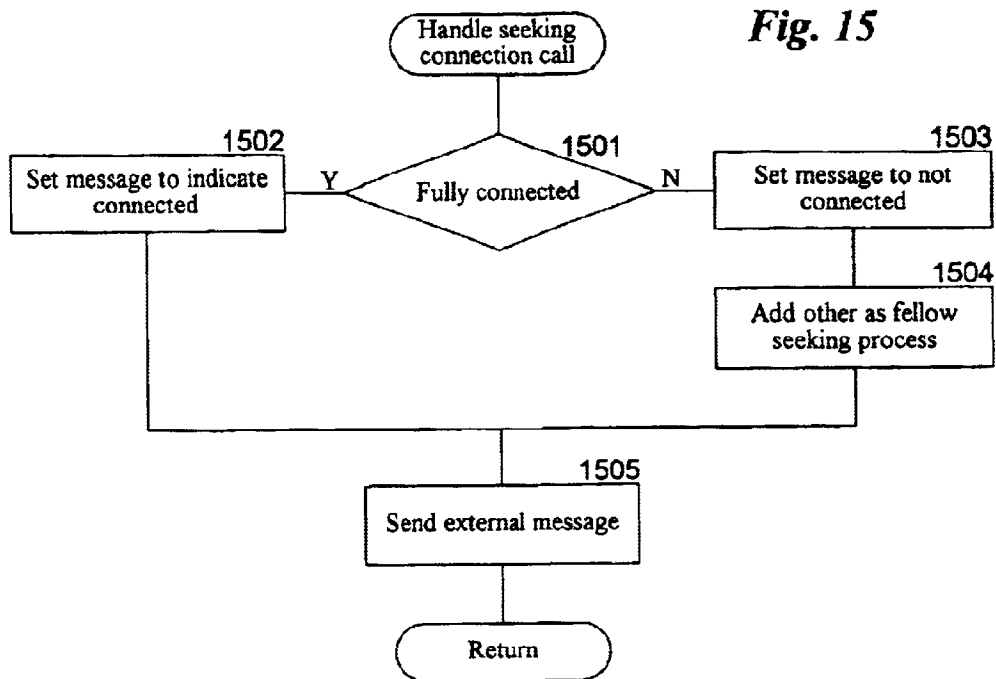


Fig. 13





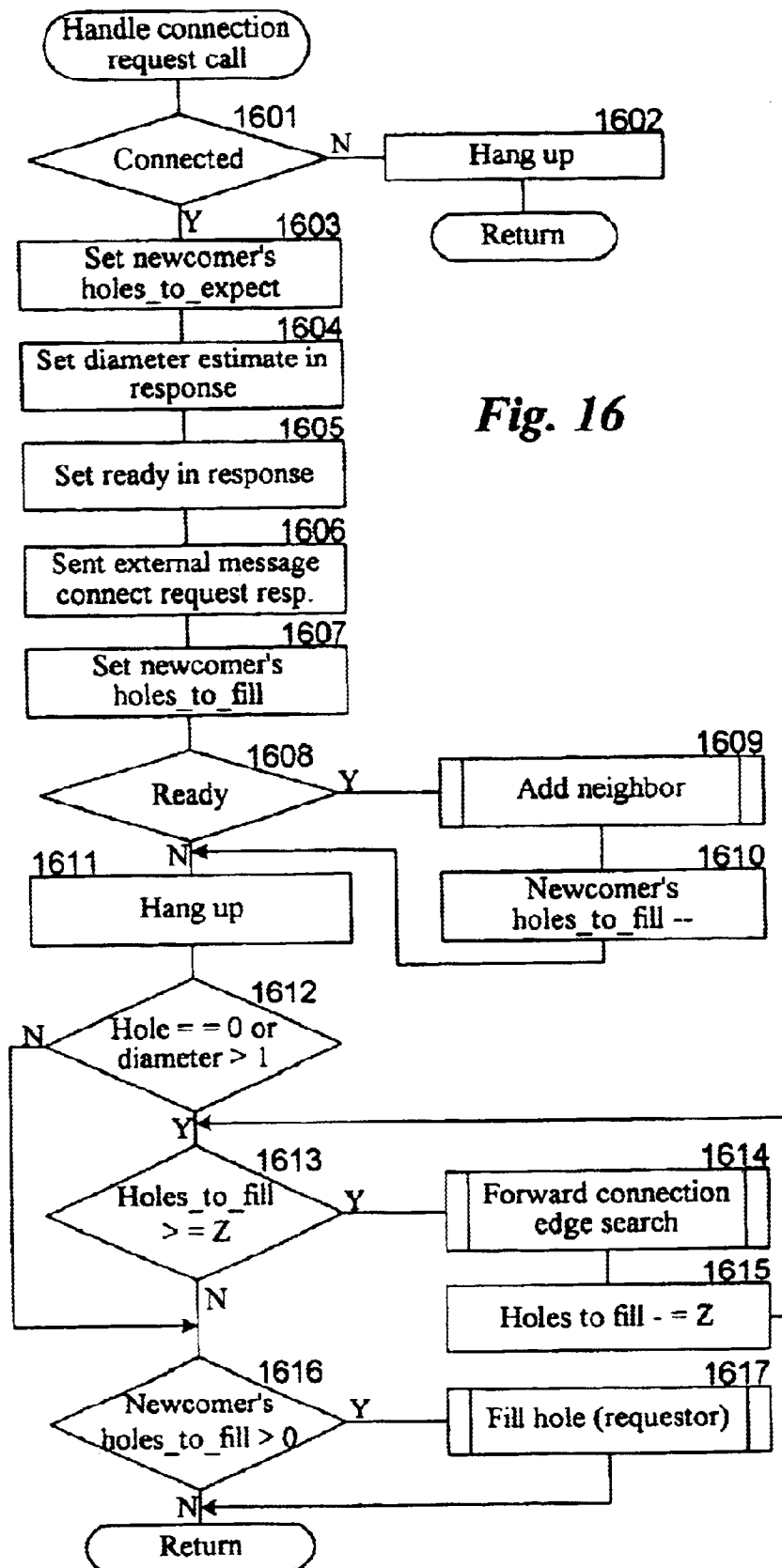


Fig. 16

Fig. 17

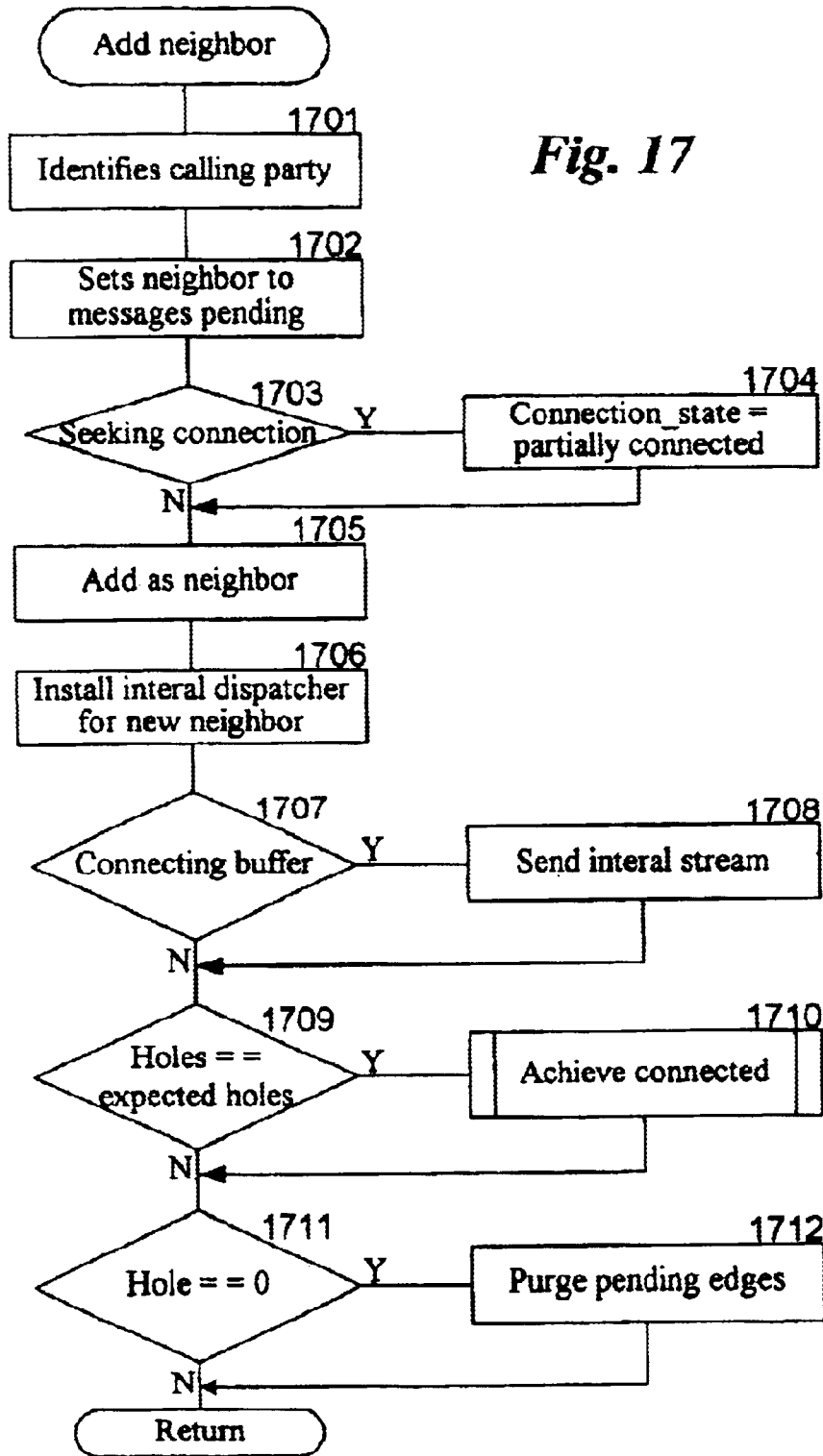
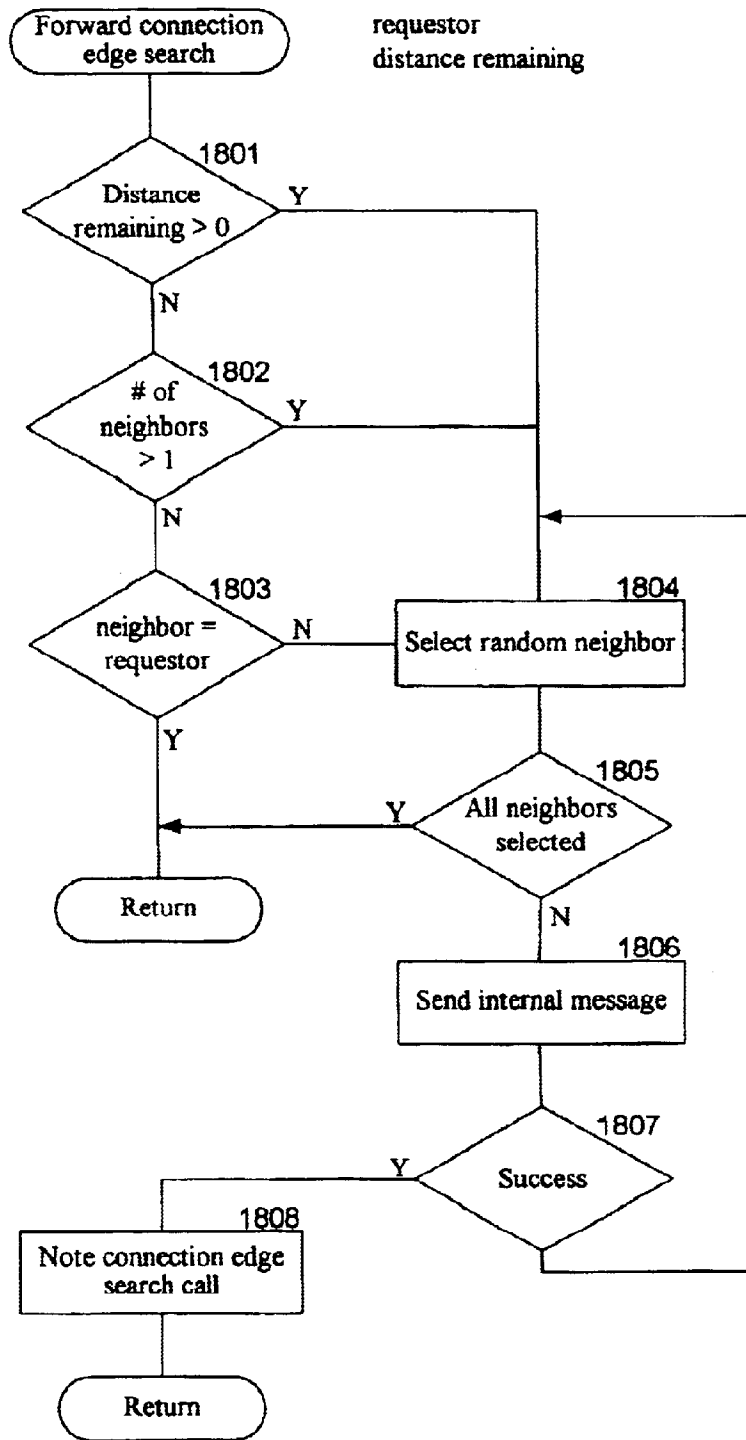
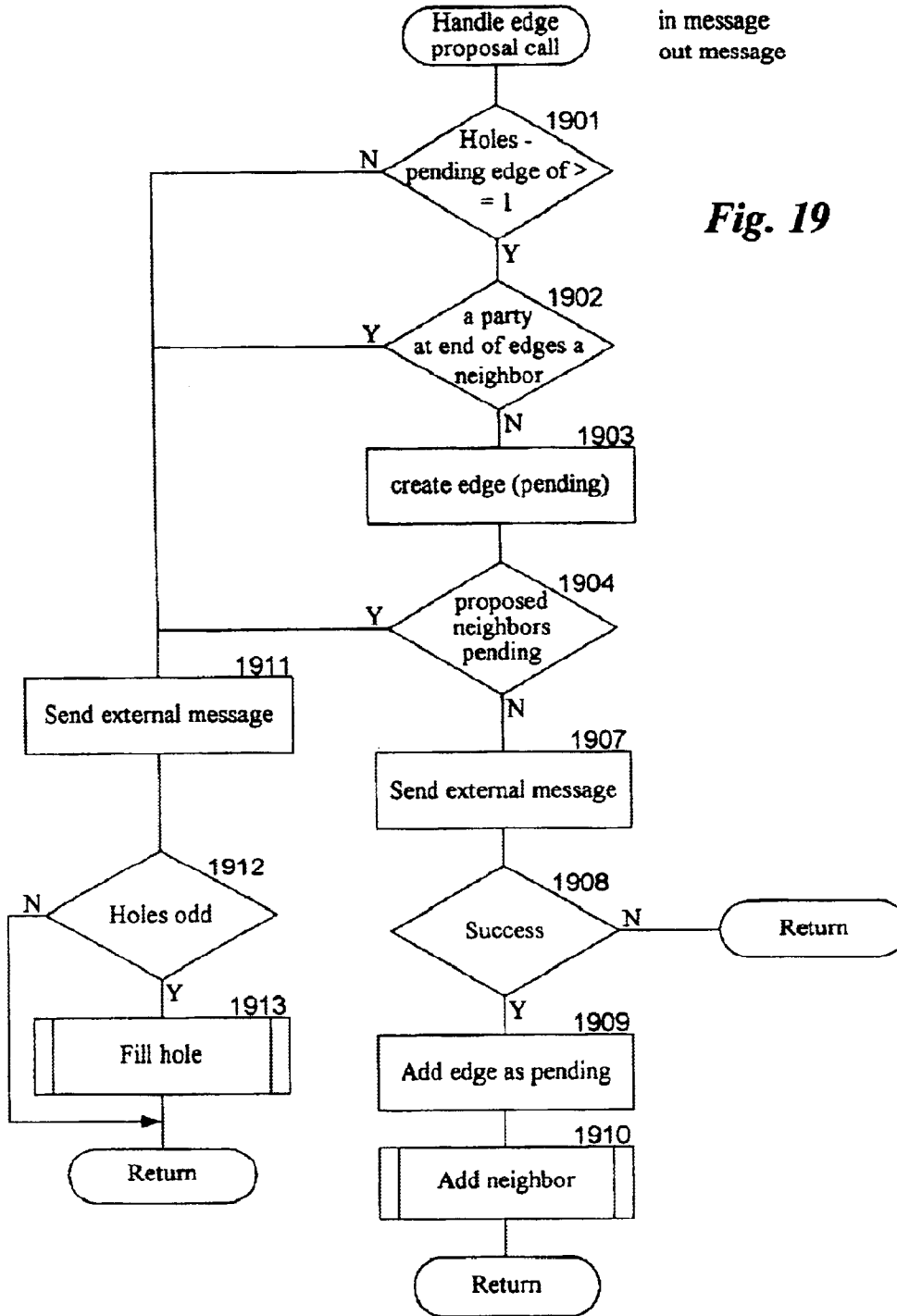


Fig. 18





in message
out message

Fig. 19

Fig. 20

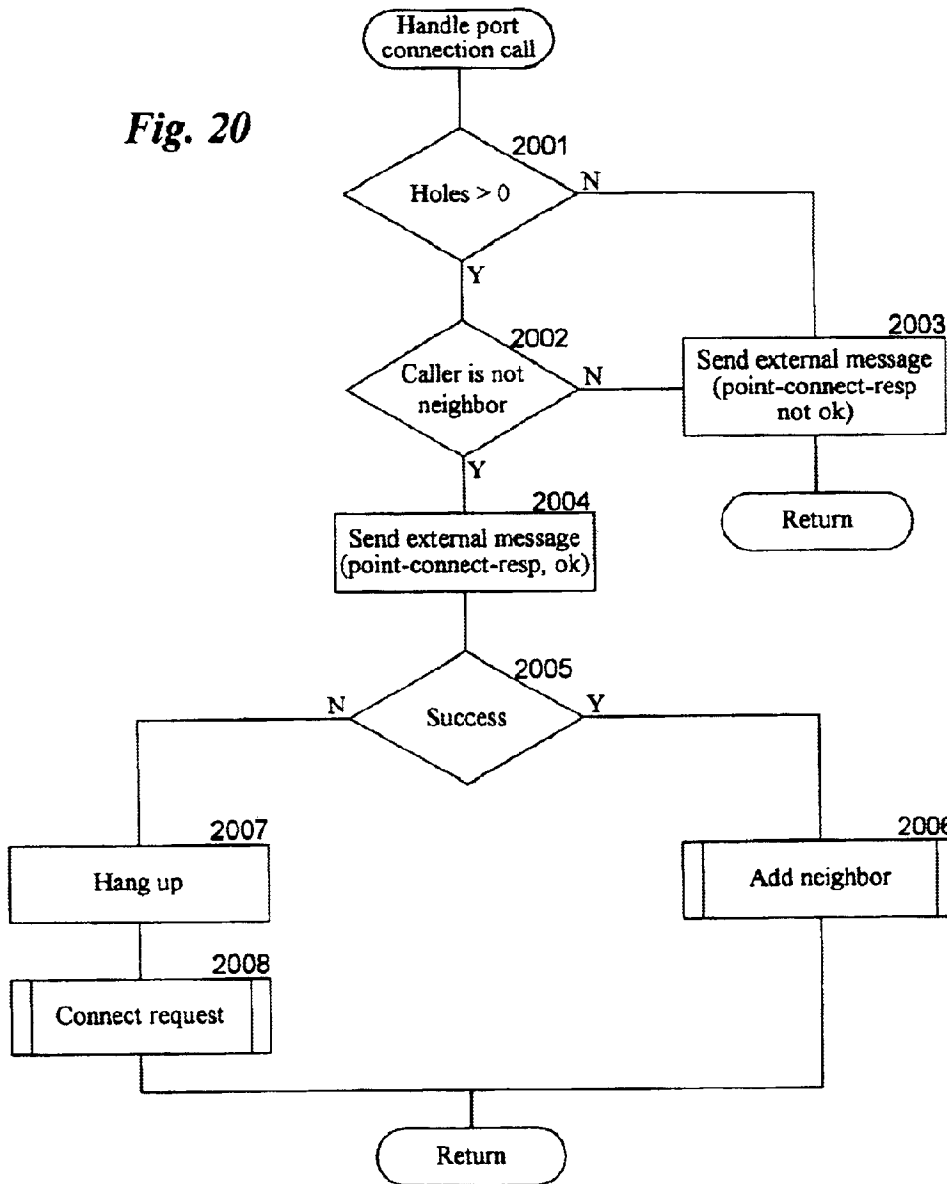


Fig. 21

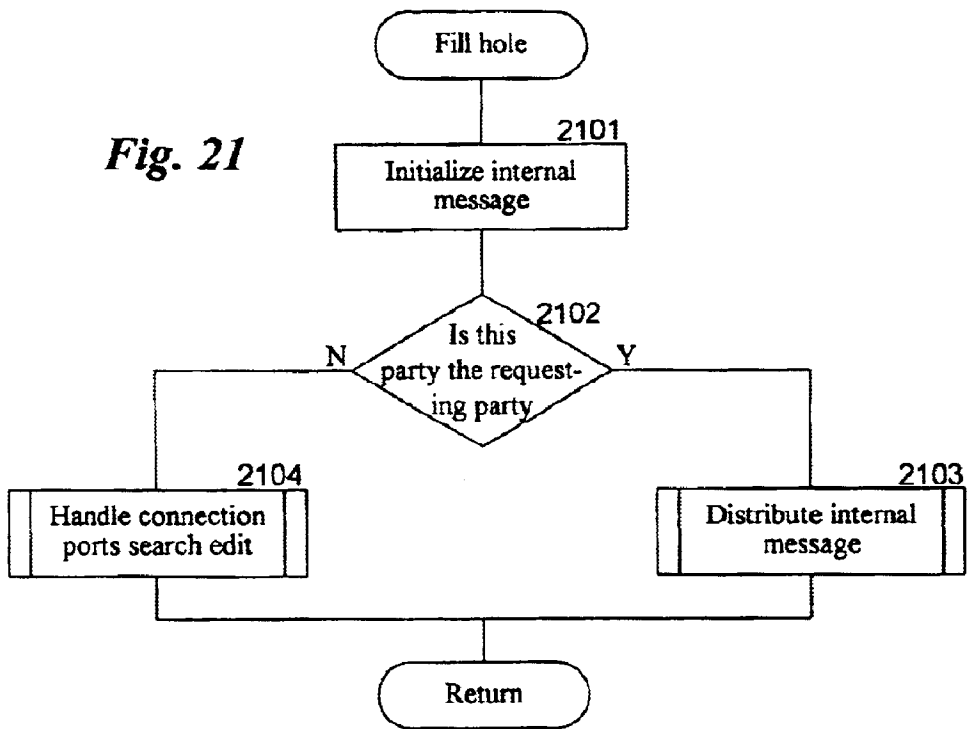


Fig. 22

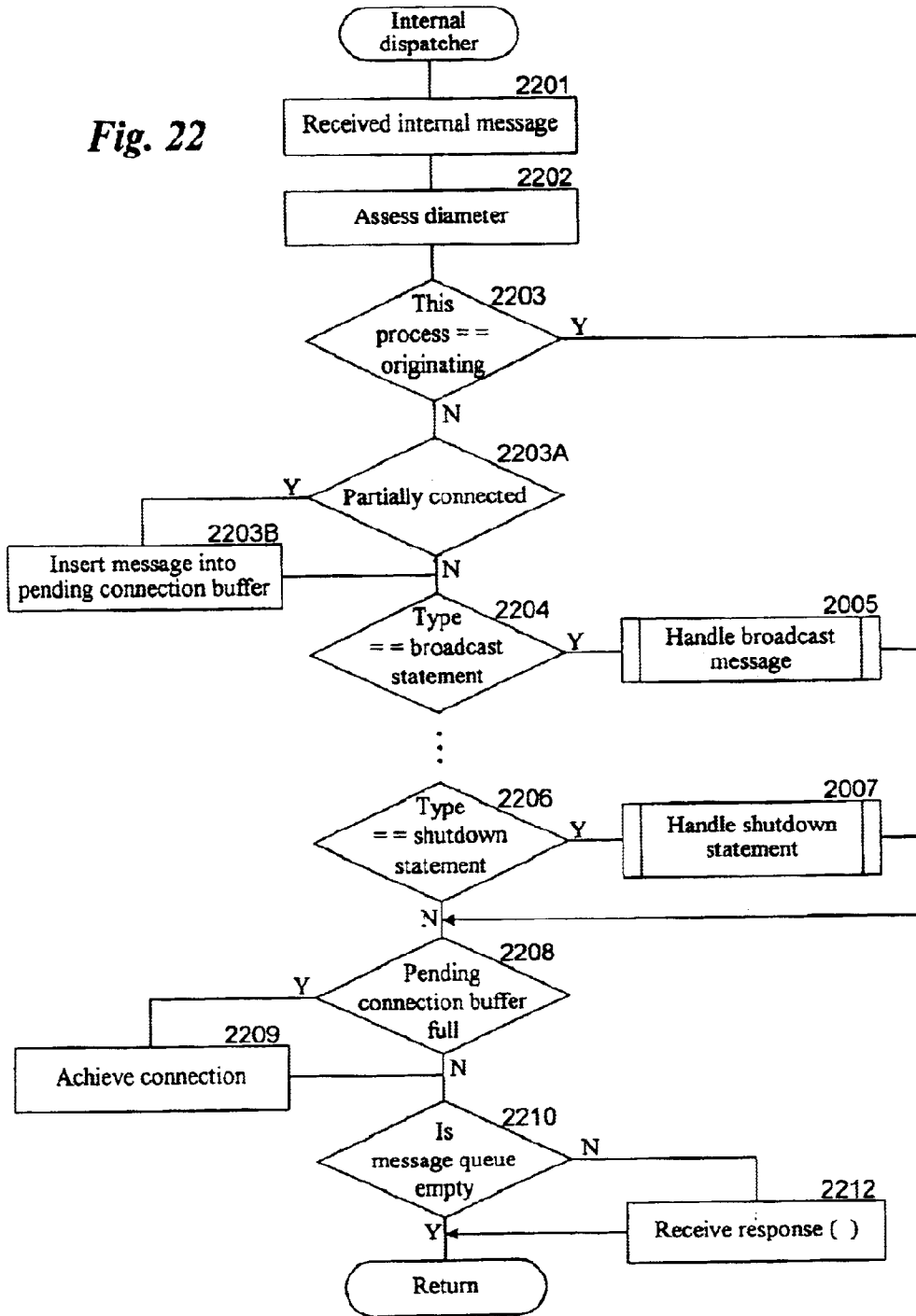


Fig. 23

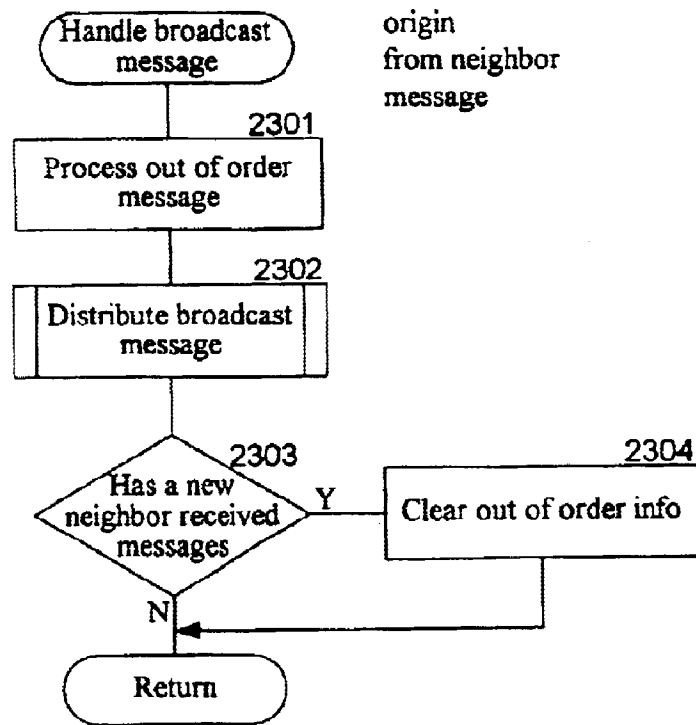
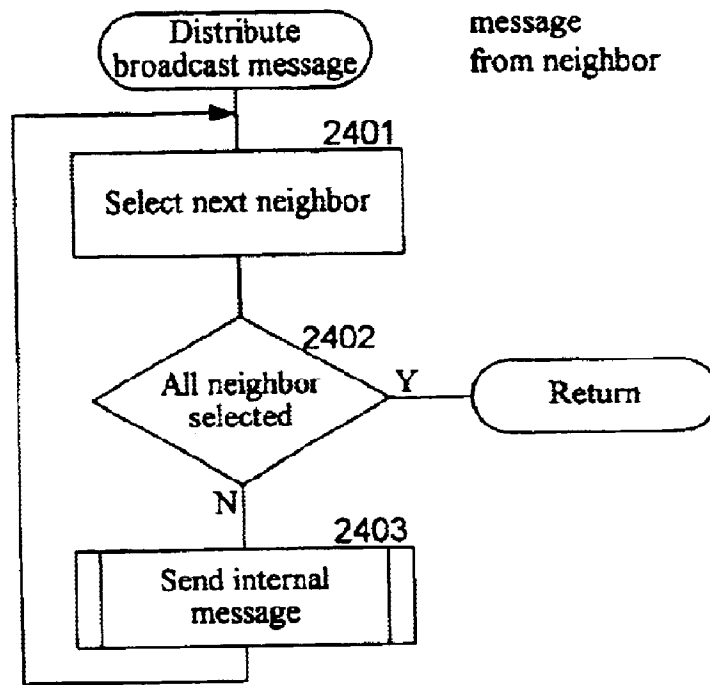


Fig. 24



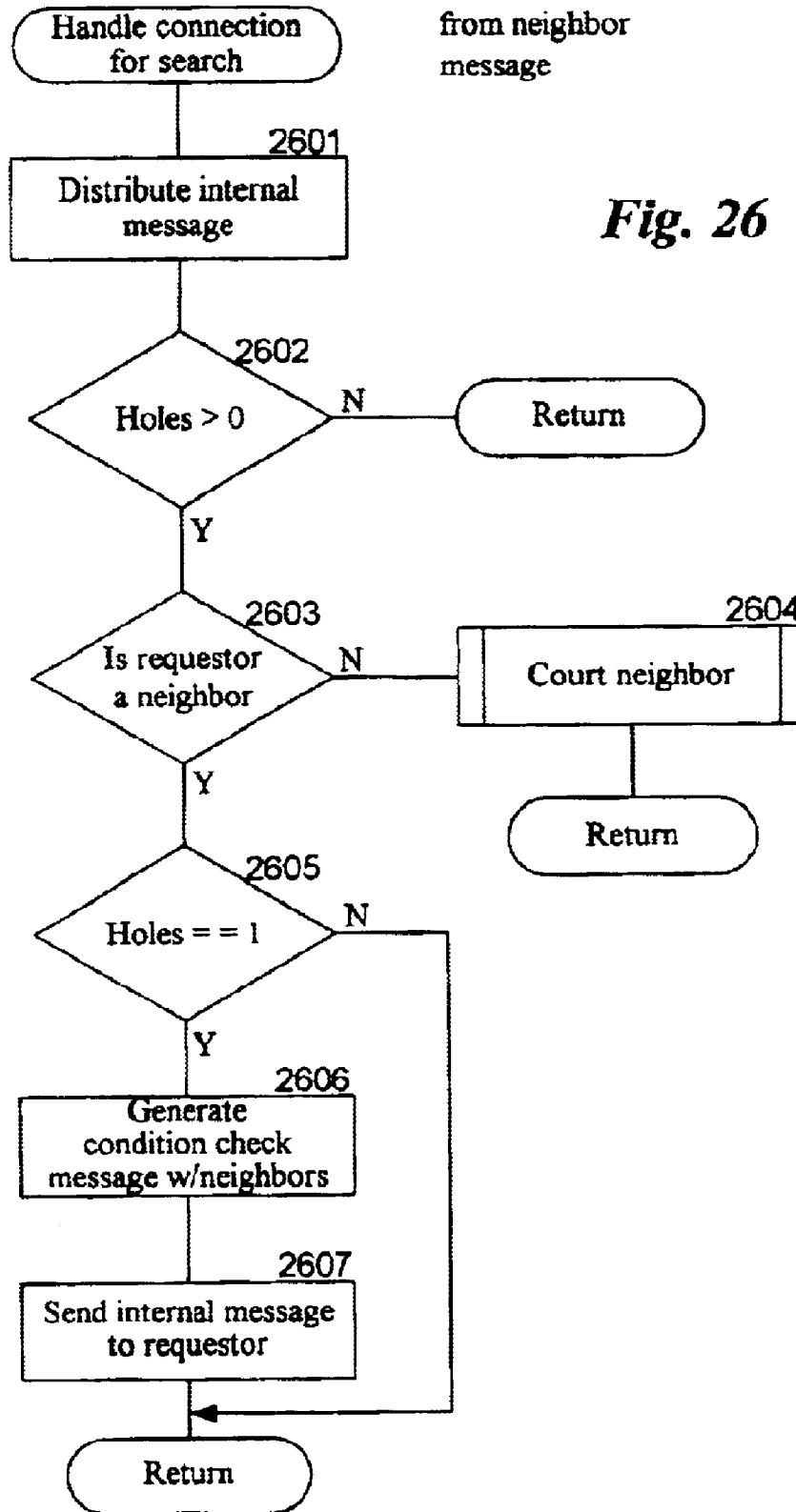


Fig. 27

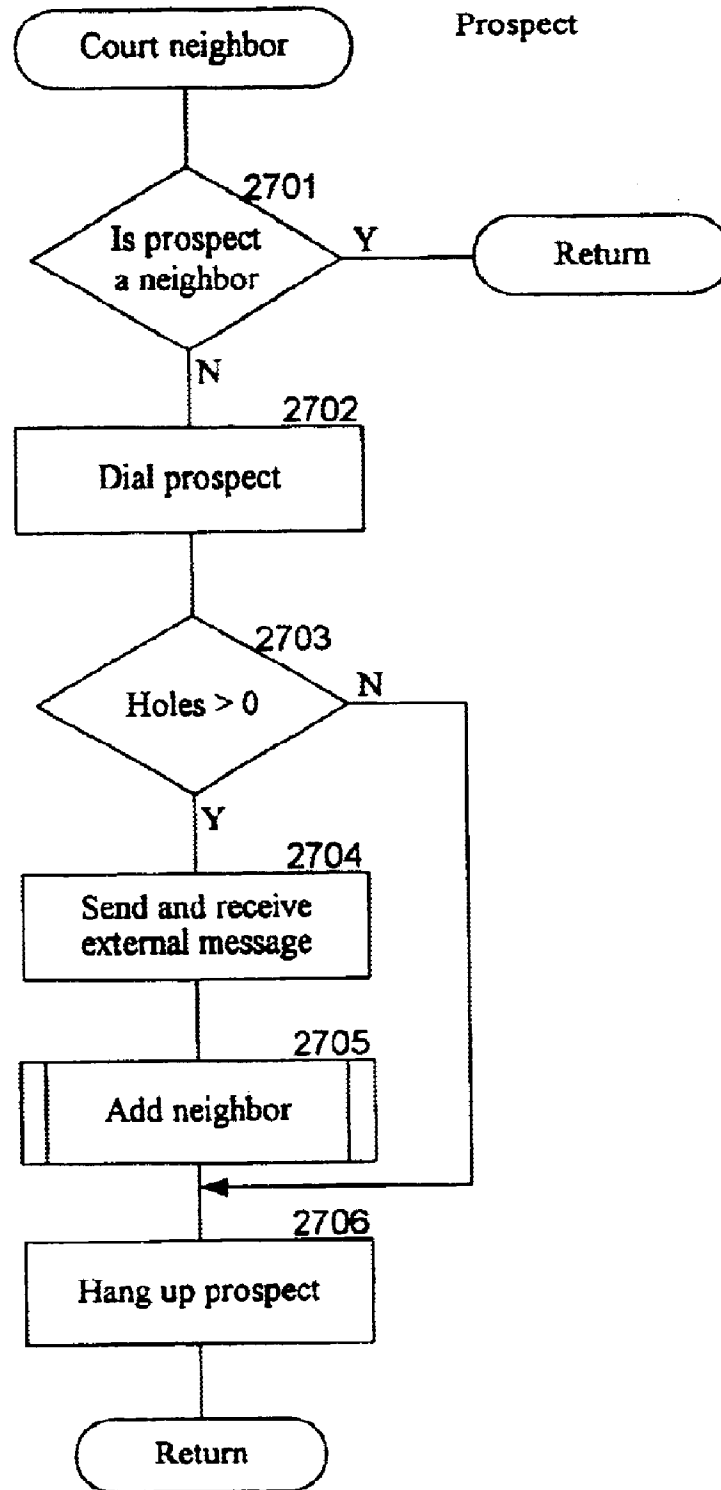


Fig. 28

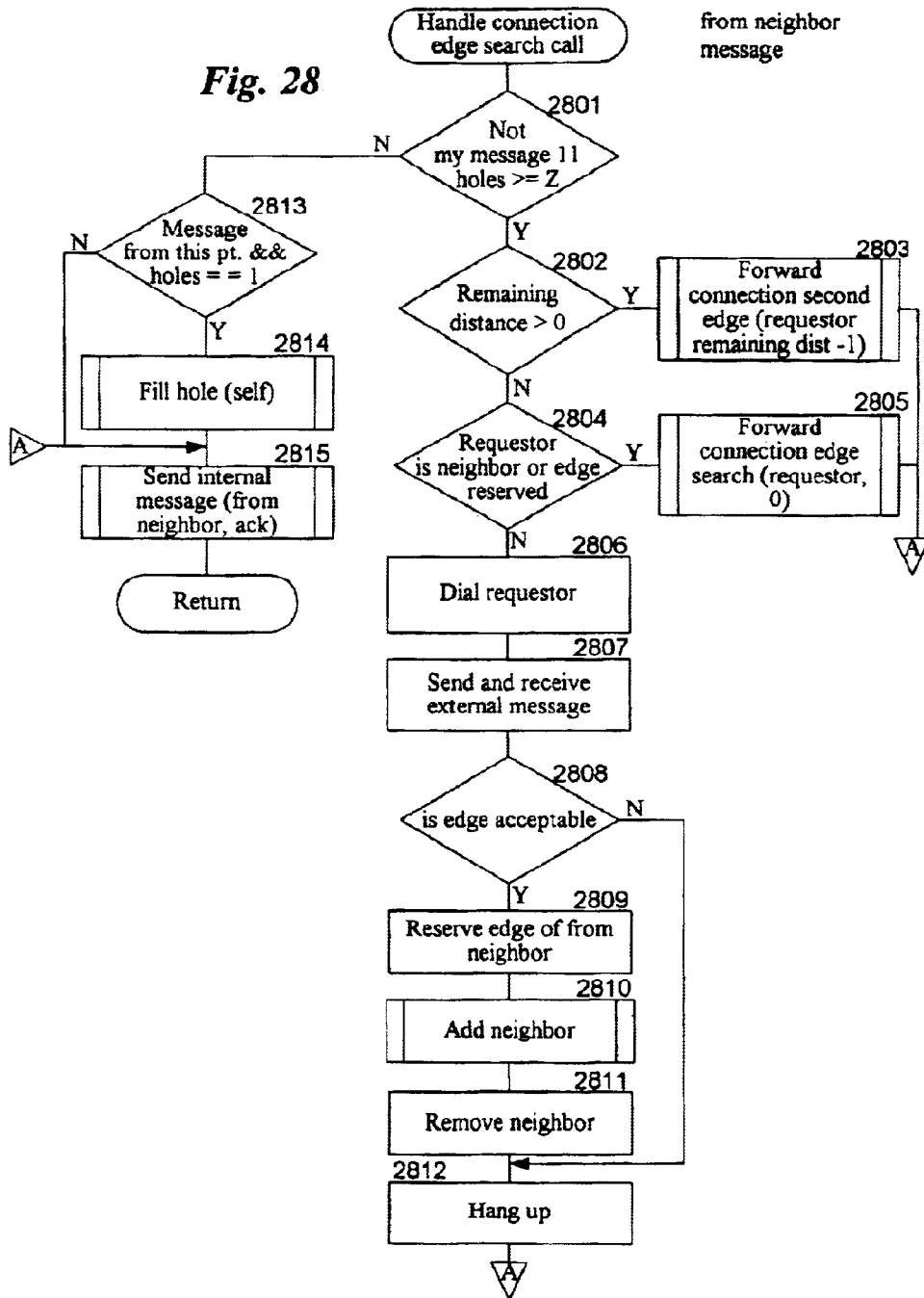


Fig. 29

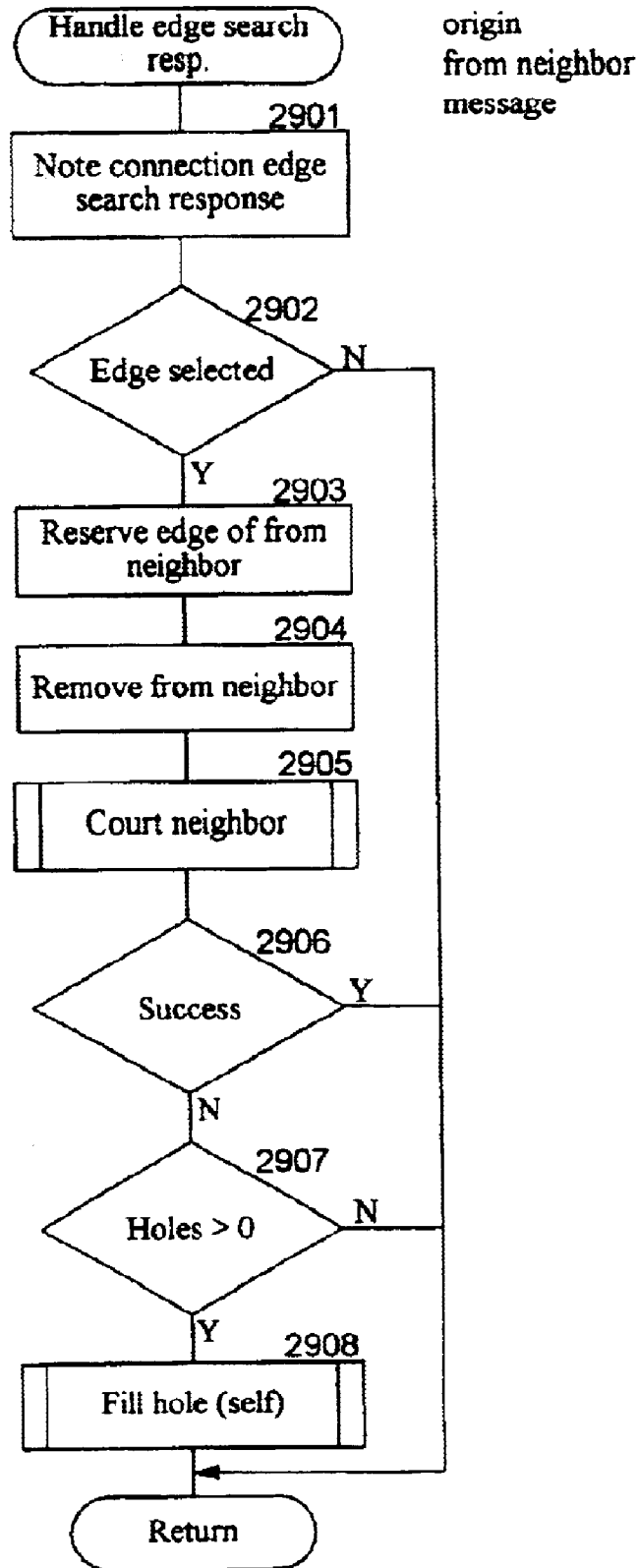
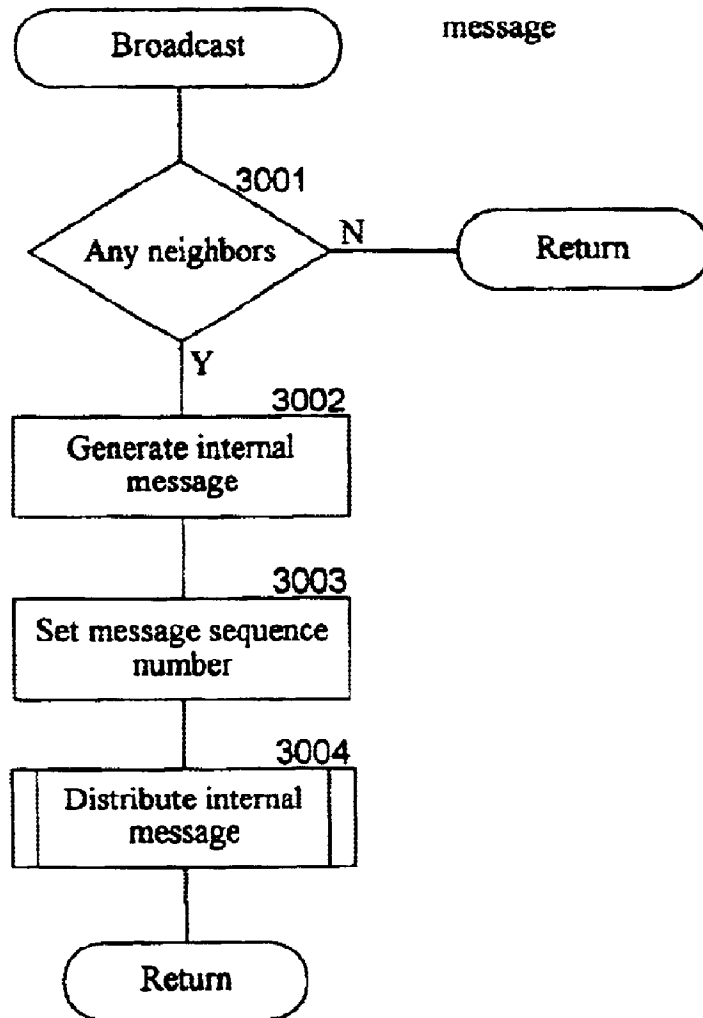


Fig. 30



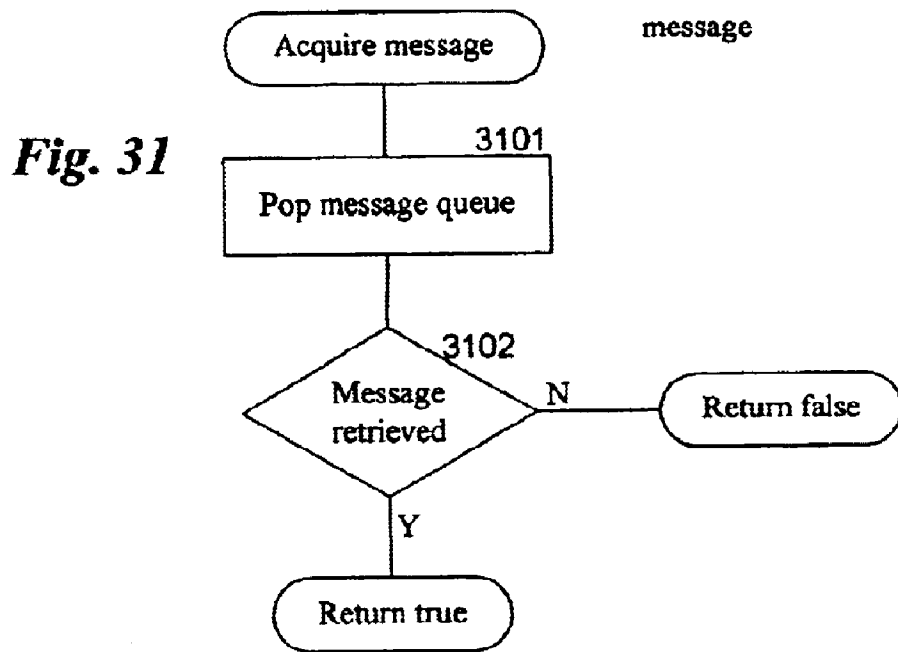


Fig. 32

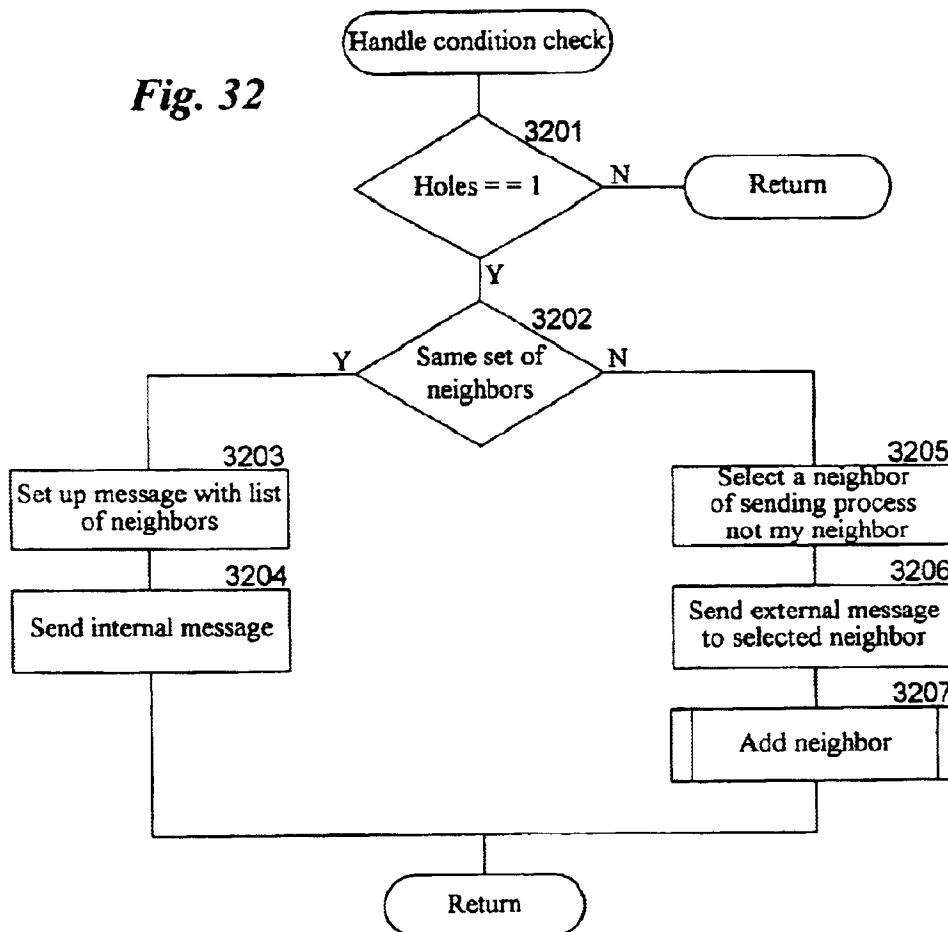
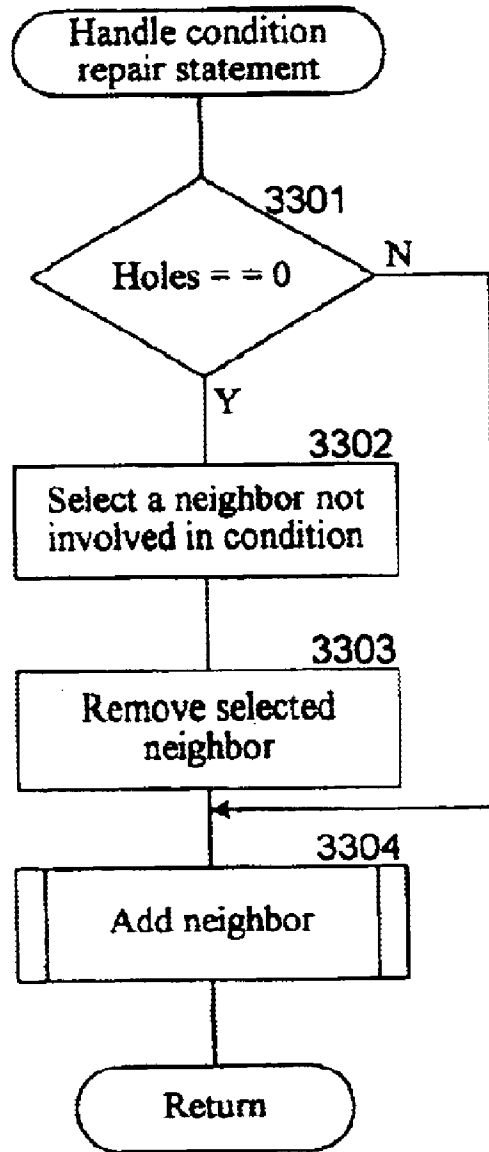
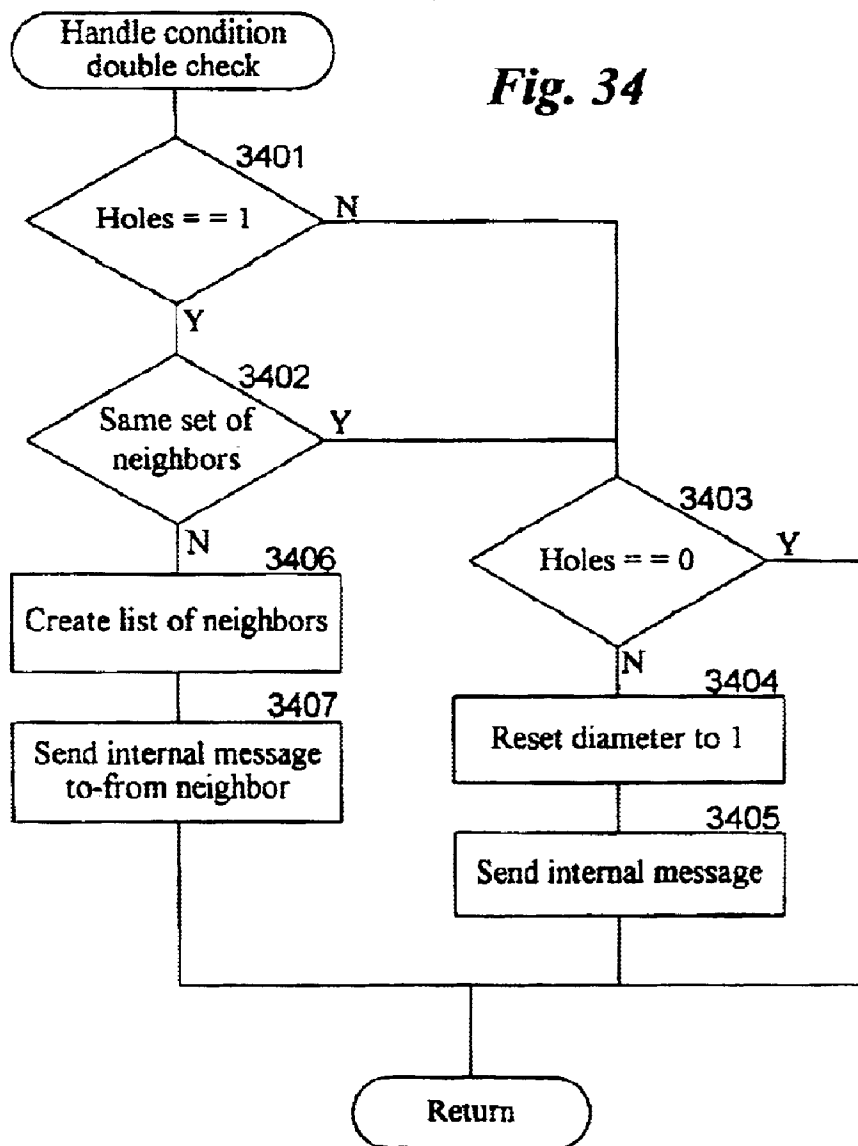


Fig. 33





JOINING A BROADCAST CHANNEL

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is related to U.S. patent application Ser. No. 09/629,576, entitled "BROADCASTING NETWORK," filed on Jul. 31, 2000; U.S. patent application Ser. No. 09/629,570, entitled "JOINING A BROADCAST CHANNEL," filed on Jul. 31, 2000; U.S. patent application Ser. No. 09/629,577, "LEAVING A BROADCAST CHANNEL," filed on Jul. 31, 2000; U.S. patent application Ser. No. 09/629,575, entitled "BROADCASTING ON A BROADCAST CHANNEL," filed on Jul. 31, 2000; U.S. patent application Ser. No. 09/629,572, entitled "CONTACTING A BROADCAST CHANNEL," filed on Jul. 31, 2000; U.S. patent application Ser. No. 09/629,023, entitled "DISTRIBUTED AUCTION SYSTEM," filed on Jul. 31, 2000; U.S. patent application Ser. No. 09/629,043, entitled "AN INFORMATION DELIVERY SERVICE," filed on Jul. 31, 2000, now U.S. Pat. No. 6,714,966; U.S. patent application Ser. No. 09/629,024, entitled "DISTRIBUTED CONFERENCING SYSTEM," filed on Jul. 31, 2000; and U.S. patent application Ser. No. 09/629,042, entitled "DISTRIBUTED GAME ENVIRONMENT," filed on Jul. 31, 2000, the disclosures of which are incorporated herein by reference.

TECHNICAL FIELD

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

BACKGROUND

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

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

The client/server middleware systems provide a server that coordinates the communications between the various clients who are sharing the information. The server functions as a central authority for controlling access to shared resources. Examples of client/server middleware systems include remote procedure calls ("RPC"), database servers, and the common object request broker architecture

("CORBA"). Client/server middleware systems are not particularly well suited to sharing of information among many participants. In particular, when a client stores information to be shared at the server, each other client would need to poll the server to determine that new information is being shared. Such polling places a very high overhead on the communications network. Alternatively, each client may register a callback with the server, which the server then invokes when new information is available to be shared. Such a callback technique presents a performance bottleneck because a single server needs to callback to each client whenever new information is to be shared. In addition, the reliability of the entire sharing of information depends upon the reliability of the single server. Thus, a failure at a single computer (i.e., the server) would prevent communications between any of the clients.

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

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

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

BRIEF DESCRIPTION OF THE DRAWINGS

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

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

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

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

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

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

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

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

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

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

US 6,910,069 B1

3

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4

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

DETAILED DESCRIPTION

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

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

FIG. 1 illustrates a graph that is 4-regular and 4-connected which represents the broadcast channel. Each of the nine nodes A-I represents a computer that is connected to the broadcast channel, and each of the edges represents an "edge" connection between two computers of the broadcast channel. The time it takes to broadcast a message to each computer on the broadcast channel depends on the speed of the connections between the computers and the number of connections between the originating computer and each other computer on the broadcast channel. The minimum number of connections that a message would need to traverse between each pair of computers is the "distance" between the computers (i.e., the shortest path between the two nodes of the graph). For example, the distance between computers A and F is one because computer A is directly connected to computer F. The distance between computers A and B is two because there is no direct connection between computers A and B, but computer F is directly connected to

US 6,910,069 B1

5

computer B. Thus, a message originating at computer A would be sent directly to computer F, and then sent from computer F to computer B. The maximum of the distances between the computers is the “diameter” of broadcast channel. The diameter of the broadcast channel represented by FIG. 1 is two. That is, a message sent by any computer would traverse no more than two connections to reach every other computer. FIG. 2 illustrates a graph representing 20 computers connected to a broadcast channel. The diameter of this broadcast channel is 4. In particular, the shortest path between computers 1 and 3 contains four connections (1-12, 12-15, 15-18, and 18-3).

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

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

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

6

D are the two pairs that are identified as the neighbors for the new computer Z. The connections between each of these pairs is broken, and a connection between computer Z and each of computers B, C, D, and E is established as indicated by FIG. 3B. The process of breaking the connection between two neighbors and reconnecting each of the former neighbors to another computer is referred to as “edge pinning” as the edge between two nodes may be considered to be stretched and pinned to a new node.

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

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

A seeking computer could connect to the broadcast channel by connecting to computers either directly connected to the found portal computer or directly connected to one of its neighbors. A possible problem with such a scheme for identifying the neighbors for the seeking computer is that the

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

Broadcasting Through the Graph

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

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

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

receiving computer. The second message only has to travel a distance of one. Thus, it is possible for the second message to reach the receiving computer before the first message.

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

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

Decomposing the Graph

A connected computer disconnects from the broadcast channel either in a planned or unplanned manner. When a

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

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

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

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

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

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

FIGS. 5E and 5F further illustrate the neighbors with empty ports condition. FIG. 5E illustrates the neighbors with

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

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

Port Selection

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

channel type and channel instance, it generates the same port ordering. As described below, it is possible for a computer to be connected to multiple broadcast channels that are uniquely identified by channel type and channel instance. The algorithm may be "seeded" with channel type and channel instance in order to generate a unique ordering of port numbers for each broadcast channel. Thus, a seeking computer will dial the ports of a portal computer in the same order as the portal computer used when allocating its call-in port.

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

Locating a Portal Computer

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

When a seeking computer locates a portal computer that is itself not fully connected, the two computers do not connect when they first locate each other because the

US 6,910,069 B1

13

broadcast channel may already be established and accessible through a higher-ordered port number on another portal computer. If the two seeking computers were to connect to each other, then two disjoint broadcast channels would be formed. Each seeking computer can share its experience in trying to locate a portal computer with the other seeking computer. In particular, if one seeking computer has searched all the portal computers to a depth of eight, then the one seeking computer can share that it has searched to a depth of eight with another seeking computer. If that other seeking computer has searched to a depth of, for example, only four, it can skip searching through depths five through eight and that other seeking computer can advance its searching to a depth of nine.

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

Identifying Neighbors for a Seeking Computer

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

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

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

14

the forwarding computer toggles the new distance to travel between zero and one to help prevent two computers from sending the message back and forth between each other.

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

External Data Representation

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

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

M-Regular

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

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

Components

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

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

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

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

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

The following tables list messages sent by the broadcaster components.

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

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

-continued

INTERNAL MESSAGES

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

Flow Diagrams

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

routine continues at block 808. In block 806, the routine invokes the achieve connection routine to change the state of this process to fully connected. In block 807, the routine installs the external dispatcher for processing messages received through this process' external port for the passed channel type and channel instance. When a message is received through that external port, the external dispatcher is invoked. The routine then returns. In block 808, the routine installs an external dispatcher. In block 809, the routine invokes the connect request routine to initiate the process of identifying neighbors for the seeking computer. The routine then returns.

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

FIG. 10 is a flow diagram illustrating the processing of the contact process routine in one embodiment. This routine determines whether the process of the selected portal computer that answered the call-in to the selected port is fully connected to the broadcast channel. In block 1001, the routine sends an external message (i.e., seeking_connection_call) to the answering process indicating that a seeking process wants to know whether the answering process is fully connected to the broadcast channel. In block 1002, the routine receives the external response message

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

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

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

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

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

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

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

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

filled is greater than or equal to two, then the routine continues at block 1614, else the routine continues at block 1616. In block 1614, the routine invokes the forward connection edge search routine. The invoked routine is passed to an indication of the calling process and the random walk distance. In one embodiment, the distance is twice in the estimated diameter of the broadcast channel. In block 1614, the routine decrements the holes left to fill by two and loops to block 1613. In decision block 1616, if there is still a hole to fill, then the routine continues at block 1617, else the routine returns. In block 1617, the routine invokes the fill hole routine passing the identification of the calling process. The fill hole routine broadcasts a connection port search statement (i.e., `connection_port_search_stmt`) for a hole of a connected process through which the calling process can connect to the broadcast channel. The routine then returns.

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

FIG. 18 is a flow diagram illustrating the processing of the forward connection edge search routine in one embodiment. This routine is responsible for passing along a request to connect a requesting process to a randomly selected neighbor of this process through the internal port of the selected neighbor, that is part of the random walk. In decision block 1801, if the forwarding distance remaining is greater than zero, then the routine continues at block 1804, else the routine continues at block 1802. In decision block 1802, if the number of neighbors of this process is greater than one, then the routine continues at block 1804, else this broadcast

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

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

FIG. 20 is a flow diagram illustrating the processing of the handle port connection call routine in one embodiment. This routine is invoked when an external message is received then indicates that the sending process wants to connect to one hole of this process. In decision block **2001**, if the number of holes of this process is greater than zero, then the

routine continues at block **2002**, else the routine continues at block **2003**. In decision block **2002**, if the sending process is not a neighbor, then the routine continues at block **2004**, else the routine continues to block **2003**. In block **2003**, the routine sends a port connection response external message (i.e., `port_connection_rsp`) to the sending process that indicates that it is not okay to connect to this process. The routine then returns. In block **2004**, the routine sends a port connection response external message to the sending process that indicates that is okay to connect this process. In decision block **2005**, if the sending of the message was successful, then the routine continues at block **2006**, else the routine continues at block **2007**. In block **2006**, the routine invokes the add neighbor routine to add the sending process as a neighbor of this process and then returns. In block **2007**, the routine hangs up the external connection. In block **2008**, the routine invokes the connect request routine to request that a process connect to one of the holes of this process. The routine then returns.

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

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

the routine invokes the achieve connection routine and then continues in block 2210. In decision block 2210, if the application program message queue is empty, then the routine returns, else the routine continues at block 2212. In block 2212, the routine invokes the receive response routine passing the acquired message and then returns. The received response routine is a callback routine of the application program.

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

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

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

FIG. 27 is a flow diagram illustrating the processing of the court neighbor routine in one embodiment. This routine is passed an indication of the prospective neighbor for this process. If this process can connect to the prospective neighbor, then it sends a port connection call external message to the prospective neighbor and adds the prospective neighbor as a neighbor. In decision block 2701, if the prospective neighbor is already a neighbor, then the routine returns, else the routine continues at block 2702. In block 2702, the routine dials the prospective neighbor. In decision block 2703, if the number of holes of this process is greater than zero, then the routine continues at block 2704, else the

routine continues at block 2706. In block 2704, the routine sends a port connection call external message (i.e., port_connection_call) to the prospective neighbor and receives its response (i.e., port_connection_resp). Assuming the response is successfully received, in block 2705, the routine adds the prospective neighbor as a neighbor of this process by invoking the add neighbor routine. In block 2706, the routine hangs up with the prospect and then returns.

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

FIG. 29 is a flow diagram illustrating the processing of the handle connection edge search response routine in one embodiment. This routine is passed as indication of the requesting process, the sending neighbor, and the message. In block 2901, the routine notes that the connection edge

US 6,910,069 B1

27

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

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

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

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

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

28

ment. This routine removes an existing neighbor and connects to the process that sent the message. In decision block 3301, if this process has no holes, then the routine continues at block 3302, else the routine continues at block 3304. In block 3302, the routine selects a neighbor that is not involved in the neighbors with empty ports condition. In block 3303, the routine removes the selected neighbor as a neighbor of this process. Thus, this process that is executing the routine now has at least one hole. In block 3304, the routine invokes the add neighbor routine to add the process that sent the message as a neighbor of this process. The routine then returns.

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

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

What is claimed is:

1. A computer-based, non-routing table based, non-switch based method for adding a participant to a network of participants, each participant being connected to three or more other participants, the method comprising:

identifying a pair of participants of the network that are connected wherein a seeking participant contacts a fully connected portal computer, which in turn sends an edge connection request to a number of randomly selected neighboring participants to which the seeking participant is to connect;

disconnecting the participants of the identified pair from each other; and

connecting each participant of the identified pair of participants to the seeking participant.

2. The method of claim 1 wherein each participant is connected to 4 participants.

3. The method of claim 1 wherein the identifying of a pair includes randomly selecting a pair of participants that are connected.

29

4. The method of claim 3 wherein the randomly selecting of a pair includes sending a message through the network on a randomly selected path.

5. The method of claim 4 wherein when a participant receives the message, the participant sends the message to a randomly selected participant to which it is connected.

6. The method of claim 4 wherein the randomly selected path is proportional to the diameter of the network.

7. The method of claim 1 wherein the participant to be added requests a portal computer to initiate the identifying of the pair of participants.

8. The method of claim 7 wherein the initiating of the identifying of the pair of participants includes the portal computer sending a message to a connected participant requesting an edge connection.

9. The method of claim 8 wherein the portal computer indicates that the message is to travel a distance proportional to the diameter of the network and wherein the participant that receives the message after the message has traveled that distance is one of the participants of the identified pair of participants.

10. The method of claim 9 wherein the certain distance is twice the diameter of the network.

11. The method of claim 1 wherein the participants are connected via the Internet.

12. The method of claim 1 wherein the participants are connected via TCP/IP connections.

30

13. The method of claim 1 wherein the participants are computer processes.

14. A computer-based, non-switch based method for adding nodes to a graph that is m-regular and m-connected to maintain the graph as m-regular, where m is four or greater, the method comprising:

identifying p pairs of nodes of the graph that are connected, where p is one half of m, wherein a seeking node contacts a fully connected portal node, which in turn sends an edge connection request to a number of randomly selected neighboring nodes to which the seeking node is to connect;

disconnecting the nodes of each identified pair from each other; and

connecting each node of the identified pairs of nodes to the seeking node.

15. The method of claim 14 wherein identifying of the p pairs of nodes includes randomly selecting a pair of connected nodes.

16. The method of claim 14 wherein the nodes are computers and the connections are point-to-point communications connections.

17. The method of claim 14 wherein m is even.

* * * * *

EXHIBIT 23

**THIS EXHIBIT HAS BEEN
REDACTED IN ITS ENTIRETY**

EXHIBIT 24

**THIS EXHIBIT HAS BEEN
REDACTED IN ITS ENTIRETY**

EXHIBIT 25

Teredo @ Microsoft

Present and Future

Christopher.Palmer@Microsoft.com

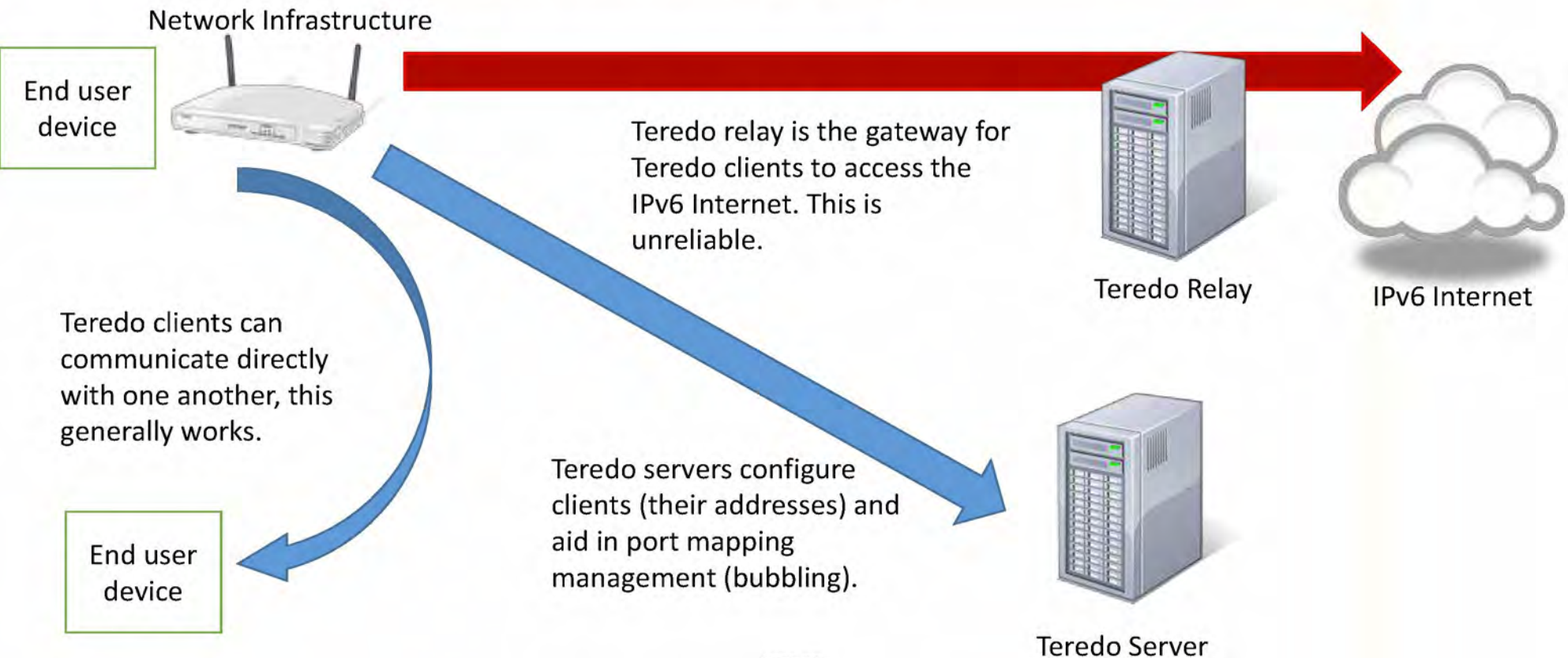
Program Manager

Networking Core – Operating System Group

Overview

- Teredo is an IPv6 transition technology that provides IPv6 addressability and connectivity for capable hosts which are on an IPv4 network but with no native connection to an IPv6 network.
 - RFC 4380, 5991, and 6081
- Microsoft has included Teredo functionality in a *default* configuration in Windows Vista, 7, and 8/8.1.
- We are simultaneously:
 - Sunsetting Teredo service for Windows Vista and Windows 7 hosts.
 - Extending Teredo support for Xbox One gaming scenarios.

Teredo – Servers and Relays



IETF 88

3

Teredo – Two Sides of the Coin

The Bad

- Teredo as a technology to reach the IPv6 native Internet lacks operational reliability.
 - Geoff Huston has considerable data on this reality.
 - <http://www.potaroo.net/ispcol/2011-04/teredo.html>
 - 40%+ effective failure rate
- Should not affect users because of RFC 3484/6724.

Teredo with relays != Reliable

The Good

- As a technology for enabling connectivity between IPv4 peers, Teredo is pretty good.
- With basic matchmaking, able to achieve connectivity between Teredo clients about 90% of the time.
- Teredo has seen successful usage in “controlled” environments such as DirectAccess (a Microsoft remote access technology).

Teredo without relays = Usable

The Teredo Service

- We don't have very specific telemetry on Teredo usage (privacy is important).
- We do know that Teredo server load had a dramatic increased - correlated to a popular BitTorrent client activating Teredo/IPv6 support.

Worldwide Teredo Server Traffic (Monthly Average - UDP Datagrams/Second)



IETF 88

6

The Overall Value of Teredo

- Teredo's value is best realized when coupled with supporting infrastructure for peer discovery, selection, and security.
 - As in, the infrastructure and API support we have for Xbox One.
- Having a tunneled IPv6 address, by itself, provides little value and causes pain for developers and end-users (because of random bad app behavior).

Proposed Sunset Plan

- We plan to deactivate our Teredo servers for Windows clients in the first half of 2014 (exact date TBD).
- Aligned to that, we encourage the deactivation of publically operated Teredo relays.
- We will maintain separate Teredo services for special-purpose scenarios that do not require public Teredo relays – like Xbox One.
- We deactivated the Teredo service earlier this year for a test. (see IETF 87 presentation)
 - Folks in the technical community seemed quite happy.
 - There were some app compat issues that we are following-up on.

Xbox One and Teredo (and IPv6)

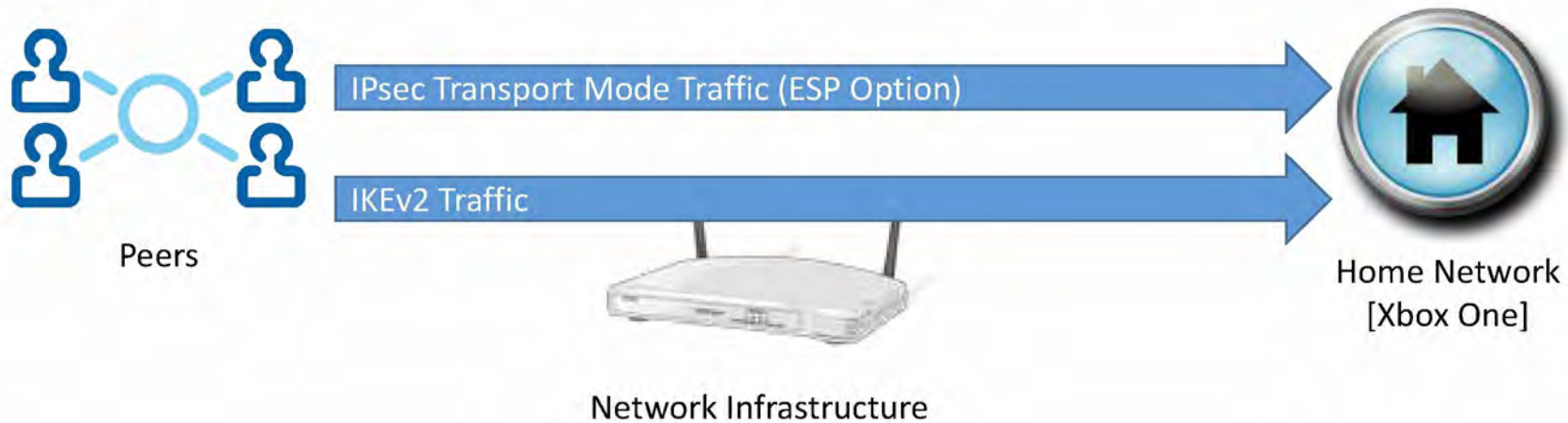
Xbox One and Teredo

- Teredo provides an IPv6 abstraction for peers.
- Combined with IPsec, this can provide straightforward, application-transparent, secure P2P connectivity.
- Xbox One uses Teredo for this purpose.

Quickly...

Going to review Xbox One behavior

IPv6 Networks: IPsec and Transparent Operation



Allow users to disable firewall capabilities (transparent operation)

Allow **unsolicited inbound** IPsec and IKE

Sometimes Teredo is more reliable for P2P than native IPv6

Xbox will consider the following peer pairs:

Teredo Client -> Teredo Client

IPv6 -> IPv6

IPv4-> IPv4

NO Teredo Client -> Native

IPv4 Networks: Allow Teredo

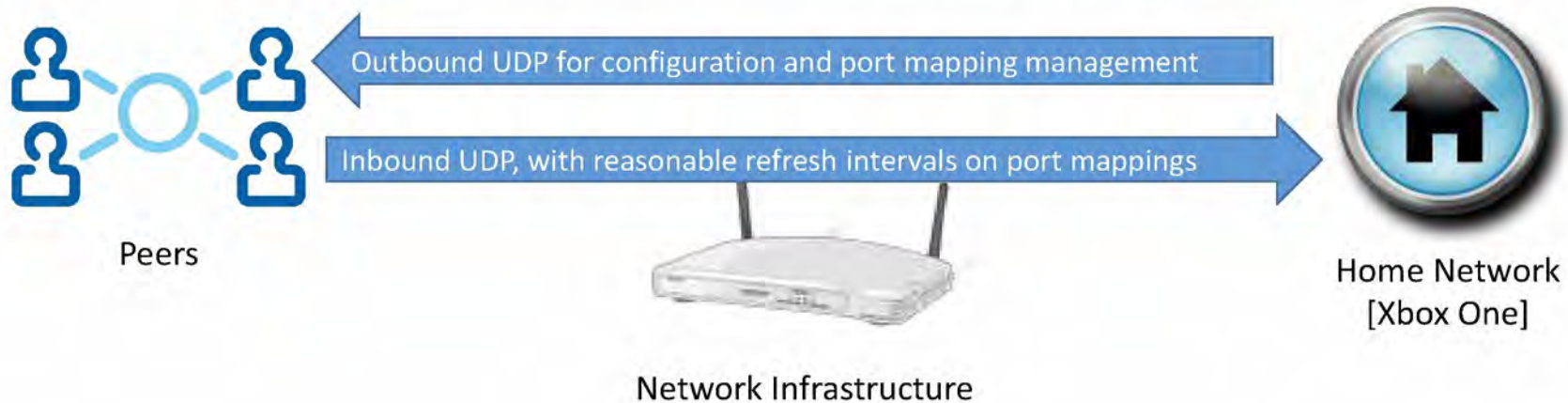
Support outbound UDP with long port mapping refresh intervals (60 seconds +)

Teredo traffic will **prefer** port 3074 for peer traffic. Port forwarding for 3074 is helpful but not necessary (usually).

The more “open” the NAT behavior, the better.

Address-Independent > Address-Dependent > Address-and-Port Dependent > UDP Blocked
with older nomenclature

Open > Address Restricted > Port Restricted > Symmetric > UDP Blocked



IETF 88

14

IPv4 Networks: Be Mindful of Hairpinning

With CGN, multiple peers may be behind the same NAT device

Hairpinning allows those peers to communicate



Network Infrastructure



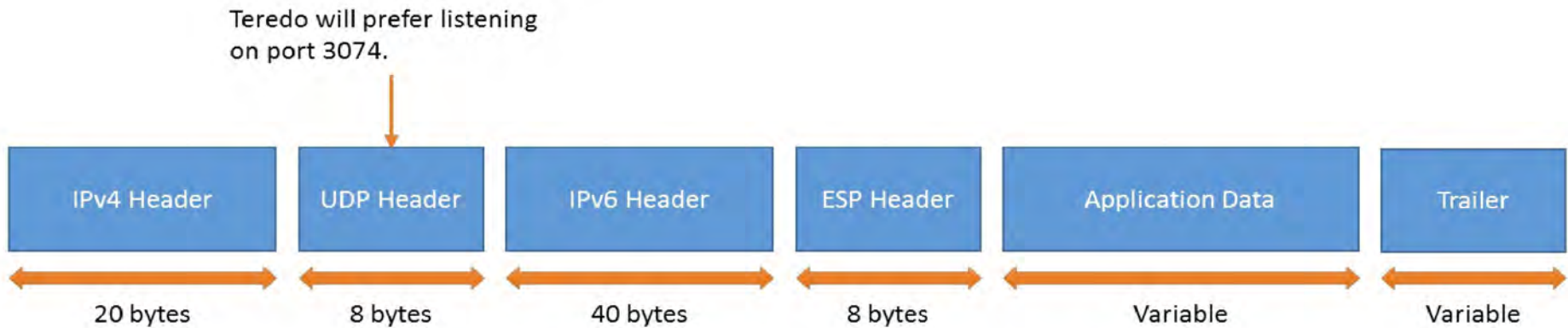
Home Network [Xbox One]



Peers

Packet Format and Native IPv4

- P2P traffic will use the ESP option for IPsec
- Native IPv4 will be used if available, generally for link-local peers.



Questions?

We will send v6ops/NANOG notice about exact Teredo service dates.

- More detailed documentation aligned to this presentation is available at www.microsoft.com/IPv6.
- Relevant RFC's
 - RFC 6092 for IPv6 security recommendations
 - RFC 4380, 5991, and 6081 for more information on Teredo
 - RFC 4787 and 6888 have recommendations for NAT behavior

EXHIBIT 26

Xbox One

Technical information on P2P Networking Behavior

Version 1.0

Last updated: October 6, 2013

Many gaming and app scenarios on Xbox® One rely on low-latency peer-to-peer (P2P) networking capabilities. The platform for these capabilities leverages **IPv6, Teredo, and IPsec Internet standards**.

The primary purpose of this document is provide technical information on Xbox One P2P behavior, so that network operators and network equipment manufacturers can address possible configuration issues and ensure end-users have a positive experience.

IPv6 and IPsec

One of the many benefits of IPv6 is simplified connectivity between peers on the Internet. Xbox One takes advantage of this, and IPv6 **may** be attempted as a connectivity mechanism between peers if available.

Proper authentication of peers and integrity of data traffic is critical to fair, safe gaming. To this purpose, transport-mode IPsec is used for most P2P traffic in most games, with IKEv2 used for security setup ([RFC 5996](#)).

[RFC 6092](#) provides useful guidelines on the security and filtering behavior for IPv6 customer premise equipment.

- **Allow IPsec and IKE**
 - In general P2P traffic will be used with IPsec with the encapsulating security payload (ESP) option. Section 3.2.4 provides guidelines on interacting with IPsec.
 - *“The Internet Protocol security (IPsec) suite offers greater flexibility and better overall security than the simple security of stateful packet filtering at network perimeters. Therefore, residential IPv6 gateways need not prohibit IPsec traffic flows.”*
 - Ensure that you are not filtering inbound IKE traffic – which is UDP on port 500 and port 4500.
- **Enable transparent operation**
 - We also note Recommendation 49. This may be relevant for users troubleshooting connectivity issues or who in general desire transparent operation.
 - *“Internet gateways with IPv6 simple security capabilities MUST provide an easily selected configuration option that permits a “transparent mode” of operation that forwards all unsolicited flows regardless of forwarding direction, i.e., not to use the IPv6 simple security capabilities of the gateway. The transparent mode of operation MAY be the default configuration.”*

Teredo

IPv6 is not yet globally available in the consumer Internet market. For the majority of users, Teredo will be used to provide P2P connectivity. Teredo is an IPv6 transition technology that provides Teredo clients a tunneled IPv6 address for P2P connectivity.

Teredo can also be used to provide connectivity to hosts on the native IPv6 Internet, using a Teredo relay. However, that capability will be rarely used by Xbox One.

The Teredo implementation on Xbox One behaves similar to that on Windows® 7 and 8.1. The design of Teredo is described in [RFC 4380](#), [RFC 5991](#), and [RFC 6081](#). We have [additional documentation on Teredo available on TechNet](#).

Even for users that do have native IPv6 – Teredo will be used to interact with IPv4-only peers, or in cases where IPv6 connectivity between peers is not functioning. In general, Xbox One will dynamically assess and use the best available connectivity method (Native IPv6, Teredo, and even IPv4). The implementation is similar in spirit to [RFC 6555](#).

For that reason, it is important for all interested network operators to understand Teredo operating requirements. Xbox One does not support operating on an IPv6-only network because of the need to reliably interoperate with nodes on IPv4-only networks.

- **Allow UDP**
 - Teredo uses UDP to create port mappings in NAT equipment and to communicate with Teredo peers. [RFC 4787](#) and [RFC 6888](#) provide recommendations on UDP behavior of network equipment. Sections 4-6 of RFC 4787 are especially relevant to network operators and equipment vendors.
- **Configure open NATs, avoid symmetric**
 - RFC 6081, Section 3, provides a connectivity matrix for Teredo. In general, cone, address restricted, and port restricted NATs, work fine with Teredo. Networks that block UDP traffic will not support P2P gaming. The more “open” a NAT, the more likely the user will be able to communicate directly with others, and in general the better experience the user will have.
- **Avoid requiring frequent keep-alives**
 - Even with networks that support UDP, if the keep-alive interval for UDP port mappings is less than 60 seconds – then connectivity issues may arise for end-users.
 - RFC 4787 recommends 5 minutes.
 - Teredo will refresh port mappings approximately every 30 seconds.
- **Support hairpinning**
 - Networks should support hairpinning (Section 6 of RFC 4787), to allow multiple Teredo clients on the same network to communicate with one another. Compliance with this recommendation is especially important for network operators who are deploying carrier-grade NAT functionality.
- **Don't block encapsulated IPsec**
 - With Teredo, IPv6 packets are encapsulated in an IPv4 UDP datagram. These IPv6 packets will be secured with IPsec, as described earlier.

- o If for any reason a network administrator performs de-encapsulation of Teredo traffic for inspection purposes, they should allow IPsec traffic, and generally follow the guidelines provided earlier for IPv6 security.

Teredo Ports

Teredo clients will initiate contact with a Teredo server that is listening on port 3544, using UDP, as part of the Teredo client qualification procedure. The local port is from the dynamic range. Enabling port forwarding for port 3544 is not necessary or helpful.

For actual P2P traffic, Teredo will attempt to use local port 3074, again with UDP. This is the same port that Xbox 360 uses. Enabling inbound communication to this port using explicit port forwarding rules is helpful, though rarely required. Teredo will attempt to use UPnP and UDP messages to ensure inbound connectivity to the Teredo port.

If 3074 is unavailable, a local port from the dynamic port range (49152-65535) will be used – and again Teredo will attempt to automatically provide inbound connectivity without the need for explicit port forwarding rules.

Xbox One IPv4 P2P traffic may be distinguished from other types of IPv4 traffic by identifying:

1. The **typical** usage of port 3074 in a UDP datagram.
2. That the traffic is embedding Teredo packets.
 - a. One can identify this by looking for the four bytes for the 2001:0000 prefix in the IPv6 header.
3. That the embedded Teredo packets use an ESP header.

We expect this **combination** should be fairly unique to Xbox One. Items 2 and 3 can be used to distinguish Xbox One traffic from Xbox 360 traffic, and items 1 and 3 can be used to distinguish Xbox One traffic from common Teredo traffic. We recommend network administrators treat Xbox One and Xbox 360 traffic similarly.

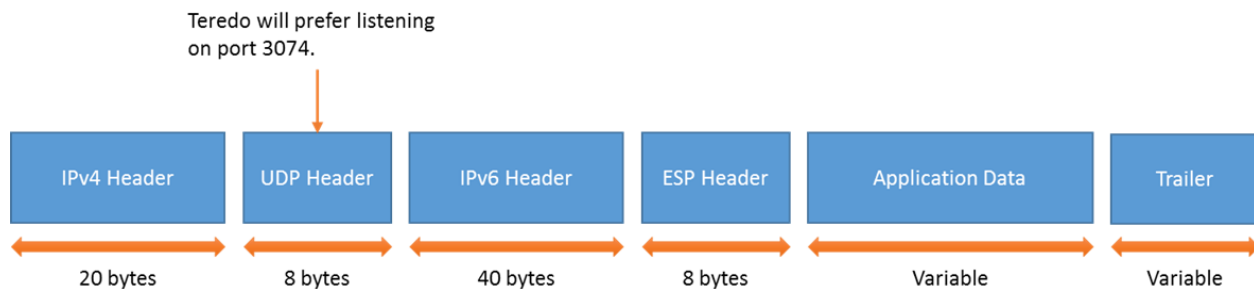


Figure 1 - Packet format with Teredo and IPsec, as used by Xbox One.

Monitoring and Support

If authorized by the end-user, Microsoft® collects telemetry on the performance and reliability of P2P networking on Xbox One. This information will be used to engage with network operators if geographically specific issues are found.

Xbox One includes a network troubleshooter, which can inform the user if their network is not compliant with the above requirements.

Network operators and equipment vendors can address questions and concerns to xboxteredo@microsoft.com.

EXHIBIT 27

**THIS EXHIBIT HAS BEEN
REDACTED IN ITS ENTIRETY**

EXHIBIT 28

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

IN THE UNITED STATES DISTRICT COURT
IN AND FOR THE DISTRICT OF DELAWARE

- - -

ACCELERATION BAY, LLC,	:	CIVIL ACTION
	:	
Plaintiff,	:	
	:	
vs.	:	
	:	
ACTIVISION BLIZZARD,	:	
	:	
Defendant.	:	NO. 16-0453-RGA
-----	:	
ACCELERATION BAY, LLC,	:	CIVIL ACTION
	:	
Plaintiff,	:	
	:	
vs.	:	
	:	
ELECTRONIC ARTS INC.,	:	
	:	
Defendant.	:	NO. 16-0454-RGA

- - -

Wilmington, Delaware
Monday, December 18, 2017
9:03 o'clock, a.m.

- - -

BEFORE: HONORABLE RICHARD G. ANDREWS, U.S.D.C.J.

- - -

Valerie J. Gunning
Official Court Reporter

1 ACCELERATION BAY, LLC, : CIVIL ACTION
 2 Plaintiff, :
 3 vs. :
 4 TAKE-TWO INTERACTIVE :
 5 SOFTWARE, INC., ROCKSTAR :
 6 GAMES, INC. and 2K SPORTS, :
 7 INC., :
 8 Defendants. : NO. 16-0455-RGA
 9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25

1 APPEARANCES (Continued):
 2
 3 WINSTON & STRAWN LLP
 4 BY: THOMAS M. DUNHAM, ESQ.
 5 (Washington, D.C.)
 6
 7 Counsel for Defendant
 8
 9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25

3

1 APPEARANCES:
 2
 3 POTTER, ANDERSON & CORROON LLP
 4 BY: ALAN SILVERSTEIN, ESQ.
 5
 6 -and-
 7
 8 KRAMER LEVIN NAFTALIS & FRANKEL LLP
 9 BY: JAMES HANNAH, ESQ.
 10 (Menlo Park, California)
 11
 12 -and-
 13
 14 KRAMER LEVIN NAFTALIS & FRANKEL LLP
 15 BY: AARON M. FRANKEL, ESQ.
 16 (New York, New York)
 17
 18 Counsel for Plaintiff
 19 Acceleration Bay LLC
 20
 21
 22 MORRIS, NICHOLS, ARSHT & TUNNELL LLP
 23 BY: JACK B. BLUMENFELD, ESQ.
 24
 25 -and-
 26
 27 WINSTON & STRAWN LLP
 28 BY: MICHAEL A. TOMASULO, ESQ.
 29 (Los Angeles, California)
 30
 31 -and-
 32
 33
 34
 35

5

1 PROCEEDINGS
 2
 3 THE COURT: All right. Good morning, everyone.
 4 Please be seated.
 5 This is Acceleration Bay versus Activision
 6 Blizzard, Inc., Civil Action No. 16-453, plus the two
 7 related cases.
 8 Mr. Silverstein?
 9 MR. SILVERSTEIN: Good morning, Your Honor.
 10 THE COURT: Good morning.
 11 MR. SILVERSTEIN: Good morning. Alan
 12 Silverstein, Potter Anderson. With me today is James
 13 Hannah.
 14 MR. HANNAH: Good morning, Your Honor.
 15 MR. SILVERSTEIN: And Aaron Frankel from Kramer
 16 Levin.
 17 THE COURT: Good morning, gentlemen.
 18 Mr. Blumenfeld?
 19 MR. BLUMENFELD: Good morning, Your Honor. Jack
 20 Blumenfeld from Morris Nichols for the defendants along with
 21 Tom Dunham and Mike Tomasulo from Winston & Strawn, and
 22 Linda Zabriskie from Take-Two is a few rows back.
 23 THE COURT: All right. Good morning to you all.
 24 MR. TOMASULO: Good morning.
 25 THE COURT: All right. Mr. Hannah?

1 immediately establish the connection. In other words, I
2 don't really know what port the portable computer has
3 allocated because I have not yet made contact. So I know
4 the address of the portal computer, but I don't know what of
5 the 65,000 ports. And so if I were to just start randomly
6 guessing at them, that would look like a firewall attack.

7 That would look like a hacker.

8 So what the invention was, was that there would
9 be a port ordering algorithm. And so -- this created a
10 little figure here. So we have this portal computer and
11 it's running the broadcast channel. So it runs this
12 algorithm to generate a specific list of ports that it will
13 be considering using. So maybe it six the 65,000 ports. It
14 runs this algorithm, and it generates a specific list of
15 let's say ten ports. And so now the seeking computer runs
16 the same algorithm, gets the same list, and instead of
17 having to guess through 65,000 ports, it has now reduced the
18 port space that it can guess on down to, let's say ten.

19 Okay.

20 And so then the portal, the seeking computer
21 tries to contact the portal computer on that list of ten --
22 that list of ten ports is my example. And then the
23 algorithm is a deterministic algorithm. When the seeking
24 computer and the portal computer both put in the same
25 broadcast channel identifier, they get the same list of

1 added where the port ordering algorithm is used to identify
2 the call-in port, and wherein the communications ports, a
3 list of ports can be reordered. So there are two parts of
4 it. Basically, the algorithm generates a list of ten, and
5 then you can then subsequently reorder that if it turns out
6 it's crowded.

7 So when they made that amendment, they explained
8 in great detail what a port ordering algorithm was not.
9 And what they said a port ordering algorithm was not is
10 something that is random.

11 So they say here, Kayashima -- first they are
12 saying why they're different than Kayashima. They say
13 Kayashima fails to disclose a method for locating a call-in
14 port in which quote a port ordering algorithm is used to
15 identify -- that's the language they added. Instead
16 they say, Kayashima discloses a method in which a
17 portal -- computer's communications ports are randomly
18 dialed. Then they go on to say why that's different. The
19 say, the use of a port ordering algorithm -- in other words,
20 that's what the claim limitation is, the port ordering
21 algorithm, minimizes the time required to locate the call-in
22 port of the portable computer, which is what we said, they
23 reduce the port down to let's say ten ports, by identifying
24 a non-random port number order that a portal computer should
25 use when finding an available port for its call-in port.

1 ports. That's the whole point, is to reduce the amount of
2 guessing that the seeking computer has to do from 65,000
3 ports down to let's say ten ports. And so that's the
4 purpose of the invention. That's the solution that the
5 inventors came up with, because they did not want to have to
6 go to a server and have the server say, oh, you contact the
7 portal computer. Here's where the portal computer's full
8 address is.

9 We go to a server. The portal computer address
10 is ABC, and it's operating on Port 3250. Now you're ready
11 to go. They wanted to have a completely distributed system,
12 and so this was the way that they came up with to reduce the
13 amount of guessing that the seeking computer would have to
14 do to contact the portal computer.

15 So their original claims were rejected as
16 anticipated by Kayashima, and just to speed things up, I
17 won't go through what the Examiner explained about
18 Kayashima, but the original claims were quite broad. The
19 Examiner recited chapter and verse how those claims were met
20 by Kayashima. Then to overcome the anticipation and the
21 obviousness rejection for the purpose of patentability, they
22 amended the claims to add the port ordering algorithm
23 limitation. These are the last two limitations here, port
24 ordering algorithm and port ordering algorithm. So they
25 added these to overcome the Kayashima rejection. And they

1 And then they say those are the amendments. Down here they
2 explain that these were added and that overcomes the
3 Kayashima rejections. There is no question that these were
4 added and then explains to overcome the examiner's
5 rejection.

6 And then they go on to explain, if that wasn't
7 clear enough, in the context of the obviousness objection,
8 they explain it even further. They say, possibly more
9 important, Kayashima teaches away from the claimed
10 invention. Under Kayashima, a portal computer's
11 communications ports are randomly dialed to find the call-in
12 port. The presently claimed invention, and then they define
13 what that is, which utilizes a non-random port ordering
14 algorithm. So they define the invention. The presently
15 claimed invention uses a non-random port ordering algorithm
16 to minimize the time to overcome the problems associated
17 with Kayashima.

18 Randomness would defeat the purpose of the
19 invention. The problem, as you recall, is the 65,000 ports.
20 It would take too long to guess them all.

21 THE COURT: And I'm sorry. This is probably
22 not important, but where is the number 65,000 coming from?

23 MR. TOMASULO: It's in the specification.

24 THE COURT: Okay.

25 MR. TOMASULO: I guess that's how many there

EXHIBIT 29

**THIS EXHIBIT HAS BEEN
REDACTED IN ITS ENTIRETY**

EXHIBIT 30

PRINCETON DIGITAL IMAGE CORPORATION,	:	
	:	
Plaintiff,	:	
	:	
v.	:	C.A. No. 13-408-LPS
	:	
NORDSTROM.COM LLC,	:	
NORDSTROM.COM INC., and NORDSTROM INC.	:	
	:	
Defendants.	:	

Sean T. O’Kelly, George Pazuniak, and Daniel P. Murray, O’KELLY ERNST & JOYCE, LLC, Wilmington, DE

Attorneys for Plaintiff.

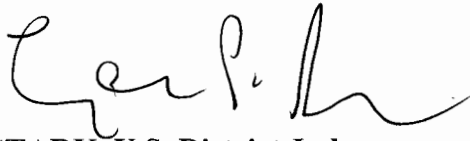
Kelly E. Farnan, RICHARDS LAYTON & FINGER, Wilmington, DE

Tara D. Elliott, Rachel Weiner Cohen, and Brittany Amadi, WILMER CUTLER PICKERING HALE AND DORR LLP, Washington, DC

Attorneys for Intervenor Adobe Systems Incorporated.

MEMORANDUM OPINION

August 1, 2017
Wilmington, Delaware



STARK, U.S. District Judge:

On June 6, 2011, Adobe Systems Incorporated (“Adobe”) and Princeton Digital Image Corporation (“PDIC”) entered into a licensing agreement concerning U.S. Patent No. 4,813,056, whereby PDIC agreed not to sue Adobe or its licensees, customers, and end users on any of PDIC’s patents. (See C.A. No. 13-239-LPS D.I. 196 Ex. 1)¹ In 2013, PDIC brought numerous suits against internet retailers (“Defendants”) for infringement of the ’056 patent, based on the encoding of JPEG images on their websites. (E.g., D.I. 1) A number of the defendants claimed to be Adobe’s licensed customers and sought indemnity and defense from Adobe. As a result, Adobe moved to intervene in 2014. (D.I. 11) In May 2015, the Court granted Adobe’s motion, and Adobe filed a complaint in intervention, alleging, among other things, that PDIC breached its contract with Adobe by suing Adobe’s customers. (D.I. 47) Subsequently, PDIC dismissed with prejudice the underlying patent infringement claims, leaving only the claims between Adobe and PDIC. (E.g., D.I. 56)

Presently before the Court are PDIC’s motion to exclude certain expert testimony and motion for summary judgment. A jury trial is scheduled to begin August 21, 2017.

For the reasons stated below, the Court will deny PDIC’s motion to exclude expert opinions and grant in part and deny in part PDIC’s motion for summary judgment.

I. LEGAL STANDARDS

A. *Daubert* Motion

In *Daubert v. Merrell Dow Pharmaceuticals, Inc.*, 509 U.S. 579, 597 (1993), the Supreme Court explained that Federal Rule of Evidence 702 creates “a gatekeeping role for the

¹Unless otherwise noted, the references to the docket are to C.A. No. 13-239-LPS.

[trial] judge” in order to “ensur[e] that an expert’s testimony both rests on a reliable foundation and is relevant to the task at hand.” Rule 702(a) requires that expert testimony “help the trier of fact to understand the evidence or to determine a fact in issue.” Expert testimony is admissible only if “the testimony is based on sufficient facts or data,” “the testimony is the product of reliable principles and methods,” and “the expert has reliably applied the principles and methods to the facts of the case.” Fed. R. Evid. 702(b)-(d).

There are three distinct requirements for proper expert testimony: (1) the expert must be qualified; (2) the opinion must be reliable; and (3) the expert’s opinion must relate to the facts. *See Elcock v. Kmart Corp.*, 233 F.3d 734, 741 (3d Cir. 2000).

B. Summary Judgment

Under Rule 56(a) of the Federal Rules of Civil Procedure, “[t]he court shall grant summary judgment if the movant shows that there is no genuine dispute as to any material fact and the movant is entitled to judgment as a matter of law.” The moving party bears the burden of demonstrating the absence of a genuine issue of material fact. *See Matsushita Elec. Indus. Co., Ltd. v. Zenith Radio Corp.*, 475 U.S. 574, 585-86 (1986). An assertion that a fact cannot be – or, alternatively, is – genuinely disputed must be supported either by “citing to particular parts of materials in the record, including depositions, documents, electronically stored information, affidavits or declarations, stipulations (including those made for purposes of the motion only), admissions, interrogatory answers, or other materials,” or by “showing that the materials cited do not establish the absence or presence of a genuine dispute, or that an adverse party cannot produce admissible evidence to support the fact.” Fed. R. Civ. P. 56(c)(1)(A) & (B). If the moving party has carried its burden, the nonmovant must then “come forward with specific facts

showing that there is a genuine issue for trial.” *Matsushita*, 475 U.S. at 587 (internal quotation marks omitted). The Court will “draw all reasonable inferences in favor of the nonmoving party, and it may not make credibility determinations or weigh the evidence.” *Reeves v. Sanderson Plumbing Prods., Inc.*, 530 U.S. 133, 150 (2000).

To defeat a motion for summary judgment, the nonmoving party must “do more than simply show that there is some metaphysical doubt as to the material facts.” *Matsushita*, 475 U.S. at 586; *see also Podobnik v. U.S. Postal Serv.*, 409 F.3d 584, 594 (3d Cir. 2005) (stating party opposing summary judgment “must present more than just bare assertions, conclusory allegations or suspicions to show the existence of a genuine issue”) (internal quotation marks omitted). The “mere existence of some alleged factual dispute between the parties will not defeat an otherwise properly supported motion for summary judgment;” a factual dispute is genuine only where “the evidence is such that a reasonable jury could return a verdict for the nonmoving party.” *Anderson v. Liberty Lobby, Inc.*, 477 U.S. 242, 247-48 (1986). “If the evidence is merely colorable, or is not significantly probative, summary judgment may be granted.” *Id.* at 249-50 (internal citations omitted); *see also Celotex Corp. v. Catrett*, 477 U.S. 317, 322 (1986) (stating entry of summary judgment is mandated “against a party who fails to make a showing sufficient to establish the existence of an element essential to that party’s case, and on which that party will bear the burden of proof at trial”). Thus, the “mere existence of a scintilla of evidence” in support of the nonmoving party’s position is insufficient to defeat a motion for summary judgment; there must be “evidence on which the jury could reasonably find” for the nonmoving party. *Anderson*, 477 U.S. at 252.

II. DISCUSSION

A. PDIC's Motion to Exclude Certain Testimony of Adobe's Experts

PDIC moves to exclude certain opinions of Dr. Andrew B. Lippman, Adobe's expert on liability, and Dr. Stephen L. Becker, Adobe's damages expert. PDIC contends that both experts' opinions improperly rely on letters and declarations that are inadmissible hearsay. PDIC raises additional challenges specific to each expert, contending that Dr. Lippman should not be allowed to testify about a Wayback Machine investigation, Defendants' activities, or Scene7, and that Dr. Becker should not be permitted to testify about Defendants' activities or restitution. The Court disagrees with each of PDIC's positions and will deny PDIC's motion.²

1. Reliance on Letters and Declarations

PDIC first contends that both experts' use of hearsay documents is improper. Federal Rule of Evidence 703 "permits experts to rely on hearsay so long as that hearsay is of the kind normally employed by experts in the field." *In re TMI Litig.*, 193 F.3d 613, 697 (3d Cir. 1999). The Court finds Drs. Lippman and Becker's use of alleged hearsay here to be compliant with Rule 703 and, hence, permissible. *See Inline Connection Corp. v. AOL Time Warner Inc.*, 470 F. Supp. 2d 435, 442 (D. Del. 2007).

Dr. Lippman was "asked to review and analyze the evidence that PDIC considered in preparing to file lawsuits asserting infringement of the '056 patent against Adobe's customers" and "to assess and evaluate PDIC's contentions that infringement by Adobe's customers occurs at the web server and does not involve an Adobe product." (D.I. 197 Ex. 8 at ¶ 18) To form his

²Adobe contends that PDIC failed to comply with Local Rule 7.1.1, which it views as fatal. (D.I. 199 at 19) As PDIC's motion was filed in compliance with the scheduling order (D.I. 106 at 3), and as the Court is denying it, the Court will consider the motion on the merits.

opinions, Dr. Lippman considered evidence of record, including testimony from Thomas and William Meagher about PDIC's presuit investigation (*see id.* at ¶ 19), as well as letters and declarations from Defendants about how their technology worked (*see id.* at ¶¶ 84-85). In forming his opinions, Dr. Lippman reasonably relied on relevant evidence produced by the parties. The Court is persuaded that this is a type of evidence on which an expert in the field providing an opinion on these topics would rely. *See TMI Litig.*, 193 F.3d at 697.

The same is true with respect to Dr. Becker. Dr. Becker was asked "to offer expert opinions on the amount of damages suffered by Adobe as a result of PDIC's breach of the covenant not to sue provision of the Adobe License." (D.I. 197 Ex. 6 at ¶ 3) His report includes communications between the parties during the relevant period. For example, Dr. Becker cites to various Defendants' requests to Adobe for indemnification. (*See, e.g., id.* at ¶ 44) Dr. Becker's reliance on the letters and declarations is permissible. He was tasked with determining the damages attributable to PDIC's alleged breach, making it entirely reasonable to examine the events and communications between the parties. This is a type of evidence on which a damages expert ordinarily relies.

It is a separate question whether evidence underlying the experts' opinions is admissible and may be presented to the jury. The Court does not at this point resolve this question. It is enough to find, as the Court does, that even if the letters and declarations are hearsay, and even if they are not admitted at trial, "their probative value in helping the jury evaluate the opinion substantially outweighs their prejudicial effect," rendering them proper for the jury to consider in evaluating the credibility of the experts, consistent with Rule 703. Although this evidence consists of documents that resulted from litigation and were "prepared in an adversarial context"

(D.I. 195 at 7), these characteristics do not (here) result in unfair prejudice to PDIC, particularly as the entire basis for Adobe's claim is that PDIC breached its contract by engaging in adversarial litigation against Adobe's customers. It is not unfair for Adobe to use documents stemming from that allegedly breaching conduct, i.e., the patent infringement suits.

2. Dr. Lippman

None of PDIC's specific attacks on Dr. Lippman's opinions provide a meritorious basis for exclusion. PDIC contends that Dr. Lippman's expert report does not adequately disclose an investigation using the Wayback Machine (a publicly-available internet archive) or the results of that investigation. Dr. Lippman's report, however, details PDIC's pre-suit investigation using the Wayback Machine (*see* D.I. 197 Ex. 8 at ¶¶ 58-64), describes various types of metadata associated with images that allows for identification of the software used to create an image (*see id.* at ¶¶ 117-22), and describes what information PDIC could have gleaned from its Wayback Machine investigation had it looked at metadata (*see id.* at ¶¶ 136-37). In his deposition, Dr. Lippman testified that he did some Wayback Machine searches "to see if [he] could determine the source of those images." (D.I. 197 Ex. 7 at 43) Dr. Lippman's report fairly discloses the theory on which he relies (*see, e.g.*, D.I. 197 Ex. 8 at ¶ 137), and his deposition testimony shows that he performed some minimum, additional tests to confirm the veracity of his opinion. While PDIC characterizes this additional testing as irrelevant, since "Dr. Lippman reviewed merely a handful of images on selected websites" (D.I. 195 at 8), PDIC is free to cross-examine Dr. Lippman on these issues at trial.

PDIC next contends that Dr. Lippman should not be allowed to testify that any of the Defendants used any particular technology because his opinion was formed on the basis of

inadmissible letters and declarations. This argument is simply a repeat of PDIC's general argument that the experts cannot rely on hearsay in forming their opinions, and the Court rejects it for the same reasons as given above.

Finally, PDIC asserts that Dr. Lippman cannot properly testify about Scene7, contending that he has no personal knowledge of its functionality as of 2007. Dr. Lippman, a Ph.D. electrical engineer and a senior research scientist at MIT (*see* D.I. 197 Ex. 8 at ¶¶ 2-16), is qualified to offer an opinion here, and his report describes Adobe's acquisition of Scene7, general Scene7 capabilities, and his opinion that use of Scene7 was licensed (*see, e.g., id.* at ¶¶ 31, 94-95). Any alleged shortcoming or deficiency in Dr. Lippman's opinion is an issue of weight and credibility appropriately addressed through cross-examination and/or the presentation of contrary evidence.

3. Dr. Becker

PDIC contends that "Dr. Becker's expert report indicates a possibility that he intends to testify that some or all of the Defendants had been using Adobe Products in order to provide the enlarged or on-the-fly images on their websites," which PDIC asserts is inappropriate because Dr. Becker has no personal knowledge of the technology. (D.I. 195 at 11) PDIC does not, however, identify any specific portions of Dr. Becker's report that it is challenging. Accordingly, Dr. Becker will be allowed to testify consistent with the opinions expressed in his expert report.

PDIC also contends that Dr. Becker's understanding of restitution is contrary to New Jersey law. Although there is no indication that Dr. Becker's understanding of how to measure restitution is inconsistent with governing law, the Court need not reach this issue. As discussed below, the Court finds that restitution is not an available remedy in this case.

B. PDIC's Motion for Summary Judgment

Under New Jersey law, which the Court has determined governs the agreement between PDIC and Adobe, *see Princeton Digital Image Corp. v. Office Depot Inc.*, 2016 WL 1533697, at *6 (D. Del. Mar. 31, 2016), a party asserting breach of contract must show that: (1) a contract existed between the parties; (2) the defendant breached a duty imposed by the contract, (3) the plaintiff performed its obligations under the contract, and (4) the plaintiff was damaged as a result of the breach. *See Miller v. Butler*, 2014 WL 585409, at *3 (D.N.J. Feb. 14, 2014). PDIC moves for summary judgment on the basis that Adobe cannot show that PDIC breached the contract or that Adobe is entitled to damages for any breach. PDIC also seeks to eliminate restitution as a potential alternative theory of Adobe's recoverable damages.

1. Breach of Contract

The parties' agreement "grants to Adobe and Adobe Related Entities an irrevocable, unrestricted, fully paid-up, perpetual covenant not to sue Adobe or Adobe Related Entities with respect to any and all past and present claims . . . based on any of the Princeton patents." (D.I. 196 Ex. 1 at ¶ 3.1) Further, "[t]he covenant not to sue exhausts all claims of all Princeton Patents . . . as to any Licensed Product to the extent that the claims arose in whole or part owing to an Adobe Licensed Product." (*Id.*) Adobe alleges that PDIC breached this covenant not to sue by bringing infringement actions against Defendants, who are Adobe's customers using Adobe products.

PDIC contends that summary judgment is appropriate because Adobe cannot demonstrate that PDIC breached the contract. In particular, PDIC asserts that it "never alleged any infringement against any Adobe product or service, and did not accuse any Defendant of

infringement based on that Defendant’s use of an Adobe product or service.” (D.I. 193 at 6) Moreover, PDIC argues that Adobe cannot prove that any Defendant actually used Adobe products or services in performing the allegedly infringing acts.

In opposing summary judgment, Adobe contends that PDIC’s infringement allegations were written broadly and that PDIC’s own statements acknowledge that the complaints can be read to cover Adobe products. Adobe asserts that there is sufficient evidence of record to conclude that Defendants did use Adobe products. Further, according to Adobe, PDIC’s contention that Defendants used non-Adobe products is merely speculative and is disputed by Adobe’s expert.

The Court agrees with Adobe that there are genuine issues of material fact precluding summary judgment. A reasonable juror could accept Adobe’s view that PDIC’s infringement allegations – which generally accused Defendants of infringement based on their “encod[ing] image data in JPEG files for purposes of producing JPEG images of products” (D.I. 1 at ¶ 9) – cover the use of Adobe products. A reasonable juror could also conclude that Defendants did, indeed, use Adobe products, based on Adobe’s expert and PDIC’s admissions. (*See* D.I. 197 Ex. 8 at ¶¶ 135-40; D.I. 73 at 1 (“[T]hree Defendants appear to have used solely Adobe’s Scene7 software for the activities accused of infringement”)) There are also genuine factual disputes about PDIC’s theory of Defendants’ use of non-Adobe products, which a jury will have to resolve by weighing evidence and evaluating witness credibility. (*See* D.I. 181 Ex. 1 at 190-91; D.I. 197 Ex. 8 at ¶ 135)

Accordingly, the Court cannot grant summary judgment based on the purported failure to adduce evidence sufficient to support a reasonable finding of breach.

2. Damages

a. Attorney Fees

Adobe seeks consequential damages stemming from PDIC's alleged breach, claiming that appropriate damages include both attorney fees incurred in defending Defendants against infringement ("defense fees") and attorney fees incurred in pressing Adobe's breach of contract claims ("affirmative fees"). PDIC contends that the American Rule, and New Jersey application of that Rule, prevents Adobe from claiming any attorney fees as consequential damages. PDIC also asserts that even if defense fees are a proper measure of damages, Adobe cannot prove the amount of its defense fees because its expert has not separated defense fees from affirmative fees.

Under New Jersey contract law, "a party who breaches a contract is liable for all of the natural and probable consequences of the breach of that contract." *Pickett v. Lloyd's*, 621 A.2d 445, 454 (N.J. 1993) "In the absence of a contrary agreement, an injured party with the legal right to be compensated for the breach of a contract is entitled to the amount of damages . . . which, ***excluding attorneys' fees and court costs for prosecuting the breach of contract action itself***, will put that party in the same position it would have been in if the breaching party had performed the contract in accordance with its terms, no better position and no worse." *Magnet Res., Inc. v. Summit MRI, Inc.*, 723 A.2d 976, 985 (N.J. Super. Ct. App. Div. 1998) (emphasis added). Accordingly, consistent with the American Rule that "the prevailing litigant is ordinarily not entitled to collect a reasonable attorneys' fee from the loser," *N. Bergen Rex Transp., Inc. v. Trailer Leasing Co.*, 730 A.2d 843, 848 (N.J. 1999), attorney fees incurred in bringing a breach of contract action "are not normally a proper measure of contract damages," *New Flyer of Am., Inc. v. Mid-Newark, L.P.*, 2010 WL 2794249, at *6 (N.J. Super. Ct. App. Div. July 6, 2010).

Otherwise, attorney fees would be consequential damages in nearly every contract claim, as one natural and probable consequence of a breached contract is that a party will bring a lawsuit. *See id.* at *7.

Courts have recognized exceptions to this general rule, often involving circumstances in which a non-breaching party engages in legal proceedings as a result of the breached contract, apart from the breach of contract suit. For example, a breach of a covenant not to sue may result in an underlying suit (i.e., the suit that breaches the contract), as well as a breach of contract suit. Attorney fees may be “a proper element of damages when the right violated is the right to be free from suit.” *Riveredge Assocs. v. Metro. Life Ins. Co.*, 774 F. Supp. 897, 901 (D.N.J. 1991). But courts generally limit damages under this theory to those “attorneys’ fees and other costs of ***defending against a wrongful action.***” *Id.* at 902 (emphasis added); *Gerhardt v. Cont’l Ins. Cos.*, 225 A.2d 328, 334 (N.J. 1966) (“Since the insurer defaulted on its obligation to defend the insured in the workmen’s compensation proceeding, it must bear, as a traditional element of damage, the expense including the reasonable counsel fee incurred by the insured in defending there in its stead.”); *Verhagen v. Platt*, 61 A.2d 892, 895 (N.J. 1948) (“If a breach of contract is the cause of litigation between the plaintiff and third parties that the defendant had reason to foresee when the contract was made, the plaintiff’s reasonable expenditures in such litigation are included in estimating his damages.”); *see also Am. Home Assur. Co. v. United Space All., LLC*, 378 F.3d 482, 490 (5th Cir. 2004) (“[T]he recovery of attorney’s fees in such circumstances are appropriately based upon the equitable ground that the claimant was required to defend against litigation as a consequence of the wrongful conduct of the defendant.”); *Anchor Motor Freight, Inc. v. Int’l Bhd. of Teamsters, Chauffeurs, Warehousemen & Helpers of Am.*, 700 F.2d 1067,

1072 (6th Cir. 1983) (“The Union is not precluded . . . from recovering costs incurred in defending against an action filed in breach of a covenant not to sue.”); *Prospect Energy Corp. v. Dallas Gas Partners, LP*, 761 F. Supp. 2d 579, 595-96 (S.D. Tex. 2011) (“When a covenant not to sue is used offensively in a counterclaim for breach of contract, the counterclaimants’ attorneys’ fees cannot be considered actual damages, but are incidental costs of litigation.”).

Because this exception displaces the strong background presumption that parties will carry their own costs in litigation, some courts have also limited these damages to those that arise from an obvious or bad-faith breach of a covenant not to sue (unless there is an indication that the parties intended for the contract to award other damages). *See Riveredge*, 774 F. Supp. at 902 (“If the covenant breached is a promise not to file suit in bad faith, then the logical measure of damages is the direct consequence of breach of that right: the costs of defending the wrongful suit.”); *see also Lubrizol Corp. v. Exxon Corp.*, 957 F.2d 1302, 1306-07 (5th Cir. 1992) (“Lubrizol’s breach of the parties’ covenant not to sue was obvious and . . . the district court rightfully imposed liability”); *Cefali v. Buffalo Brass Co.*, 748 F. Supp. 1011, 1027 (W.D.N.Y. 1990) (“[W]hether fees should be awarded therefore depends on whether the suit was brought ‘in obvious breach or otherwise in bad faith.’”). Although not all courts have adopted this view, *see, e.g., Widener v. Arco Oil & Gas Co., Div. of Atl. Richfield Co.*, 717 F. Supp. 1211, 1217 (N.D. Tex. 1989), the parties identify no authority directly on point, and at least two federal courts predicting New Jersey law on this issue have applied the bad-faith or obvious breach requirement. *See Riveredge*, 774 F. Supp. at 902 (requiring showing of bad faith for attorney fee award on implied covenant not to sue); *Borbely v. Nationwide Mut. Ins. Co.*, 547 F. Supp. 959, 980 (D.N.J. 1981) (requiring showing of bad faith for attorney fee award on express covenant not

to sue). The “primary function” of a covenant not to sue “is to serve as a shield rather than as a sword.” *Artvale, Inc. v. Rugby Fabrics Corp.*, 363 F.2d 1002, 1008 (2d Cir. 1966). Thus, “[i]n the absence of contrary evidence, sufficient effect is given the usual covenant not to sue if, in addition to its service as a defense, it is read as imposing liability only for suits brought in obvious breach or otherwise in bad faith.” *Id.*

Applying this law here, the Court agrees with PDIC that Adobe cannot collect as damages for the breach of contract any attorney fees Adobe incurred in the affirmative breach-of-contract suit. These affirmative fees are not fees that arose because Adobe had to “defend[] against a wrongful action,” *Riveredge*, 774 F. Supp. at 902, but are instead the classic type of attorney fees that Adobe, in attempting to vindicate its contract rights, must bear itself. There is no indication that the parties agreed to an arrangement inconsistent with this general presumption.

However, the defense fees – that is, those Adobe incurred in defending Defendants from PDIC’s infringement suit, suits that were brought in alleged violation of the covenant not to sue – are the type of fees that may be awarded as damages. Adobe’s legal costs in defending its customers are a proper measure of the harm to Adobe caused by PDIC’s alleged breach.

Adobe has adduced evidence from which a reasonable factfinder may find that PDIC’s breach of the covenant not sue was obvious or in bad faith. The Court reaches this conclusion despite the fact that, in earlier declining to award fees or sanctions under 35 U.S.C. § 285, Rule 11, 28 U.S.C. § 1927, and its inherent authority, the Court found “that PDIC was not pursuing these actions in bad faith or for an improper purpose.” *Princeton Digital*, 2016 WL 1533697, at *17-18. The standards for awarding sanctions differ from those governing whether a party “may seek damages for having to defend against that suit,” with the latter being less stringent.

Riveredge, 774 F. Supp. at 900.³

Nor is the Court persuaded that Adobe cannot disaggregate its defense fees from its affirmative fees. Adobe has produced evidence breaking down its attorney fees and costs by date, attorney, hourly rate, and hours worked. (See D.I. 200 Exs. 10-13, 20) Adobe’s damages expert, Dr. Becker, indicated that this evidence is sufficiently detailed such that he could separate the defense fees and affirmative fees if so required. (See D.I. 200 Ex. 9 at 124-25) By separate order, the Court will direct Adobe to serve a supplemental expert report relating to the revised amount of damages Adobe is seeking in light of today’s rulings.

Accordingly, the Court will grant in part and deny in part PDIC’s motion for summary judgment on attorney fees as damages.

b. Restitution

Adobe alternatively seeks restitution, amounting to the \$200,000 license fee Adobe paid to PDIC. PDIC contends that Adobe cannot claim restitution damages here because there has not been a “total” breach of the contract.

New Jersey has adopted the Restatement (Second) of Contracts’ approach to restitution,

³Even if this were not the case, the Court would not view itself as bound by “law of the case doctrine” and compelled to grant summary judgment to PDIC. “Reconsideration of a previously decided issue may . . . be appropriate in certain circumstances, including when the record contains new evidence.” *Hamilton v. Leavy*, 322 F.3d 776, 787 (3d Cir. 2003). The sanctions decision was made relatively early in the case, before significant discovery on the breach of contract issue had taken place. In connection with the sanctions decision, the Court “accepted Mr. Meagher’s representations as true” on certain points, *Princeton Digital*, 2016 WL 1533697, at *14-15, something a jury would not be obligated to do. Further, the Court noted in connection with its sanctions decision that Adobe had not presented “any evidence – expert or otherwise – to challenge the merits of [PDIC’s] infringement theory,” *id.* at *15, but Adobe has now done so (see, e.g., D.I. 197 Ex. 8). See also generally *Speeney v. Rutgers*, 369 Fed. App’x 357, 360-61 (3d Cir. 2010) (“[L]aw of the case doctrine only precludes relitigation of issues that the parties had a full and fair opportunity to litigate.”).

see *Kutzin v. Pirnie*, 591 A.2d 932, 941 (N.J. 1991), which entitles the non-breaching party to damages “for any benefit that he has conferred on the other party by way of part performance or reliance,” Restatement (Second) of Contracts § 373(a). The Restatement (Second) also provides that “restitution is available only if the breach gives rise to a claim for damages for total breach and not merely to a claim for damages for partial breach.” *Id.* at cmt. a; see also *Hansen Bancorp Inc. v. United States*, 367 F.3d 1297, 1309 (Fed. Cir. 2004). A total, or material, breach “is one that ‘goes to the essence of the contract,’” *Miller v. Butler*, 2014 WL 585409, at *6 (D.N.J. Feb. 14, 2014), and whether a breach is material is a question of fact for the jury, *Magnet Res.*, 723 A.2d at 982. “When one party commits a material breach of contract, the other party has a choice between two inconsistent rights – it can either elect to allege a total breach, terminate the contract and bring an action or, instead, elect to keep the contract in force, declare the default only a partial breach, and recover those damages caused by that partial breach.” 13 Williston on Contracts § 39; see also *Dover Shopping Ctr., Inc. v. Cushman’s Sons, Inc.*, 164 A.2d 785, 790 (N.J. Super. Ct. App. Div. 1960).

Here, Adobe fully performed its obligation under the contract by paying \$200,000 to PDIC. (See D.I. 196 at ¶ 4.1) PDIC’s obligations under the contract included a covenant not to sue (see *id.* at ¶ 3), and the Court has already explained above that the record permits a reasonable factfinder to find that PDIC breached that covenant by suing Defendants. The record further contains sufficient evidence from which a reasonable jury could find that this breach was material – for instance, multiple sections of the contract detail the license and covenant not to sue (see *id.* at ¶¶ 2-3), and Adobe evidently viewed the covenant as a necessary provision (see D.I. 200 Ex. 15 at 1). Even so, even if the record permits a finding of material breach by PDIC, a

reasonable factfinder could only find that Adobe, nonetheless, elected to keep the contract in force and attempt to recover damages for PDIC's partial breach. Adobe's complaint alleges that "Adobe has a valid and enforceable license to the '056 Patent." (D.I. 47 at ¶ 37) Furthermore, Adobe's requested relief includes "[a]n order enjoining [PDIC] from suing or maintaining suit against other Adobe customers and end users based on their use of Adobe licensed products and services, alone or in combination with other technology." (*Id.* at 10) That is, Adobe is asking the Court to enforce the contract, and not to treat Adobe's actions as a total breach. It follows, pursuant to New Jersey law and the Restatement, that restitution is not available to Adobe.

Accordingly, the Court will grant PDIC's motion for summary judgment that restitution is not available as a remedy.

III. CONCLUSION

For the foregoing reasons, the Court will deny PDIC's motion to exclude certain expert opinions and grant in part and deny in part PDIC's motion for summary judgment on Adobe's breach of contract claim. An appropriate Order follows.