No. 16-2741
(Consolidated with Nos. 16-2742, 16-2743, and 16-2744)

In the
United States Court of Appeals
for the Federal Circuit

PARALLEL NETWORKS, LLC,

Appellant,

v.

RELOADED GAMES, INC.,
KOG GAMES, INC.,
SG INTERACTIVE, INC., and
BLIZZARD ENTERTAINMENT, INC.,

Appellees.

APPEAL FROM THE UNITED STATES DISTRICT COURT FOR THE DISTRICT OF DELAWARE
CASE NO. 13-CV-00828 AND RELATED CASES NOS. 13-CV-00178, 13-CV-00826, 13-CV-00827
DISTRICT JUDGE RICHARD G. ANDREWS

NONCONFIDENTIAL BRIEF OF APPELLANT

Gregory N. Stillman, Esq.
HUNTON & WILLIAMS, LLP
500 East Main Street, Suite 1000
Norfolk, VA 23150
Telephone: (757) 640-5300
gstillman@hunton.com

Thomas R. Waskom, Esq.
HUNTON & WILLIAMS, LLP
951 East Byrd Street
Richmond, VA 23219
Telephone: (804) 788-8408
twaskom@hunton.com

Dated: December 13, 2016

FORM 9. Certificate of Interest

Form 9
Rev. 03/16

UNITED STATES COURT OF APPEALS FOR THE FEDERAL CIRCUIT

Parallel Networks, LLC **v.** SG Interactive, Inc., Blizzard Entertainment, Inc

Case No. 16-2741, 2742, 2743, 2744

CERTIFICATE OF INTEREST

Counsel for the:
☐ (petitioner) ☒ (appellant) ☐ (respondent) ☐ (appellee) ☐ (amicus) ☐ (name of party)

Parallel Networks, LLC

certifies the following (use "None" if applicable; use extra sheets if necessary):

1. Full Name of Party Represented by me	2. Name of Real Party in interest (Please only include any real party in interest NOT identified in Question 3) represented by me is:	3. Parent corporations and publicly held companies that own 10 % or more of stock in the party
Parallel Networks, LLC	n/a	None

4. The names of all law firms and the partners or associates that appeared for the party or amicus now represented by me in the trial court or agency or are expected to appear in this court (**and who have not or will not enter an appearance in this case**) are:

Adam W. Poff, James M. Lennon, Monte T. Squire, and Gregory J. Brodzik from Young Conaway Stargatt & Taylor, LLP; and Brian A. Carpenter, Eric W. Buether, Christopher M. Joe, and Michael D. Ricketts of Buether Joe & Carpenter, LLC

Dec 13, 2016
Date

/s/ Thomas R. Waskom
Signature of counsel

Please Note: All questions must be answered

Thomas R. Waskom
Printed name of counsel

cc:

Reset Fields

TABLE OF CONTENTS

TABLE OF AUTHORITIES iv

STATEMENT OF RELATED CASES vii

JURISDICTIONAL STATEMENT 1

STATEMENT OF ISSUES 2

STATEMENT OF THE CASE 3

I. The Patent-in-Suit 4

II. The Accused Instrumentalities 9

A. The Blizzard System 9

B. The Reloaded System 12

III. The Proceedings Below 15

A. Claim Construction 15

B. Summary Judgment 18

SUMMARY OF ARGUMENT 20

ARGUMENT 21

I. Standard of Review 22

II. The District Court's Claim Construction Is Wrong. 23

A. The Reallocation Limitation Should Be Given Its Ordinary Meaning. 23

B. The District Court's Construction Wrongly Adds the "Portions" / "Some but Not All" Concept to the Reallocation Limitation. 26

1. The District Court's "Portions"/"Some but Not All" Construction Is Wrong Because It Imports Limitations from the Specification. 27

2. The District Court's "Portions"/"Some but Not All" Construction Is Wrong Because It Is Contrary to the Teaching of the '145 Patent. 31

C. The District Court's "Each Peer" Construction Is Wrong Because It Is Contrary to the Teaching of the '145 Patent. 35

III. The District Court Erred in Finding that the Blizzard and Reloaded Systems Do Not Meet the Reallocation Limitation. 36

CONCLUSION 42

STATEMENT REGARDING ORAL ARGUMENT.....42

ADDENDUM

CERTIFICATE OF FILING AND SERVICE

CERTIFICATE OF COMPLIANCE

CONFIDENTIAL MATERIAL OMITTED

Pursuant to Federal Circuit Rule 28(d)(2)(B), Parallel Networks, LLC states that it has redacted certain information on pages 9-14 of the non-confidential, publicly-filed version of this brief. The redacted material was designated as confidential by the Appellees in the district court proceedings on the basis that it is proprietary business information and trade secrets. Material redacted in non-confidential brief is bracketed in confidential brief.

TABLE OF AUTHORITIES

Cases **Page(s)**

Absolute Software, Inc. v. Stealth Signal, Inc.,
659 F.3d 1121 (Fed. Cir. 2011) 21,22

Al Sharif v. U.S. Citizenship & Immigration Servs.,
734 F.3d 207 (3d Cir. 2013) (en banc) 22

Amstar Corp. v. Envirotech Corp.,
730 F.2d 1476 (Fed. Cir. 1984) 40

In re Baxter,
656 F.2d 679 (C.C.P.A. 1981) 40

Boehringer Ingelheim Vetmedica, Inc. v. Schering-Plough Corp.,
320 F.3d 1339 (Fed. Cir. 2003) 40

Classen Immunotherapies, Inc. v. Elan Pharms., Inc.,
786 F.3d 892 (Fed. Cir. 2015) 22

Cook Biotech, Inc. v. Acell, Inc.,
460 F.3d 1365 (Fed. Cir. 2006) 29

David Netzer Consulting Eng'r LLC v. Shell Oil Co.,
824 F.3d 989 (Fed. Cir. 2016) 21,40

Dippin' Dots v. Mosey,
476 1337, 1343 (Fed. Cir. 2007) 40

Dow Chem. Co. v. Sumitomo Chem. Co.,
257 F.3d 1364 (Fed. Cir. 2001) 40

Every Penny Counts, Inc. v. Am. Express Co.,
563 F.3d 1378 (Fed. Cir. 2009) 25

Finjan, Inc. v. Secure Computing Corp.,
626 F.3d 1197 (Fed. Cir. 2010) 26, 36

Genentech, Inc. v. Chiron Corp.,
112 F.3d 495 (Fed. Cir. 1997) 40

Kaneka Corp. v. Xiamen Kingdomway Group Co.,
790 F.3d 1298 (Fed. Cir. 2014) 40

O2 Micro Int'l Ltd. v. Beyond Innovation Tech. Co., Ltd.,
521 F.3d 1351 (Fed. Cir. 2008) 24

Phillips v. AWH Corp.,
415 F.3d 1303 (Fed. Cir. 2005) (en banc) 20, 29, 31, 35

Ruehl v. Viacom, Inc.,
500 F.3d 375 (3d Cir. 2007) 23

SciMed Life Sys. v. Advanced Cardiovascular Sys.,
242 F.3d 1337 (Fed. Cir. 2001) 29

Scotts v. Harris,
550 U.S. 372 (2007) 11

Summit 6, LLC v. Samsung Elecs. Co.,
802 F.3d 1283 (Fed. Cir. 2015) 26

Teva Pharms. USA, Inc. v. Sandoz, Inc.,
135 S. Ct. 831 (2015) 22

Texas Instruments, Inc. v. U.S. Int'l Trade Comm'n,
805 F.2d 1558 (Fed. Cir. 1986) 29

Thorner v. Sony Computer Entm't Am. LLC,
669 F.3d 1362 (Fed. Cir. 2011) 24

U.S. Ethernet Innovations, LLC v. Acer, Inc.,
646 Fed. Appx. 929 (Fed. Cir. Apr. 25, 2016) 30, 31

Vitronics Corp. v. Conceptronic, Inc.,
90 F.3d 1576 (Fed. Cir. 1996) 24

Young v. Martin,
801 F.3d 172 (3d Cir. 2015) 23

Statutes

28 U.S.C. § 1295(a)(1) 1

28 U.S.C. § 1331 1

28 U.S.C. § 1338(a) 1

28 U.S.C. § 2107(a) 1

Books

Jacques Derrida, *Structure, Sign and Play in the Discourse of the Human Sciences*, in WRITING AND DIFFERENCE (Alan Bass trans., 1980). 25

STATEMENT OF RELATED CASES

Pursuant to Federal Circuit Rule 47.5, Appellant Parallel Networks, LLC ("Parallel Networks") states as follows:

(a) Defendant Reloaded Games, Inc. previously appealed to this Court the decision of the Patent Trial and Appeal Board, Case No. IPR2014-00136, on *inter partes* review of U.S. Patent No. 7,188,145, the patent-in-suit. Parallel Networks was the Appellee in that action. The Appeal was Case No. 15-1915. On May 4, 2016, the Court entered a summary affirmance in favor of Parallel Networks pursuant to Federal Circuit Rule 36. The panel was Circuit Judges Moore, O'Malley and Hughes. The decision is not reported in the Federal Reporter but is in the Federal Appendix at 639 Fed. Appx. 640.

(b) Other than the cases consolidated for purposes of this Appeal, there are no other cases known to counsel to be pending in this or any other court that will directly affect or be directly affected by this Court's decision in the Appeal.

JURISDICTIONAL STATEMENT

The district court had jurisdiction under 28 U.S.C. §§ 1331 and 1338(a) and entered a final judgment on September 21, 2016. This Appeal, noticed on September 29, 2016, is timely. 28 U.S.C. § 2107(a); Fed. R. App. P. 4. This Court has jurisdiction under 28 U.S.C. § 1295(a)(1).

STATEMENT OF ISSUES

1. In its claim construction, the district court imported limitations from the patent specification and construed terms in a manner contrary to what the patent teaches. Did the district court misconstrue the claims?

2. The defendants' accused instrumentalities met every limitation in the claims at issue—their methods included every step in the patented method claims, and their systems were operable to perform every step in the patented system claims. The district court held that because the accused instrumentalities also performed additional steps, they did not infringe the patent. Did the district court err in finding non-infringement as a matter of law?

STATEMENT OF THE CASE

The technology at issue in these consolidated appeals is complex, but the issue before the Court is simple. Appellant Parallel Networks is the assignee of Patent 7,188,145 (the "'145 Patent"), which teaches a method and system for dynamic distributed data caching—that is, a way that groups of computers can work together to access information on the Internet more efficiently than if each worked on its own. The method and system provide a new and unique way for groups of computers to work collectively in a manner that allows each individual computer (called a "peer") to accomplish its own goal. Simply put, the invention allows the defendants in this case to shift the responsibility for distribution of their content away from their own servers by enlisting their customers' computers to share content with one another. Thus, the teachings of the '145 Patent increase the efficiency, and reduce the cost, of distributing the Defendants' content to their customers.

Appellees Blizzard Entertainment, Inc. ("Blizzard") and Reloaded Games, Inc., KOG Games, Inc., and SG Interactive, Inc. (collectively, "Reloaded," and collectively with Blizzard, "Defendants") develop and sell video games. The Defendants' customers download the games via systems that use specialized software to create peer-to-peer communities called "Swarms." A few customers download pieces of the game from a Defendant's origin server,

and then the customers' computers in the Swarm—i.e., the "peers"—obtain "pieces" of the game from one another, so that in some cases each computer in the swarm is uploading (distributing content) and downloading (receiving content) pieces of the game at the same time.

The district court held that because the goal of the Blizzard and Reloaded systems is for every customer to download 100% of a video game, they do not infringe the '145 Patent. That is the narrow question before this Court: whether the "reallocation" step of the '145 Patent excludes any system in which a single peer can download 100% of the content in the Swarm. As explained below, the district court's decision was wrong as a matter of law, as to both its claim construction and its conclusion that the Defendants' game-download systems do not infringe the '145 Patent.

I. The Patent-in-Suit

The purpose of the '145 Patent is to increase efficiency in downloading content from the Internet. When the '145 Patent was filed 15 years ago, Internet users were increasingly frustrated by network bottlenecks causing slow download speeds. When hundreds or thousands of users wanted to download information from a single source (e.g., a website), that source's origin servers quickly became overloaded, resulting in slow response times. Appx37, 1:14-19. One solution was to employ cache servers—secondary

servers to temporarily store some of the commonly-requested content. Appx37, 1:20-23. But that solution, too, had its limits: cache servers would also become overloaded, because the number of available cache servers was simply insufficient to meet the incoming requests of users in a timely manner. Appx37, 1:28-32.

The '145 Patent provides innovative technical solutions to these problems. Appx38, 4:3-34. The '145 Patent establishes a peer-to-peer community, also called a Swarm, comprised of clients. Appx37, 1:43-45. Client peers are allowed to join the Swarm, and the newly admitted client peers establish connections with other peers in the Swarm so that they can obtain content from one another. Appx37, 2:37-42. As peers obtain content from other peers in the Swarm, the Swarm is updated on which peer has what content. Appx38, 3:18-21. That allows each peer to know which peer to contact next to get the content it wants.

Figure 6 is a block diagram illustrating a dynamic caching system comprising one embodiment of the invention:

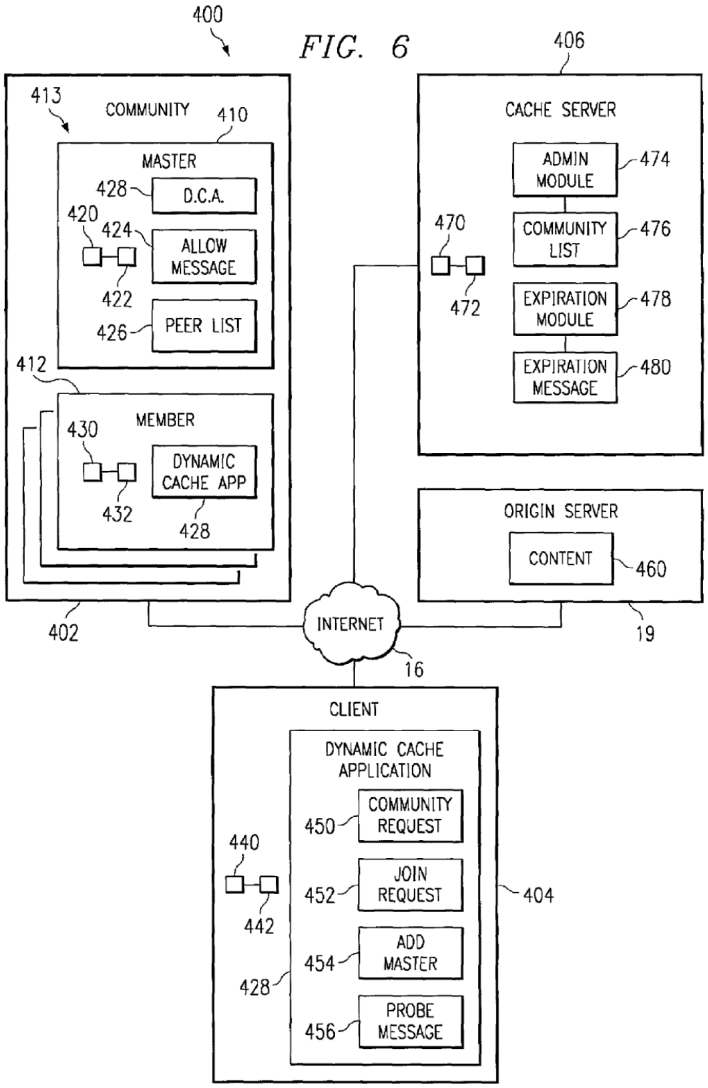


FIG. 6

The system 400 as a whole comprises a cache community 402, a prospective new client 404, a cache server 406, and the origin server 19. Appx45, 17:52-55. The community 402 is the Swarm, and it comprises one or more peers that are cooperating to act as a distributed caching system. Appx4, 18:28-29.

The prospective new client 404 sends a join request 452 to the community 402. Appx46, 19:21-23. If the community 402 permits client 404 to join, client 404 will learn which peers 412 have the information client 404

needs. Appx47, 21:1-13. The content sought by the community "may comprise audio data, video data, text data, animations, applications, applets and other suitable content." Appx46, 19:29-31.

Client 404 can then download pieces of the information it needs from the other peers 412. As new peers join and existing peers leave, and as peers 412 acquire new information from one another or the server, the community 402 updates the shared information regarding which peers 412 have which pieces of information.

The '145 Patent includes two independent claims that have been asserted—1, for a method, and 15, for a system. Those claims provide:

> What is claimed is:
> **1.** A method for dynamic distributed data caching comprising:
> providing a cache community on a first side of a point of presence, the cache community comprising at least one peer, the cache community being associated with content obtained from a second side of the point of presence, the content being cached by the at least one peer;
> allowing a client to join the cache community;
> updating a peer list associated with the cache community to include the client, the peer list indicating the peers in the cache community;
> associating the content with the client based on joinder of the client;
> ***re-allocating the cache storage of the content among the peers in the cache community*** in response to allowing the client to join the community.
>
> **15.** A system for dynamic distributed data caching comprising:
> logic encoded on storage and operable to:
> provide a cache community on a first side of a point of presence, the cache community comprising at least one peer, the cache

> community being associated with content obtained from a second side of the point of presence, the content being cached by the at least one peer;
> allow a client to join the cache community;
> update a peer list associated with the cache community to include the client, the peer list indicating the peers in the cache community;
> associate the content with the client based on joinder of the client;
> ***re-allocate the cache storage of the content among the peers in the cache community*** in response to allowing the client to join the community.

Appx50-51, 28:49-65, 29:53-30:3 (emphasis added). The bolded language is the limitation that proved critical, and dispositive, in the district court.

The '145 Patent was in prosecution for over six years at the United States Patent and Trademark Office during its initial examination. The Patent Examiner determined that the '145 was novel, non-obvious and contained patentable subject matter. Appx3088-3089.

The '145 Patent was also the subject of two requests filed by Reloaded for *inter partes* review of 36 claims. *See* IPR2014-00950 and IPR2014-00136. On May 14, 2015, the Patent Trial and Appeals Board (the "PTAB") issued its final order that none of the claims on which *inter partes* review was instituted were unpatentable. Reloaded then appealed the PTAB's final order to this Court, which affirmed the PTAB's determination of validity.

II. The Accused Instrumentalities

Both Blizzard and Reloaded use systems intended to allow large numbers of their customers to simultaneously download video games without overloading the Defendants' origin servers.[1] At a high level, the systems operate in the same way. The district court effectively merged the infringement analysis for the two systems. Thus, the district court's analysis regarding Blizzard was wrong for the same reason its analysis regarding Reloaded was wrong. For both, the district court focused on a single issue: whether every customer utilizing the Blizzard System or Reloaded System was, ultimately, trying to download 100% of the game content. That greatly simplifies what facts regarding the two systems are relevant to this appeal.

A. The Blizzard System

The Blizzard System comprises several components: [redacted] Appx7119-7120, ¶ 65. Each Blizzard customer uses a game installer, which in turn uses the Blizzard Downloader to access

[1] The Blizzard system is referred to in this brief as the Blizzard System. In the proceedings below, the district court and the parties referred to the system used by Reloaded as the Pando System, because the technology was developed by Pando Networks, Inc., and licensed to Reloaded. In this brief, Parallel Networks will use the term "Reloaded System" for clarity and consistency.

the Blizzard System. Appx10-11. The customer then uses the Blizzard System to download the video game package (the "Game Package"). Appx11.

Blizzard places content (e.g., video games or game updates) for distribution on its CDN servers. Appx11. Each Game Package is split into components called "Pieces." [ ] Appx7120, ¶ 69.

[ ] Appx7120-7121, ¶ 71. [ ] Appx7120-7121, ¶¶ 71-72.

Each Blizzard Client sends an "announce message" to the tracker server upon first joining the Swarm. Appx11. When a new Blizzard Client joins the Swarm and sends an announce message, the tracker server responds with a list of a subset of all Blizzard Clients currently in the Swarm (the "Selected List"). Appx11-12. [ ] Appx7122, ¶ 80.

Upon receiving a Selected List, a Blizzard Client attempts to form a network connection with some or all of the other peers on that list. Appx11-12. If the connection is formed, the Blizzard Clients exchange information

with one another about which Pieces each has at the time, and which Pieces each is seeking. Appx12. After receiving information about the location of Pieces in the Swarm, a Blizzard Client can obtain that Piece from another peer in the Swarm. *Id*. When the download of that Piece is complete, the Blizzard Client sends a message to its peers, telling the others that it now has that new Piece and will share it upon request. *Id*.

Each Blizzard Client decides which Piece to download next based on a "Rarest First" algorithm.[2] Appx12. [ ] Appx7125-7126, ¶ 90.

The Swarm is an ecosystem that evolves over time as Blizzard Clients enter and exit. Appx7440, ¶ 30. For each individual Blizzard Client, the process of obtaining all necessary Pieces for a Game Package takes a significant amount of time: for instance, the demonstration version of

[2] At the summary judgment stage, [ ] *See* Appx6405-6407. For purposes of this appeal, the Court must view the evidence in the light favorable to Parallel Networks. *Scotts v. Harris*, 550 U.S. 372, 380 (2007).

Blizzard's StarCraft II: Wings of Liberty is 1.6 gigabytes, [redacted] Appx7446, ¶¶ 47, 50.

The download process is intended to continue until the Blizzard Client accumulates every Piece in the Game Package. Appx7126, ¶ 92. In practice, that does not always happen. In fact, [redacted] Appx7446, ¶ 50.

B. The Reloaded System

The Reloaded Defendants develop and sell video games, and use the Reloaded System to facilitate customers' download of the games. Appx7-8. The Reloaded System has a great deal in common with the Blizzard System—in structure, in function, in purpose, and in the way it infringes the '145 Patent.

The Reloaded System distributes Game Packages via peer-to-peer networks using hardware and software that includes tracker servers, CDN servers, and, on the customer side, the Reloaded Game Downloader. Appx7-8. Reloaded's Game Packages range in size from hundreds of megabytes to several gigabytes. Appx8. The Reloaded System breaks the Game Packages down into Pieces. *Id*. Reloaded customers download the Game Downloader,

which itself installs the Reloaded Client software on the customer's computer. *Id*.

Reloaded places its Game Packages, broken down into the Pieces, on CDN servers. Appx3801, ¶ 70. A customer executes the Reloaded Client, which contacts the Reloaded tracker server to learn how to download the Game Package. Appx8. The tracker server maintains a list of all Reloaded Clients that have requested, to that point, to download the Game Package. *Id*. The collection of all clients participating in the transfer of the Pieces to one another is called the Swarm (just like in the Blizzard System). *Id*.

The tracker server responds to the Reloaded Client seeking to join the Swarm with a list of certain clients already in the Swarm (the "Selected List"). Appx8. [redacted] Appx3802-3803, ¶ 75. For any Remote Clients that accept that invitation (an "Established Connection"), the Reloaded Client forms a network connection and begins exchanging information. Appx9.

The Reloaded Client and the Established Connections then tell one another which Pieces they have, and each selects a Piece from the other's list and begins to download it. Appx9. As with Blizzard, the Reloaded System

uses a Rarest First algorithm to decide which Piece to download next.[3] Appx9-10. The Reloaded Client determines which Piece is rarest (and thus the best target for download) by regularly collecting data from each of its Established Connections. Appx10. [redacted] Appx3805-3806, ¶ 82.

[redacted] Appx3806, ¶ 84. [redacted] *Id*.

Downloading every Piece takes a substantial amount of time—for the game APB Reloaded, for instance, [redacted] Appx4696-4697, ¶ 10 & n.1. As with the Blizzard System, the goal of the Reloaded System is to deliver 100% of the Game Package to each Reloaded customer. But as with Blizzard, the Reloaded System does not

[3] [redacted] Appx3807, ¶ 88. [redacted] *Id*.

always meet its goal—meaning that in some cases, a Reloaded Client exits the Swarm without having downloaded 100% of a Game Package. Appx4696, ¶ 9.

III. The Proceedings Below

Parallel Networks filed suit in the District of Delaware against KOG for infringement of the '145 Patent on February 1, 2013. On May 9, 2013, Parallel Networks filed suit in the same district court against Reloaded, SGI and Blizzard for infringement of the '145 Patent.

On February 6, 2014, the district court stayed the proceedings pending decision on Reloaded's petition for *inter partes* review. The PTO issued its decision on May 14, 2015, affirming the patentability of all of the claims in the '145 Patent on which it had instituted review.[4] Reloaded appealed to this Court, which summarily affirmed the PTO's decision. On August 7, 2014, before the PTO issued a ruling, the district court lifted its stay.

A. Claim Construction

After limited discovery, the parties jointly briefed the claim constructions in dispute. The parties agreed on several constructions, but left two terms for the district court to construe: "cache community" and "re-

[4] The PTO simultaneously found that the claims of U.S. Patent No. 7,730,262 were unpatentable.

allocating/re-allocate the cache storage of the content among the peers in the cache community" (the "Reallocation Limitation").

With respect to the latter, Parallel Networks argued that there was no need for construction. Appx2977. Parallel Networks proffered an expert declaration from Dr. Peter Reiher, professor of computer science at the University of California, Los Angeles, and a former chief designer at NASA's Jet Propulsion Laboratory. Appx3105, ¶ 4. Dr. Reiher explained that a person of ordinary skill in the art would understand the term as used in the '145 Patent with sufficient clarity. Appx3107-3109, ¶¶ 11-17. In the alternative, Parallel Networks proposed construing "re-allocate" as "non-exclusively re-assign." Appx2981.

Blizzard asked the district court to construe the Reallocation Limitation as "reassigning/reassign the portions of the content to be stored by each peer's cache storage." Appx2987. Blizzard argued that this construction was appropriate because, in Blizzard's view, no two peers could cache the same information at the same time. *Id*.

Reloaded, on the other hand, stated that while it "agree[d] with the analysis of the '145 Patent proffered by" Blizzard, the only word in the Reallocation Limitation requiring construction was "among." Appx2987-2988. Reloaded offered its own proposal for resolving the meaning of

"among": "re-allocate/ing the cache storage of the content with a share to each of the peers in the community in response to allowing the client to join the community." Appx2988.

The district court held a claim construction hearing on August 17, 2015. At the hearing, the district court focused on whether a peer, upon joining a cache community, could download for itself *all* of the content the community held:

> I think the primary dispute, which the parties have argued, is whether each peer can cache 100 percent of the content.... I think each peer caches a portion of data or the cache content. And in the section of the summary of the invention described in the claims, each peer is associated with a content portion. I think that the summary makes clear that each peer receives some but not all of the content.

Appx3247, 115:4-14. Following the hearing, the district court issued an order adopting Blizzard's proposed construction of the Reallocation Limitation:

Term	Court's Construction
"cache community" '145 Patent Claims 1, 4, 15, 16	"one or more peers which operate as a group to cache data"
"reallocating/re-allocate the cache storage of the content among the peers in the cache community" '145 Patent Claims 1, 12, 14, 15, 26, 28	"re-allocating/re-allocate the portions of the content to be stored by each peer's cache storage"

Appx22. The district court did not issue an opinion explaining its reasoning regarding claim construction.

B. Summary Judgment

After claim construction, the parties conducted some very targeted discovery, and then filed cross-motions for summary judgment on the Reallocation Limitation. Parallel Networks and Reloaded filed a joint brief on February 12, 2016; Parallel Networks and Blizzard filed a joint brief on February 23, 2016. The district court held a joint hearing on both sets of motions on May 3, 2016.

On August 31, 2016, the district court denied Parallel Networks' motions for partial summary judgment and granted all Defendants' motions for summary judgment of non-infringement. The district court applied a different construction of the Reallocation Limitation than what it had announced in its *Markman* order. In that order, the district court had construed the Reallocation Limitation as "re-allocating/re-allocate the portions of the content to be stored by each peer's cache storage." Appx22. But even the district court, it turns out, did not know what it meant by "portions." So in its summary judgment ruling, the district court construed its own construction, stating that "[t]he use of 'portions' in the construction is important," and trying to unpack what, exactly, that term meant. Appx16. It

settled on the idea that "portions" means that "'each peer receives some but not all of the content.'" *Id*.

The district court's construction of its own construction—specifically, its "some but not all" formulation—served as the basis for its summary judgment ruling. The district court held that because "each [Reloaded] client will download every piece of the game package," and "every [Blizzard] client downloads 100% of the game package," the Blizzard and Reloaded Systems do not infringe the '145 Patent. Appx16, 19. According to the district court, the Blizzard and Reloaded Systems do not meet the Reallocation Limitation because "[t]here is no divided responsibility, as no user ultimately stores less than 100% of the content." Appx19.

Whether every user "ultimately stores … 100% of the content" was at best a genuine issue of material fact. Arguably, the only evidence in the record on that point demonstrates the opposite—that *not* every user ultimately stores 100% of the content. *See* Appx7425-7426, ¶ 9; Appx4696-4697, ¶ 10 & n.1. Nevertheless, the district court seems to have resolved this question of fact in the Defendants' favor.

According to the district court, the fact that every Blizzard or Reloaded customer enters the Swarm with the intention of downloading all necessary pieces of a Game Package means that "the portions of the content to be stored

are never reallocated…. In fact, every user will use the same amount of space to store the same data." Appx16. As explained above, that misapprehends the point of the '145 Patent, and what it teaches. The method and system are not, ultimately, intended to address a storage issue, but rather a download issue—how to maximize efficiency in accessing information.

This appeal followed.

SUMMARY OF ARGUMENT

The district court granted the Defendants summary judgment because, in the district court's view, their accused instrumentalities do not meet the limitation in Claims 1 and 15 requiring "re-allocat[ion of] the cache storage of the content among the peers in the cache community." The parties had several other disputes, but the district court's decision turned on the construction and application of the Reallocation Limitation. There are two ways to view what the district court did. Both are reversible error.

In the first view, the district court misconstrued the Reallocation Limitation. The district court arrived at a claim construction that is fatally flawed, because it imports a limitation from the specification. That is contrary to longstanding, black-letter law in this Court. *Phillips v. AWH Corp.*, 415 F.3d 1303, 1323 (Fed. Cir. 2005) (en banc) (district courts must "avoid … reading

limitations from the specification into the claim"). Moreover, the district court's construction is directly contrary to the teaching of the '145 Patent.

In the second view, the district court reached a permissible (though not optimal) claim construction, but then concluded in error that the accused instrumentalities did not infringe because even though they met every limitation in the claims, the fact that they also performed additional steps made them non-infringing. Again, that is contrary to black-letter law. *E.g.*, *David Netzer Consulting Eng'r LLC v. Shell Oil Co.*, 824 F.3d 989, 998 (Fed. Cir. 2016) (a "claim with the word 'comprising' appearing at the beginning generally allows for additional, unclaimed steps in the accused process"). The district court compounded its error by not only "resolving" a genuine issue of material fact, but getting the answer wrong.

If there is any difficulty in this case, it is only because both the district court's claim construction and its grounds for summary judgment are unclear. But on whichever ground the district court relied, it reached a conclusion at odds with the settled law of this Court.

ARGUMENT

To determine infringement, a court first construes the scope and meaning of the asserted patent claims, and then compares the construed claims to the accused product or process. *Absolute Software, Inc. v. Stealth*

Signal, Inc., 659 F.3d 1121, 1129 (Fed. Cir. 2011). The district court erred at both steps.

I. Standard of Review

For purposes of claim construction, "when the district court reviews only evidence intrinsic to the patent (the patent claims and specifications, along with the patent's prosecution history), the judge's determination will amount solely to a determination of law, and the Court of Appeals will review that construction *de novo*." *Teva Pharms. USA, Inc. v. Sandoz, Inc.*, 135 S. Ct. 831, 841 (2015). Where the district court's claim construction is based on factual findings regarding extrinsic evidence, review of "the district court's ultimate construction of the claim [is] *de novo*," but the underlying factual findings are reviewed for clear error. *Id*. The district court neither issued an opinion regarding claim construction nor made any findings of fact on the record, so the claim construction issue is reviewed *de novo* in its entirety.

This Court reviews a district court's grant of summary judgment according to the law of the regional circuit—here, the Third Circuit. *Classen Immunotherapies, Inc. v. Elan Pharms., Inc.*, 786 F.3d 892, 896 (Fed. Cir. 2015). In the Third Circuit, the court reviews the grant of summary judgment *de novo*. *See Al Sharif v. U.S. Citizenship & Immigration Servs.*, 734 F.3d 207, 210 n.2 (3d Cir. 2013) (en banc). This Court may affirm summary judgment "only

if 'drawing all reasonable inferences in favor of the nonmoving party, there is no genuine issue as to any material fact and the moving party is entitled to judgment as a matter of law.'" *Young v. Martin*, 801 F.3d 172, 177 (3d Cir. 2015) (quoting *Ruehl v. Viacom, Inc.*, 500 F.3d 375, 380 n.6 (3d Cir. 2007)).

II. The District Court's Claim Construction Is Wrong.

The Reallocation Limitation is: "re-allocating/re-allocate the cache storage of the content among the peers in the cache community." '145 Patent Claims 1 & 15. In construing just those few words, the district court made at least three errors. First, the district court wrongly rejected Parallel Networks' proposed construction: that the terms be given their ordinary meaning. Second, the district court improperly imported from the specification the idea that each peer can only download "portions"—or "some but not all"—of the content sought by the swarm. Third, the district court misconstrued the limitation to require reallocation to "each" peer, when the patent teaches otherwise.

A. The Reallocation Limitation Should Be Given Its Ordinary Meaning.

The proper construction here is to allow the jury to consider the Reallocation Limitation according to its terms' ordinary meaning. The district court was not "required to construe every limitation present in a patent's

asserted claims." *O2 Micro Int'l Ltd. v. Beyond Innovation Tech. Co., Ltd.*, 521 F.3d 1351, 1362 (Fed. Cir. 2008).

Claim terms are generally given their ordinary and customary meaning as understood by a person of ordinary skill in the art when read in the context of the specification and prosecution history. *Thorner v. Sony Computer Entm't Am. LLC*, 669 F.3d 1362, 1365 (Fed. Cir. 2011). There are "only two exceptions to this general rule: 1) when a patentee sets out a definition and acts as his own lexicographer, or 2) when the patentee disavows the full scope of a claim term either in the specification or during prosecution." *Id*. (citing *Vitronics Corp. v. Conceptronic, Inc.*, 90 F.3d 1576, 1580 (Fed. Cir. 1996)). Neither exception applies here.

After all of the briefing and argument was complete, the district court (rightly) found that no word could better capture or explain the term than "reallocate" itself: "I believe that, in fact, the actual word 're-allocate,' there's actually no dispute, so I'm not going to change that word." Appx3246-3247, 114:25-115:3. That is the correct outcome with respect to that term. As Dr. Reiher explained, a person of ordinary skill in the art would understand "reallocate" with sufficient clarity. Appx3107-3109, ¶¶ 11-17.

The district court should have stopped there, and construed nothing further. The problems with construing "reallocate" apply with equal or

greater force to the problems with construing the other terms in the Reallocation Limitation. The Defendants and the district court were rearranging words and inserting new concepts—not clarifying anything, but rather adding limitations where none existed before.

In fact, the district court's construction was so confusing that the court itself had to parse its own meaning at the summary judgment stage. As noted above, at summary judgment the district court did not use the construction of the Reallocation Limitation memorialized in the *Markman* order. Because its own claim construction was even more inscrutable than the claims themselves, the district court "'need[ed] to interpret [its own] interpretations.'" *Every Penny Counts, Inc. v. Am. Express Co.*, 563 F.3d 1378, 1383 (Fed. Cir. 2009) (quoting Jacques Derrida, *Structure, Sign and Play in the Discourse of the Human Sciences*, in WRITING AND DIFFERENCE 278 (Alan Bass trans., 1980)).

The district court extrapolated, from its own use of the word "portions," that each peer in a Swarm can download "some but not all" of the content in the community. Both the district court's construction and its meta-construction are wrong. The "some but not all" construction is particularly egregious: as a second derivative of the Reallocation Limitation, it is wholly divorced from the claims themselves. The district court's difficulty in

interpreting its own construction of the Reallocation Limitation is the best evidence that the construction was flawed. It is also strong evidence that no construction better explains the terms of the Reallocation Limitation than the terms themselves do.

The Defendants do not like what the plain language of the Reallocation Limitation means for their case. But that is not a reason to permit them to rewrite the claims in Parallel Networks' patent. "While the court must resolve actual disputes regarding the proper scope of a claim term, restating a settled argument does not create an actual dispute within the meaning of *O2 Micro*." *Summit 6, LLC v. Samsung Elecs. Co.*, 802 F.3d 1283, 1291 (Fed. Cir. 2015) (internal citations omitted) (citing *Finjan, Inc. v. Secure Computing Corp.*, 626 F.3d 1197, 1207 (Fed. Cir. 2010)). Such is the case here. Neither the parties nor the district court proposed a construction for the Reallocation Limitation that would be more helpful to the jury than the plain language from the claims.

B. The District Court's Construction Wrongly Adds the "Portions" / "Some but Not All" Concept to the Reallocation Limitation.

The most egregious problem with the district court's claim construction is that it introduces a new concept into the Reallocation

Limitation: that no single peer can ultimately download all of the content the Swarm seeks.

1. The District Court's "Portions"/"Some but Not All" Construction Is Wrong Because It Imports Limitations from the Specification.

The district court decided that the Reallocation Limitation requires that "each peer receives some but not all of the content." Appx16. That is wrong, because it improperly imports a limitation from the patent specification. The best evidence of the error is in the path the district court took to reach that result.

The district court stated that there was no dispute about the phrase "re-allocating/re-allocate," and no need to construe it. Appx3246-3247, 114:21-115:3. Nor does any other word or phrase in the Reallocation Limitation relate to how much content a single peer can download, or whether a single peer can download *all* of the content. So from where did the district court draw the "some but not all" limitation? Solely from the Summary of Invention section of specification.

The district court itself identified the Summary of Invention as the source of the "some but not all" limitation. At the *Markman* hearing, the district court asked counsel for Reloaded, "Do you think the summary of the invention … and particularly the two paragraphs, the bottom of column 1, do

they have any impact on this discussion we're having here" about the Reallocation Limitation. Appx3227, 95:3-9. The paragraphs at issue were the preferred embodiments for the method and system. They state:

> According to one embodiment of the present invention, a method for dynamic distributed data caching is provided. The method comprises providing a cache community comprising at least one peer. Each peer has an associated first content portion indicating content to be cached by the respective peer. The method further comprises allowing a client to join the cache community, updating a peer list associated with the cache community to include the client, the peer list indicating the peers in the cached community, and associating a respective second content portion with each peer based on the addition of the client. The second content portion is distinct from the first content portion.
>
> According to a further embodiment of the present invention, a system for dynamic distributed data caching is presented. The system comprises logic encoded on storage. The logic is operable to provide a cache community comprising at least one peer. Each peer has an associated first content portion indicating content to be cached by the respective peer and allow a client to join the cache community. The logic is further operable to update a peer list associated with the cache community to include the client. The peer list indicates the peers in the cache community. The logic is further operable to associate a respective second content portion with each peer based on the addition of the client. The second content portion is distinct from the first content portion.

Appx37, 1:42-66.

The district court latched onto the idea that for these particular embodiments, when a new peer joins the cache community, there is a reallocation of "content portions" among the peers, and for each peer, "[t]he second content portion is distinct from the first content portion." But that

concept is found nowhere in the claims themselves, let alone in the Reallocation Limitation. Instead, it is found only in the Summary of Invention.

This Court has strictly proscribed importing limitations from the specifications into the claims. *Phillips v. AWH Corp.*, 415 F.3d 1303, 1323 (Fed. Cir. 2005) (en banc); *see also Texas Instruments, Inc. v. U.S. Int'l Trade Comm'n*, 805 F.2d 1558, 1563 (Fed. Cir. 1986) ("This court has cautioned against limiting the claimed invention to preferred embodiments or specific examples in the specification."). Reading a "some but not all" limitation into the Reallocation Limitation would commit "one of the cardinal sins of patent law—reading a limitation from the written description into the claims." *SciMed Life Sys. v. Advanced Cardiovascular Sys.*, 242 F.3d 1337, 1340 (Fed. Cir. 2001).

There is nothing wrong with using the Summary of Invention to construe the claims themselves. *See*, *e.g.*, *Cook Biotech, Inc. v. Acell, Inc.*, 460 F.3d 1365, 1374 (Fed. Cir. 2006) (looking to summary of invention for guidance in determining patentee's "special definition" of a term). But here, the district court did not properly use the specification to construe the claims. Instead, it lifted concepts from the specification and dropped them into the claims. The district court was not "construing" any term in the claim when it asserted "that the 'primary dispute' between the parties was 'whether each

peer [could] cache 100 percent of the content.'" Appx16. The amount or percentage of content each peer can receive is a concept totally foreign to the claims themselves. The suggestion that "each peer receives some but not all of the content," *id*., comes from the Summary of Invention alone.

This Court's recent decision in *U.S. Ethernet Innovations, LLC v. Acer, Inc.*, 646 Fed. Appx. 929 (Fed. Cir. Apr. 25, 2016), is instructive. That case involved patents for transmit buffers, technology used by network adapters to hold data sent to a network by local computers. *Id*. at 930-31. The patent at issue in the appeal describes a hybrid buffer system providing the advantages of both a dedicated transmit buffer (which downloads one frame at a time from a computer and stores it until the entire frame is uploaded to the network) and a "first-in-first out" buffer (which allows for continuous downloading and uploading). *Id*. at 931. The claim at issue included the element "buffer memory for storing data of frames … for transmission." *Id*. at 933. Both the discussion of the nature of the invention and numerous embodiments described a buffer with the "capacity to store an entire Ethernet frame." *Id*.

The patent holder sued a competitor for infringement, and the competitor responded that its product did not infringe because its buffer did not have the capacity to store an entire frame. Notwithstanding the statements in the specification, both the district court and this Court rejected

the competitor's construction, because the claims themselves did "not say that the buffer must 'store all the data of frames,' or 'store *entire* frames of data,' or, tellingly, that the buffer even 'store frames of data.'" *Id*. at 934 (emphasis in original). The Court held that "'[i]t is a bedrock principle of patent law that the claims of a patent define the invention to which the patentee is entitled the right to exclude.'" *Id*. at 933 (quoting *Phillips*, 415 F.3d at 1312).

The exact same is true here. There is nothing in the claims themselves about "portions," or the amount of content each peer can download, or whether each peer can download all of the content the swarm as a whole seeks. That idea is found only in an embodiment in the specification. Just as it would have been improper in *U.S. Ethernet* to import from the specification a limitation requiring that the buffer be capable of storing a full frame, so too here would it be wrong to import from the specification any purported cap on the amount of content each peer can or must download.

2. The District Court's "Portions"/"Some but Not All" Construction Is Wrong Because It Is Contrary to the Teaching of the '145 Patent.

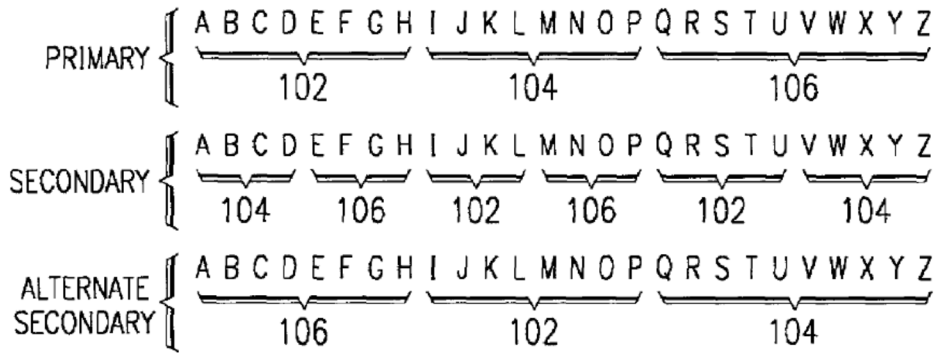
The district court's "some but not all" concept cannot be squared with the '145 Patent when read in its totality. The "some but not all" view of the claims rests on a misconception of reallocation as a zero-sum game: that if Peer A has Portion 1, and then Portion 1 is allocated to Peer B, that Portion 1

must be deleted from Peer A. If that were true, then no one peer could ever accumulate 100% of the content in the community. But reallocation under the '145 Patent does not require robbing Peter to pay Paul.

The '145 Patent teaches that "[r]edistribution may comprise actually *copying* cached content from one client … to another, *or* removing or expiring content no longer cached at a particular server." Appx44, 16:41-44 (emphasis added). And copying content from one peer to another—as opposed to cutting and pasting content from one peer to another, leaving the former peer without the content going forward—is exactly what the Blizzard and Reloaded Systems do.

Further, the '145 Patent's specification specifically contemplates that more than one peer will simultaneously cache the same information—contrary to the district court's view that one peer's storage of a file requires another peer to delete that content from its own storage. For instance, Figure 5 shows a snapshot in time of a possible allocation of cache storage:

FIG. 5



Here, Peer 102 has cached Pieces A through H. But the system has also allocated Pieces A through D to Peer 104, and Pieces E through H to Peer 106. The patent teaches that "[a]n overlapping secondary distribution," such as that shown in Figure 5, "may be used to allocation portions of the primary distribution to other clients for increased reliability." Appx44, 16:9-12. There is no restriction on the number of peers that can store the same information. Appx44, 16:26-28.

In the claim construction briefing, Blizzard actually pointed to Figure 5 as evidence that no two peers can cache the same content at the same time. Appx2986-2987. According to Blizzard, "[i]t is important to note that both the 'primary' and 'secondary' distributions are 'exclusive' distributions. In neither distribution does a single peer receive a 'cache share' that another peer receives." Appx2987. But that is exactly wrong. Blizzard misapprehended what Figure 5 represents. The primary and secondary distributions in Figure

5 occur simultaneously, not in sequence—as the plain language of the patent states. Appx44, 16:9-12. Figure 5 shows the exact opposite of what Blizzard told the district court. Yet it was on the basis of those arguments that the district court adopted Blizzard's proposed claim construction.

In its summary judgment opinion, the district court relegated this entire question to a footnote, stating that "[t]he parties dispute whether a reallocation inherently requires some removal of content, rather than duplication…. Although the parties contest this issue at length, I do not think it is material to the outcome here." Appx17, 13 n.15. The parties contested the issue at length because it was the foundation of the claim construction that Blizzard offered, and that the district court accepted.

The "some but not all" conception of the Reallocation Limitation is directly contrary to the teaching of the '145 Patent. It is based on arguments that fundamentally misunderstand what the patent says and what the invention does. It injects entirely foreign concepts into the Reallocation Limitation. The Court should vacate the claim construction order.

C. The District Court's "Each Peer" Construction Is Wrong Because It Is Contrary to the Teaching of the '145 Patent.

The Reallocation Limitation requires only reallocation "among the peers." It does not require reallocation to *every* peer, which is a potential reading of the construction adopted by the district court.

Before the district court, Reloaded argued that "among" should mean "with a share to each." Appx2972. It based that argument on dictionary definitions—and Parallel Networks was able to counter with its own dictionary definitions.[5] That is exactly why this Court has rejected a "dictionary first" approach to claim construction. *Phillips*, 415 F.3d at 1321.

Instead, the Court should start with the intrinsic evidence, which demonstrates that the construction proposed by the defendants and adopted by the district court is wrong. The '145 Patent teaches that "[t]he present invention … reallocates data to be cached by *particular* members of the distributed cache community based on the addition and subtraction of members to the distributed cache community." Appx38, 4:32-35 (emphasis added); *see also* Appx3108, ¶ 12. If "particular" members are reallocated data, then by implication and necessity some members of the cache community are

[5] Parallel Networks offered, as an alternative, definitions 3 and 4 from Random House Webster's Unabridged Dictionary, Second Edition (2001). Appx3117.

not allocated data in that embodiment. The Court should look first to the claims, specification and file history, all of which militate in favor of giving "among the peers" its ordinary meaning, rather than some unduly restrictive definition.

The district court erred in construing the Reallocation Limitation to require that each reallocation must affect "each peer." The district court should have allowed the terms of the Reallocation Limitation to have their ordinary meaning.

III. The District Court Erred in Finding that the Blizzard and Reloaded Systems Do Not Meet the Reallocation Limitation.

Even if the district court's ruling on claim construction was right, its ruling on summary judgment was wrong. That is true for two reasons. First, the district court's decision was based on supposition that is directly contrary to the record evidence. Second, the district court incorrectly found non-infringement on the basis that the Blizzard and Reloaded Systems performed additional steps after completing the infringing steps.

To prove infringement of a method claim, the plaintiff must establish that the defendant "practiced all steps of the claimed method." *Finjan,* 626 F.3d at 1206. The district court acknowledged that for both the Blizzard and Reloaded Systems, upon entering the Swarm, a peer downloads a Piece from

another peer, and then informs other peers in the swarm that it has cached that new portion. Appx9 ("Whenever a [Reloaded] client successfully downloaded a new piece, it sent a 'Have Message' to each of its established connections, thereby informing its established connections that it had a new piece of data to share."); Appx12 ("After successfully downloading a piece from another [Blizzard] client in the swarm, a [Blizzard] client sent a 'Have Message' to all of its established connections, informing them that it had a new piece available for download.").

The other peers utilize that information in their Rarest First algorithms, affecting which Piece each peer will seek to download next—i.e., *reallocating* which portions each peer will next store. Appx9-10, 12. The analysis can stop right there: that process meets the Reallocation Limitation.

But the district court did not stop right there—it went on to see what happens next in the Blizzard and Reloaded Systems. As those systems continue operating, each peer will try to download another piece, and another piece, and another, until, hopefully, it acquires every piece. That does not change the fact that infringement has already occurred.

According to the district court, peers' continued downloading of additional pieces of video games *does* alter whether infringement has already occurred. The district court said that "each [Reloaded] client will download

every piece of the game package, regardless of the order in which those pieces are downloaded," and "every [Blizzard] client downloads 100% of the game package." Appx16. That purported fact was a keystone in the district court's finding of non-infringement.

But on this point, the district court's opinion is unsupported by any evidence.[6] In fact, the district court's supposition is *contrary* to the evidentiary record. Not every Blizzard or Reloaded customer that entered the Swarm would download "every piece of the game package." *See* Appx7425-7426, ¶ 9; Appx4696-4697, ¶ 10 & n.1.

That factual mistake by the district court fatally undermines its conclusion—and also highlights why its reasoning was so flawed. The district court said that the Blizzard and Reloaded Systems do not infringe because there is no "reallocation" if a peer always, eventually, acquires 100% of the content. Yet once the new client downloads its first game piece and sends a

[6] In its recitation of the facts regarding Blizzard, the district court mentioned that "[i]f a user did not exit the swarm, that user would eventually download 100% of the game package." Appx12. But the district court did not raise even that hint of doubt with respect to Reloaded. More importantly, for purposes of its *analysis*, the district court discarded the fact that some customers will not download full game packages, and instead assumed the opposite.

message to its peers, the infringement has occurred—because that message changes other peers' selection of what to download and store next.

Assume, for example, that a peer—Peer A—is participating in the Swarm. Peer B enters the Swarm, downloads a game piece from Peer C, and sends a message to Peer A saying that Peer B has this new game piece. That causes Peer A's list of what to download next to change. Then Peer A goes on to download an additional game piece, based on its updated list. Suddenly, Peer A drops out of the Swarm for some reason—for instance, a power interruption.

The system has fully met the Reallocation Limitation by that moment in time. There has been a reallocation of the portions of content to be stored by Peer A's cache storage. The infringement has already occurred, regardless of what happens next.

The fact that Peer A dropped out of the Swarm freezes the action, and makes it clear that there was infringement by that moment in time. But the fact that Peer A dropped out of the Swarm is not *why* there was infringement—it's not even really relevant. It is merely illustrative of the mistake the district court made.

The source of the district court's error is the assumption that an instrumentality might not infringe if it goes beyond what the patented

invention does, and performs additional steps. But that assumption is directly contrary to black-letter law.

The claims in the '145 Patent include, in the preamble, the word "comprising." That is critical, because "a method claim with the word 'comprising' appearing at the beginning generally allows for additional, unclaimed steps in the accused process." *David Netzer Consulting Eng'r LLC v. Shell Oil Co.*, 824 F.3d 989, 998 (Fed. Cir. 2016) (citing *Dippin' Dots v. Mosey*, 476 1337, 1343 (Fed. Cir. 2007)).[7] An "accused process does not avoid infringement by adding additional steps, *regardless* of whether such steps are 'extraneous' or mere 'colorable variations.'" *Dow Chem. Co. v. Sumitomo Chem. Co.*, 257 F.3d 1364, 1381 (Fed. Cir. 2001) (emphasis in original).

That is exactly the mistake of law the district court made here—which only compounded its mistake of fact about how the Blizzard and Reloaded

[7] *See also*, *e.g.*, *Genentech, Inc. v. Chiron Corp.*, 112 F.3d 495, 501 (Fed. Cir. 1997) ("'Comprising' is a term of art used in claim language which means that the named elements are essential, but other elements may be added and still form a construct within the scope of the claim.") (citing *In re Baxter*, 656 F.2d 679, 686 (C.C.P.A. 1981)); *Kaneka Corp. v. Xiamen Kingdomway Group Co.*, 790 F.3d 1298, 1306 (Fed. Cir. 2014) ("The claims' preamble term 'comprises' indicates that additional … steps or results are not excluded.") (citing *Boehringer Ingelheim Vetmedica, Inc. v. Schering-Plough Corp.*, 320 F.3d 1339, 1350 (Fed. Cir. 2003); *Amstar Corp. v. Envirotech Corp.*, 730 F.2d 1476, 1484 (Fed. Cir. 1984) ("When, as here, Envirotech adds its pre-mixing chamber to apparatus that, in its structure and in the process it performs, literally infringes claims 1, 8, and 9, infringement is not by that addition avoided.").

Systems operate. Either error on its own would mandate reversal. Even using the district court's flawed claim construction, Parallel Networks met its burden in opposing the Defendants' motions for summary judgment, and raised a genuine issue of material fact. The Court should reverse.

CONCLUSION

This Court should vacate the judgment of the district court, construe the Reallocation Limitation, and remand with instructions to conduct further proceedings on the issue of infringement.

STATEMENT REGARDING ORAL ARGUMENT

Parallel Networks respectfully requests that the Court hear oral argument on this Appeal.

Dated: December 13, 2016

Respectfully submitted,

/s/ Thomas R. Waskom
Gregory N. Stillman, Esq.
Hunton & Williams, LLP
500 East Main Street, Suite 1000
Norfolk, VA 23150
Telephone: (757) 640-5300

Thomas R. Waskom, Esq.
Hunton & Williams, LLP
951 East Byrd Street
Richmond, VA 23219
Telephone: (804) 788-8408

Counsel for Appellant Parallel Networks, LLC

ADDENDUM

TABLE OF CONTENTS

Page

Order on Summary Judgment Motions.....Appx1

Memorandum Opinion regarding Summary Judgment Motions.....Appx3

Order on Joint Claim Construction.....Appx21

U.S. Patent No. 7,188,145 B2.....Appx24

IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

PARALLEL NETWORKS, LLC, Plaintiff, v. KOG GAMES, INC., Defendant.	Civil Action No. 13-178-RGA
PARALLEL NETWORKS, LLC, Plaintiff, v. BLIZZARD ENTERTAINMENT, INC., Defendant.	Civil Action No. 13-826-RGA
PARALLEL NETWORKS, LLC, Plaintiff, v. RELOADED GAMES, INC., Defendant.	Civil Action No. 13-827-RGA
PARALLEL NETWORKS, LLC, Plaintiff, v. SG INTERACTIVE, INC., Defendant.	Civil Action No. 13-828-RGA

ORDER

For the reasons discussed in the accompanying Memorandum Opinion, **IT IS HEREBY ORDERED**:

Plaintiff's motions for partial summary judgment (C.A. No. 13-178, D.I. 130; C.A. No. 13-826, D.I. 151; C.A. No. 13-827, D.I. 119; C.A. No. 13-828, D.I. 125) are **DENIED**.

The motion for summary judgment filed by Blizzard (C.A. No. 13-826, D.I. 151) is **GRANTED**.

The motion for summary judgment filed by KOG, Reloaded, and SG Interactive (C.A. No. 13-178, D.I. 130; C.A. No. 13-827, D.I. 119; C.A. No. 13-828, D.I. 125) is **GRANTED**.

Entered this 31 day of August, 2016.

Richard G. Andrews
United States District Judge

IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

PARALLEL NETWORKS, LLC, Plaintiff, v. KOG GAMES, INC., Defendant.	Civil Action No. 13-178-RGA
PARALLEL NETWORKS, LLC, Plaintiff, v. BLIZZARD ENTERTAINMENT, INC., Defendant.	Civil Action No. 13-826-RGA
PARALLEL NETWORKS, LLC, Plaintiff, v. RELOADED GAMES, INC., Defendant.	Civil Action No. 13-827-RGA
PARALLEL NETWORKS, LLC, Plaintiff, v. SG INTERACTIVE, INC., Defendant.	Civil Action No. 13-828-RGA

MEMORANDUM OPINION

Adam W. Poff, Esq., James M. Lennon, Esq., Gregory J. Brodzik, Esq., Young Conaway Stargatt & Taylor LLP, Wilmington, DE; Brian A. Carpenter, Esq. (argued), Eric W. Buether, Esq., Christopher M. Joe, Esq., Michael D. Ricketts, Esq., Buether Joe & Carpenter, LLC, Dallas, TX, attorneys for Plaintiff Parallel Networks, LLC.

Kenneth L. Dorsney, Esq., Morris James LLP, Wilmington, DE; Eric A. Buresh, Esq. (argued), Michelle L. Marriot, Esq., Mark C. Lang, Esq., Erise IP, P.A., Overland Park, KS, attorneys for Defendants KOG Games, Inc., Reloaded Games, Inc., and SG Interactive, Inc.

Kenneth L. Dorsney, Esq., Morris James LLP, Wilmington, DE; Todd E. Landis, Esq. (argued), Eric J. Klein, Esq., Vinson & Elkins LLP, Dallas, TX, Fred I. Williams, Esq., Vinson & Elkins LLP, Austin, TX, Kellie M. Johnson, Esq., Akin Gump Strauss Hauer & Feld LLP, Dallas, TX, attorneys for Defendant Blizzard Entertainment, Inc.

August 31, 2016

ANDREWS, U.S. DISTRICT JUDGE:

Presently before the Court are cross-motions for summary judgment. In the cases against KOG, Reloaded, and SG Interactive ("the Reloaded Defendants"), Plaintiff Parallel Networks ("Parallel") moves for partial summary judgment of infringement with respect to the "re-allocating/re-allocate" limitation. (C.A. No. 13-178, D.I. 130).[1] The Reloaded Defendants move for summary judgment of non-infringement. (*Id.*). In the case against Blizzard, Parallel moves for partial summary judgment of infringement with respect to the "associating/associate" and "re-allocating/re-allocate" limitations. (C.A. No. 13-826, D.I. 151). Blizzard moves for summary judgment of non-infringement. (*Id.*). The issues have been fully briefed. The Court heard oral argument on May 3, 2016. (D.I. 138; C.A. No. 13-826, D.I. 161). For the reasons stated herein, Parallel's motions for partial summary judgment are both **DENIED**, the Reloaded Defendants' motion for summary judgment is **GRANTED**, and Blizzard's motion for summary judgment is **GRANTED**.

I. BACKGROUND

These are patent infringement cases. (D.I. 1). Parallel alleges infringement of U.S. Patent No. 7,188,145 ("the '145 patent"). (D.I. 130 at p. 5). On August 17, 2015, the Court held a Markman hearing. (D.I. 102). The Court construed a number of terms, including the "re-allocating/re-allocate" and "associating/associate" terms, upon which these motions focus. (D.I. 105). On October 14, 2015, pursuant to a stipulation filed by the parties, the Court allowed Parallel and the Reloaded Defendants to move for early summary judgment on the "re-allocating/re-allocate" limitation. (D.I. 117). On October 30, 2015, the Court allowed Parallel

[1] All citations will be to C.A. No. 13-178, unless otherwise indicated.

and Blizzard to move for early summary judgment on the "re-allocating/re-allocate" and "associating/associate" limitations. (C.A. No. 13-826, D.I. 140).[2]

The '145 patent is directed to a system and method for "dynamic distributed data caching." '145 patent Cover Page. The claims recite a "cache community" with at least "one peer." '145 patent Abstract.[3] "Each peer has an associated first content portion . . . indicating content to be cached by the respective peer." *Id.* When a "client" joins this cache community, the cache community's "peer list," which "indicates the peers in the cache community," "is updated to include the client." *Id.* Then, "[a] respective second content portion . . . is associated with each peer based on the addition of the client." *Id.* "The second content portion is distinct from the first content portion." *Id.* at 1:52-53. The "invention . . . reallocates the data to be cached by particular members of the distributed cache community based on the addition and subtraction of members to the distributed cache community." *Id.* at 4:32-34. The claimed invention allows for "decreased utilization of expensive connections to the Internet and increased utilization of cheaper local area network connections and broadband connections." *Id.* at 4:10-12. "By caching content at local machines on a local area network or on broadband connections to an Internet Service Provider, response time to requests for content is decreased by retrieving the content from local machines." *Id.* at 4:14-18.

The '145 patent has two independent claims: claim 1, a system claim, and claim 15, a method claim. The "re-allocating/re-allocate" and "associating/associate" limitations appear in both independent claims. During claim construction, the Court construed "re-allocating/re-

[2] Parallel has not raised a doctrine of equivalents argument in either case. Thus, both sets of cross-motions relate only to literal infringement.

[3] The parties do not dispute the meaning of "cache" or "caching." "Generally, it means storing data in a way that is relatively cheap to access (for example, in terms of time or bandwidth) in order to avoid getting it from a place that is more expensive to access." (D.I. 130 at 26).

allocate the cache storage of the content among the peers in the cache community" to mean "re-allocating/re-allocate the portions of the content to be stored by each peer's cache storage." (D.I. 105 at p. 2). The Court construed "associating/associate" to mean "[indicating/indicate] content to be cached by the client." (*Id.* at p. 3).

Claim 1, which is illustrative, reads:

> A method for dynamic distributed data caching comprising:
>
> providing a cache community on a first side of a point of presence, the cache community comprising at least one peer, the cache community being associated with content obtained from a second side of the point of presence, the content being cached by the at least one peer;
>
> allowing a client to join the cache community;
>
> updating a peer list associated with the cache community to include the client, the peer list indicating the peers in the cache community;
>
> associating the content with the client based on joinder of the client;
>
> re-allocating the cache storage of the content among the peers in the cache community in response to allowing the client to join the community.

'145 patent at 28:49-65.[4]

There are two accused products in these cases: Pando and the Blizzard Downloader. The two accused products are similar, but not identical. The relevant facts surrounding each are described separately.

A. Pando

The Reloaded Defendants and Parallel submitted a joint stipulation of facts. (D.I. 130 at pp. 1-4). I will summarize the relevant facts.[5] The Reloaded Defendants distributed video games through a system called Pando. Pando delivered content to end users through multiple

[4] While claim 1 recites "associating" and "re-allocating," claim 15 recites "associate" and "re-allocate."
[5] All facts are gleaned from the joint stipulation, unless otherwise indicated.

avenues, including direct downloads from a Content Delivery Network ("CDN") and peer-to-peer downloads from other users. These cases focus on the peer-to-peer aspect of Pando.

Game packages for video games are very large, often several gigabytes in size. (D.I. 131-2 at 80). To effectively share such large files, Pando allowed users to transfer small, manageable portions of data amongst themselves. The relevant unit of data is called is called a "piece." Pando divided game packages into "pieces," which were typically one megabyte in size.

When a user sought to download a game using Pando, that user downloaded the Pando Game Downloader, which in turn downloaded and installed the Pando Media Booster (the "PMB client"). The PMB client then connected to a server, called a tracker server, which kept a list of all the other active PMB clients downloading the same game package. This collection of clients, all downloading the same game, was called the "swarm." Before it began downloading the desired game package, a given PMB client ("the subject PMB client") first, and at a regular interval thereafter, sent an "Announce Message" to the tracker server. After the tracker server received the Announce Message, it added the subject PMB client to the list of PMB clients in the swarm.[6]

In response to the subject PMB client sending an Announce Message to the tracker server, the tracker server sent the subject PMB client a list of some subset of the clients in the swarm ("the Selected List"). The tracker server chose PMB clients for the Selected List based on network proximity and random selection. In response to each Announce Message, the tracker

[6] Every PMB client in the swarm was deemed either a "seed" or a "leech." A seed had successfully downloaded 100% of the game package, while a "leech" was in the process of downloading the game package. The tracker server knew whether a PMB client was a leech or a seed and how much data a given PMB client had downloaded, but did not know which pieces that PMB client had downloaded.

server sent the subject PMB client a new Selected List, which the subject PMB client merged with previously received Selected Lists to create a "Merged List." As a result of this process, two PMB clients in the swarm likely had non-identical Merged Lists.

The subject PMB client attempted to form connections with some or all of the PMB clients in its Merged List. A successful connection was referred to as an "established connection." A PMB client had a maximum limit on the number of established connections it could have at one time.

After establishing a connection, the subject PMB client and the PMB client to which it connected exchanged "Bit Field Messages." A Bit Field Message contained an inventory, called a bit field, for each piece of the game package a PMB client had. In other words, a Bit Field Message indicated which pieces a given PMB client possessed and which it did not. Whenever a PMB client successfully downloaded a new piece, it sent a "Have Message" to each of its established connections, thereby informing its established connections that it had a new piece of data to share.

Whenever the subject PMB client received a Bit Field Message or a Have Message from an established connection, it independently determined what, if anything, it should request from that established connection. The parties dispute the ways in which a PMB client selected which piece to download. Of central importance to Parallel's infringement theory is the "Rarest-First Algorithm." Parallel contends that the Rarest-First Algorithm was "the only strategy used by the PMB [c]lient to generate requests for content." (D.I. 130 at 15). The Reloaded Defendants argue that the algorithm was "not necessarily used to make requests, and in many instances will never be used at all," as a PMB client "must run at least three different sub-algorithms that request portions of content" before it may "get[] to the Rarest-First Algorithm." (*Id.* at 51). In

addressing the Reloaded Defendants' motion, I accept that the Rarest-First Algorithm was used by Pando.[7]

The parties do not dispute how the Rarest-First Algorithm functioned. When using the Rarest-First Algorithm, the subject PMB client generated a list of pieces which a particular established connection had, and which it had not requested from any other PMB client. (D.I. 130 at 16). The subject PMB client then assessed which piece, within that list, was the rarest.[8] (*Id.*). It then sought to download that rarest piece, or if there were several equally-rare pieces, it downloaded one of the equally-rare pieces. (*Id.*).

The downloading process continued until a PMB client obtained 100% of the game package. That is, the goal of every user was to obtain 100% of the game package, since the game could not be played until the entire game package was downloaded.

B. Blizzard Downloader

Blizzard and Parallel did not submit a joint stipulation of facts. The parties do, however, agree on most of the material facts.[9]

To deliver game content to users, Blizzard used two software applications: the Blizzard Installer and the Blizzard Downloader ("the BDL client"). (C.A. No. 13-826, D.I. 151 at 26, 50). This case concerns the BDL client. Blizzard's process began with game packages being divided

---

[7] Since Parallel is the non-moving party in the Reloaded Defendants' dispositive motion, I will "assume the truth of the [Parallel's] evidence, and draw all justifiable inferences in [its] favor." *Clark v. Sears, Roebuck & Co.*, 827 F. Supp. 1216, 1218 (E.D. Pa. 1993) (citing *Anderson v. Liberty Lobby, Inc.*, 477 U.S. 242, 255 (1986)).

[8] The process for determining rarity is as follows. Each PMB client maintained an integer value associated with each piece, called the "Have Count," which indicated the number of its established connections that had received that piece. (*Id.*). Upon receiving a Bit Field Message or a Have Message from one of its established connections, the subject PMB client updated the Have Count associated with a particular piece. (*Id.*).

[9] Blizzard relies on a declaration from fact witness Jay Lauffer, a senior software engineer at Blizzard. (C.A. No. 13-826, D.I. 153-10 at 111). Parallel relies on declaration from its expert, Dr. Peter Reiher. (C.A. No. 13-826, D.I. 153-9 at 8).

into pieces, which were typically 256 kilobytes in size. (*Id.* at 35, 50). Each piece was "hashed," thereby giving each piece a unique identifier, so that each piece could be verified as authentic. (*Id.*). After all pieces were hashed, all of the hashes for the game package were hashed again into a hash called "info_hash." (*Id.*). The info_hash, along with information about the hash for each piece, was stored in a "torrent" file. (*Id.* at 35-36, 50). This torrent file also contained a list of the files to be downloaded, and a list of the direct download addresses which allowed a user to download pieces from CDNs. (*Id.*).

When a user sought to download a game from Blizzard, the torrent file associated with that game was loaded into the user's BDL client ("the subject BDL client"). (*Id.* at 36, 51). After the user received the torrent file, the user began downloading game content through the CDNs provided in the torrent file. (*Id.*). If a user had enabled peer-to-peer sharing, the subject BDL client then extracted the info_hash value from the torrent file and sent it to Blizzard's tracker server through an Announce Message. (*Id.*). The tracker server then used a method called "Torrent::FindTracker" to retrieve a "Tracker" object, based on the info_hash value sent in the Announce Message. (*Id.* at 52).[10] This Tracker object was copied to a data field called "m_tracker" within the "Peer" object that the tracker server had associated with the joining client. (*Id.*).

Thereafter, the tracker server sent an "Announce Response" to the subject BDL client. That response included a Selected List, a list of BDL clients which were some subset of the total BDL clients in the swarm. (*Id.*). The subject BDL client then attempted to connect to some or all of the BDL clients in its Selected List. (*Id.* at 13). As with Pando, successful connections

[10] With respect to this process, Blizzard does not address some of the facts described by Parallel. I therefore understand them to be undisputed. To the extent there is a dispute, I will assume the truth of Parallel's evidence.

were referred to as established connections. (*Id.* at 14). Established connections then exchanged Bit Field Messages, each of which detailed which pieces of the game package a given BDL client had. (*Id.*). After the subject BDL client received a Bit Field Message, it selected which piece to download, if any, from that established connection. (*Id.*). After successfully downloading a piece from another BDL client in the swarm, a BDL client sent a "Have Message" to all of its established connections, informing them that it had a new piece available for download. (*Id.*).

Blizzard created a priority system for its game packages.[11] (*Id.* at 37, 54-55). Each piece of the game package was placed into one of three buckets: a red bucket, a yellow bucket, and a green bucket. (*Id.* at 37). The red bucket included all game files required to begin playing the game. (*Id.*). Users had to download all of the files within the red bucket before beginning to download files in the yellow bucket, and had to download all of the files in the yellow bucket before beginning to download the files in the green bucket. (*Id.*). While downloading files within a particular bucket, the BDL client used a "Rarest-First Algorithm." (*Id.* at 16, 54-55). The Rarest-First Algorithm employed by the BDL client is identical, in all material respects, to that employed by Pando. (*Id.* at 16-17).

If a user did not exit the swarm, that user would eventually download 100% of the game package. (*Id.* at 38, 43). After a user finished downloading the game, that user was considered a "seed," and soon exited the swarm. (*Id.*).

---

[11] There are three versions of the BDL client. The first two versions are accused, but the third is not. This priority system was only implemented in the second version of the BDL client. Blizzard previously agreed "that it [would] not present any non-infringement arguments . . . on the 're-allocating' and 'associating' limitations that peer-to-peer functionality differ[ed] among different versions of Blizzard source code." (C.A. No. 13-826, D.I. 140 at p. 2). Thus, Blizzard is precluded from making any non-infringement arguments based on the priority system, or based on the operation of the third version of the BDL client.

II. LEGAL STANDARD

"The 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." Fed. R. Civ. P. 56(a). The moving party has the initial burden of proving the absence of a genuinely disputed material fact relative to the claims in question. *Celotex Corp. v. Catrett*, 477 U.S. 317, 330 (1986). Material facts are those "that could affect the outcome" of the proceeding, and "a dispute about a material fact is 'genuine' if the evidence is sufficient to permit a reasonable jury to return a verdict for the nonmoving party." *Lamont v. New Jersey*, 637 F.3d 177, 181 (3d Cir. 2011) (quoting *Anderson v. Liberty Lobby, Inc.*, 477 U.S. 242, 248 (1986)). The burden on the moving party may be discharged by pointing out to the district court that there is an absence of evidence supporting the non-moving party's case. *Celotex*, 477 U.S. at 323.

The burden then shifts to the non-movant to demonstrate the existence of a genuine issue for trial. *Matsushita Elec. Indus. Co. v. Zenith Radio Corp.*, 475 U.S. 574, 586-87 (1986); *Williams v. Borough of West Chester, Pa.*, 891 F.2d 458, 460-61 (3d Cir. 1989). A non-moving party asserting that a fact is genuinely disputed must support such an assertion by: "(A) citing to particular parts of materials in the record, including depositions, documents, electronically stored information, affidavits or declarations, stipulations . . . , admissions, interrogatory answers, or other materials; or (B) showing that the materials cited [by the opposing party] do not establish the absence . . . of a genuine dispute . . . ." Fed. R. Civ. P. 56(c)(1).

When determining whether a genuine issue of material fact exists, the court must view the evidence in the light most favorable to the non-moving party and draw all reasonable inferences in that party's favor. *Scott v. Harris*, 550 U.S. 372, 380 (2007); *Wishkin v. Potter*, 476 F.3d 180, 184 (3d Cir. 2007). If the non-moving party fails to make a sufficient showing on

an essential element of its case with respect to which it has the burden of proof, the moving party is entitled to judgment as a matter of law. *See Celotex*, 477 U.S. at 322.

A patent is infringed when a person "without authority makes, uses, offers to sell, or sells any patented invention, within the United States . . . during the term of the patent." 35 U.S.C. § 271(a). A two-step analysis is employed in making an infringement determination. *See Markman v. Westview Instruments, Inc.*, 52 F.3d 967, 976 (Fed. Cir. 1995) (en banc), *aff'd*, 517 U.S. 370 (1996). First, the court must construe the asserted claims to ascertain their meaning and scope. *See id.* The trier of fact must then compare the properly construed claims with the accused infringing product. *See id.* at 976. This second step is a question of fact. *See Bai v. L & L Wings, Inc.*, 160 F.3d 1350, 1353 (Fed. Cir. 1998).

"Literal infringement of a claim exists when every limitation recited in the claim is found in the accused device." *Kahn v. Gen. Motors Corp.*, 135 F.3d 1472, 1477 (Fed. Cir. 1998). "If any claim limitation is absent from the accused device, there is no literal infringement as a matter of law." *Bayer AG v. Elan Pharm. Research Corp.*, 212 F.3d 1241, 1247 (Fed. Cir. 2000). If an accused product does not infringe an independent claim, it also does not infringe any claim depending thereon. *See Wahpeton Canvas Co. v. Frontier, Inc.*, 870 F.2d 1546, 1553 (Fed. Cir. 1989). The patent owner has the burden of proving infringement and must meet its burden by a preponderance of the evidence. *See SmithKline Diagnostics, Inc. v. Helena Lab. Corp.*, 859 F.2d 878, 889 (Fed. Cir. 1988) (citations omitted).

When an accused infringer moves for summary judgment of non-infringement, such relief may be granted only if at least one limitation of the claim in question does not read on an element of the accused product. *See Chimie v. PPG Indus., Inc.*, 402 F.3d 1371, 1376 (Fed. Cir. 2005); *see also TechSearch, L.L.C. v. Intel Corp.*, 286 F.3d 1360, 1369 (Fed. Cir. 2002)

("Summary judgment of noninfringement is . . . appropriate where the patent owner's proof is deficient in meeting an essential part of the legal standard for infringement, because such failure will render all other facts immaterial."). Thus, summary judgment of non-infringement can only be granted if, after viewing the facts in the light most favorable to the non-movant, there is no genuine issue as to whether the accused product is covered by the claims (as construed by the court). *See Pitney Bowes, Inc. v. Hewlett–Packard Co.*, 182 F.3d 1298, 1304 (Fed. Cir. 1999).

III. ANALYSIS

A. The Reloaded Defendants' Motion for Summary Judgment

Parallel's infringement theory centers on the execution of the Rarest-First Algorithm under certain circumstances. Parallel contends that the "re-allocating/re-allocate" limitation is met in two ways. First, when an existing peer[12] adds to its Have Count in response to receiving the first Have Message from a new peer that has just joined the swarm, [13] its view of rarity will be altered. After executing the Rarest-First Algorithm, the existing peer may request a different piece than it otherwise would have, had the new peer not joined the swarm and sent its first Have Message. Specifically, Parallel argues that the reallocation limitation is met when an existing peer adds to its Have Count and executes the Rarest-First Algorithm. Second, when a peer joins the swarm and sends its first Have Message to its established connections, the established connections will alter their respective Have Counts. For each peer, after executing the Rarest-First Algorithm, "the order of the [p]ieces to be downloaded and stored in the future" may be affected. (D.I. 130 at 14). Specifically, Parallel argues that the reallocation limitation is met

[12] "Peer" is the term used by the '145 patent. In the accused Pando system, each peer is a PMB client. An existing peer is a PMB client that has already joined the swarm.

[13] The claims require that the reallocation occur "in response to allowing the client to join the community." '145 patent at 28:63-65, 30:1-4. Thus, Parallel contends that this limitation is satisfied when a Have Message is sent from a new peer that was allowed to join the swarm.

when the peer that has just joined the swarm sends its first Have Message. These arguments make up two sides of the same coin. According to Parallel, sending or receiving a Have Message constitutes a reallocation, because, following the execution of the Rarest-First Algorithm, the Have Message may alter the order in which pieces are downloaded.

The reallocation limitation requires that there be a "re-allocation" of "the portions of the content to be stored by each peer's cache storage." (D.I. 105 at p. 2). The use of "portions" in the construction is important. At the Markman hearing, I noted that the "primary dispute" between the parties was "whether each peer [could] cache 100 percent of the content." (D.I. 102 at 115). I concluded that each peer could not, and stated that "each peer receives some but not all of the content." (*Id.*). "[I]f each peer had all the data, the invention would face the same problems that existed in the prior art." (*Id.*).

Parallel's infringement theory is fatally flawed. In the accused Pando system, the "portions of the content to be stored" by each PMB client are never altered. In fact, every user will use the same amount of space to store the same data. In other words, each PMB client will download every piece of the game package, regardless of the order in which those pieces are downloaded. Accordingly, the portions of the content to be stored are never reallocated, or, indeed, altered in any way.

Parallel urges the Court to look at the transfer of a particular piece, from peer to peer, in isolation. When a new peer joins the swarm and sends its first Have Message, Parallel argues, an existing peer's view of rarity is affected. Then, upon execution of the Rarest-First Algorithm, a PMB client will download pieces in an order that depends on its view of rarity. When the reordering occurs as a result of a Have Message being sent or received, Parallel argues that a reallocation has occurred. According to Parallel, this "re-allocation" occurs every time a new

peer sends a Have Message, up until an existing peer actually has 100% of the content.[14] No reasonable juror could reach that conclusion.

When the '145 patent recites reallocation, it envisions some degree of divided responsibility for portions of content. For instance, the specification explains that the "[d]ynamic cache application 428 . . . reallocates the content 460 to be cached by particular members 412 and master 410 so that the newly added client 404 is responsible for some subset of content 460 cached in community 402." '145 patent at 21:10-13; *see also id.* at 4:32-34 ("The present invention also reallocates the data to be cached by particular members of the distributed cache community based on the addition and subtraction of members to the distributed cache community."). Whether or not there is some overlap in that divided responsibility[15]—i.e., some level of data duplication between peers in the cache community—does not alter the necessary conclusion that "each peer receives some [portion,] but not all of the content." (D.I. 102 at 115). The claimed invention requires that a new peer's entry into the swarm causes the particular portions of content that each peer is responsible for storing to change.

With Pando, the portions of the content to be stored by each peer never changes. Each PMB client constantly progresses toward its goal of receiving 100% of the content. When a

---

[14] In light of the Court's claim construction ruling, Parallel argues that infringement occurs during "the period of time between a [p]eer being allowed to join and the [p]eer obtaining 100% of the [g]ame [p]ackage." (D.I. 130 at 55).

[15] The parties dispute whether a reallocation inherently requires some removal of content, rather than duplication. In other words does reallocation require moving a piece from peer A to peer B, or can it simply be copying that piece, such that peer A and peer B both have a copy? I think the '145 patent contemplates both. As the specification explains:

> Redistribution may comprise actually copying cached content from one client . . . to another, or removing or expiring content no longer cached at a particular server. Simply removing the cached items may cause a request that could have been satisfied by the community cache to be forwarded to the origin server, but avoids the bandwidth usage associated with copying. The decision whether to copy or remove may be configured at modules 124, 144 and 164 and may consider bandwidth usage issues . . . .

'145 patent at 16:41-50. Although the parties contest this issue at length, I do not think it is material to the outcome here.

PMB client adds to its Have Count and executes the Rarest-First Algorithm, all that is decided is which piece the PMB client should copy from another PMB client at that instant. The PMB client, from the moment it joined the swarm, always sought to download that piece. In other words, the PMB client would have downloaded that piece regardless of whether a new PMB client ever joined the swarm and sent a Have Message. This cannot constitute a reallocation of "the portions of the content to be stored by each" PMB client. (D.I. 105 at p. 2).

Parallel notes that "the act of incrementing the Have Count affecting [an] [e]xisting [p]eer's view of rarity . . . constitute[s] a re-allocation of the order the [p]ieces will be downloaded in the future." (D.I. 130 at 29). Herein lies the problem with Parallel's theory. The '145 patent does not require a reallocation "of the order the [p]ieces will be downloaded;" it requires a reallocation of the "portions of the content to be stored" among the peers. Additionally, Parallel's expert is correct that "'to be' is prospective in nature." (D.I. 131-4 at 78). Thus, a reallocation would require a change in what each peer ultimately seeks to store. That does not happen in the Pando system.

Elsewhere in its briefing, Parallel argues that "[t]he 'Have Count' . . . affected each receiving [p]eer's choice . . . as to what [p]ieces of content . . . it would request and store in the future." (D.I. 130 at 30). This statement not only contradicts Parallel's own arguments about "re-allocat[ing] . . . the order the [p]ieces will be downloaded," but also the stipulated facts. (*Id.* at 29). In the stipulated facts, Parallel agreed that "[t]he goal of each PMB [c]lient that participated in a [s]warm was to successfully download and store 100% of the [g]ame [p]ackage." (*Id.* at 10). Thus, the Have Count does not affect the pieces of content that would be requested and stored by a peer in the future. It only affects the order in which those pieces will be requested. To put it succinctly, a reordering is not a reallocation. Parallel's infringement

theory is premised on an assumption that changing the order in which a particular client downloads content from another constitutes a reallocation of the portions of content among the peers in the cache community. No reasonable juror could so conclude.

Parallel has failed to show a genuine dispute of material fact which would preclude the entry of summary judgment.

B. Parallel's Motion for Partial Summary Judgment Against the Reloaded Defendants

Parallel moves for partial summary judgment of infringement on the "re-allocating/re-allocate" limitation. Having concluded that the Reloaded Defendants are entitled to summary judgment of non-infringement, Parallel's motion for summary judgment is denied.

C. Blizzard's Motion for Summary Judgment

Blizzard moves for summary judgment of non-infringement. While the Reloaded Defendants focused solely on the "re-allocating/re-allocate" limitation, Blizzard focuses on both that limitation and the "associating/associate" limitation. Since I find the "re-allocating/re-allocate" limitation to be dispositive, I need not and do not address the "associating/associate" limitation.

Parallel's infringement theory for the BDL client is identical, in all material respects, to its infringement theory for Pando. Generally speaking, Parallel argues that sending or receiving a Have Message constitutes a reallocation, because, following the execution of the Rarest-First Algorithm, the Have Message may alter the order in which pieces are downloaded.

Parallel's infringement theory fails with respect to Blizzard for the same reasons it failed with respect to the Reloaded Defendants. As with Pando, every BDL client downloads 100% of the game package. That is, every user utilizes the same amount of space to store the exact same data. There is no divided responsibility, as no user ultimately stores less than 100% of the

content. While the Rarest-First Algorithm may alter the order in which certain pieces are downloaded, it does not alter the portions of the content to be stored by each user in the swarm.

Therefore, a reasonable juror could not conclude that the accused BDL client satisfies the "re-allocating/re-allocate" limitation. Blizzard's motion for summary judgment of non-infringement is granted.

D. Parallel's Motion for Partial Summary Judgment Against Blizzard

Parallel moves for partial summary judgment of infringement on the "re-allocating/re-allocate" and the "associating/associate" limitations. Having concluded that Blizzard is entitled to summary judgment of non-infringement, Parallel's motion for partial summary judgment is denied.

IV. CONCLUSION

For the reasons set forth above, the Reloaded Defendants' motion for summary judgment of non-infringement is **GRANTED**, Blizzard's motion for summary judgment of non-infringement is **GRANTED**, and Parallel's motions for partial summary judgment of infringement are **DENIED**. An appropriate order will be entered.

IN THE UNITED STATES DISTRICT COURT

FOR THE DISTRICT OF DELAWARE

PARALLEL NETWORKS, LLC, Plaintiff, v. KOG GAMES, INC., Defendant.	C.A. No. 13-178-RGA
PARALLEL NETWORKS, LLC, Plaintiff, v. RIOT GAMES, INC., Defendant.	C.A. No. 13-183-RGA
PARALLEL NETWORKS, LLC, Plaintiff, v. TURBINE, INC., Defendant.	C.A. No. 13-184-RGA
PARALLEL NETWORKS, LLC, Plaintiff, v. BLIZZARD ENTERTAINMENT, INC., Defendant.	C.A. No. 13-826-RGA
PARALLEL NETWORKS, LLC, Plaintiff, v. RELOADED GAMES, INC., Defendant.	C.A. No. 13-827-RGA

PARALLEL NETWORKS, LLC, Plaintiff, v. SG INTERACTIVE, INC., Defendant.	C.A. No. 13-828-RGA
PARALLEL NETWORKS, LLC, Plaintiff, v. KOLLECTIVE TECHNOLOGY, INC., Defendant.	C.A. No. 13-914-RGA

[PROPOSED] CLAIM CONSTRUCTION ORDER

As set forth in its Oral Order on August 17, 2015, the Court has determined that, as used in the asserted claims of U.S. Patent No. 7,188,145 ("the '145 Patent"), the following terms have the following meanings:

Term	Court's Construction
"cache community" '145 Patent Claims 1, 4, 15, 16	"one or more peers which operate as a group to cache data"
"re-allocating/re-allocate the cache storage of the content among the peers in the cache community" '145 Patent Claims 1, 12, 14, 15, 26, 28	"re-allocating/re-allocate the portions[1] of the content to be stored by each peer's cache storage"

Further, the parties to the above-captioned matters previously agreed that, as used in the asserted claims of the '145 Patent, the following terms have the following meanings (*See* C.A. No. 13-178-RGA, D.I. 98 at 9), which constructions are hereby adopted by the Court:

[1] Although the parties recall that the construction orally given by the Court during the hearing of August 17, 2015, used the plural "portions," the singular "portion" appears in the Transcript of the hearing (at page 116, l. 6). To the extent the construction reflected above is not what the Court intended, the Parties will submit a revised proposed order in accordance with the Court's instructions.

Term	Agreed Construction
"associate/ing the content with the client"	"indicate/ing content to be cached by the client"
"determining a second peer associated with the requested content"	"determining a second peer that is responsible for caching the requested content"
"peer"	"a computer that is a member of a cache community"
"point of presence"	"a communications interface defining a boundary between a cache community on one side and an origin server on the other"
"first side of a point of presence"	"the side of a communications interface on which the cache community exists"
"second side of a point of presence"	"the side of a communications interface on which the origin server exists"
"content"	No construction necessary.

SO ORDERED this 24 day of August, 2015.

Richard G. Andrews
United States District Judge



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

(10) **Patent No.: US 7,188,145 B2**
(45) **Date of Patent: Mar. 6, 2007**

(54) **METHOD AND SYSTEM FOR DYNAMIC DISTRIBUTED DATA CACHING**

(75) Inventors: **Keith A. Lowery**, Richardson, TX (US); **Bryan S. Chin**, Plano, TX (US); **David A. Consolver**, Arlington, TX (US); **Gregg A. DeMasters**, Plano, TX (US)

(73) Assignee: **EpicRealm Licensing LLC**, Dallas, TX (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 875 days.

(21) Appl. No.: **09/759,406**

(22) Filed: **Jan. 12, 2001**

(65) **Prior Publication Data**

US 2002/0107934 A1 Aug. 8, 2002

(51) **Int. Cl.**
G06F 15/167 (2006.01)

(52) **U.S. Cl.** **709/214**; 709/203; 709/217; 709/219; 709/226; 707/10; 707/200; 710/56; 711/119; 711/126; 711/130; 711/135

(58) **Field of Classification Search** 709/213, 709/217, 201, 212, 203, 205, 215, 216, 248; 707/8, 10; 711/151
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,603,382	A *	7/1986	Cole et al.	710/56
5,522,045	A *	5/1996	Sandberg	709/215
5,537,572	A *	7/1996	Michelsen et al.	711/135
5,701,427	A *	12/1997	Lathrop	709/237
5,778,185	A *	7/1998	Gregerson et al.	709/226
5,864,854	A *	1/1999	Boyle	707/10
5,940,838	A *	8/1999	Schmuck et al.	707/200
5,968,176	A *	10/1999	Nessett et al.	713/201
5,987,477	A *	11/1999	Schmuck et al.	707/201
5,988,847	A *	11/1999	McLaughlin et al.	700/9
6,006,254	A *	12/1999	Waters et al.	709/205
6,026,474	A	2/2000	Carter et al.	711/202
6,065,102	A *	5/2000	Peters et al.	711/151
6,098,064	A *	8/2000	Pirolli et al.	707/2

(Continued)

FOREIGN PATENT DOCUMENTS

EP	0993 163	A1	4/2000

(Continued)

OTHER PUBLICATIONS

Author: John Borland, Article entitled "Net video not yet ready for prime time", printed from web site www.cnet.com on Aug. 15, 2000.

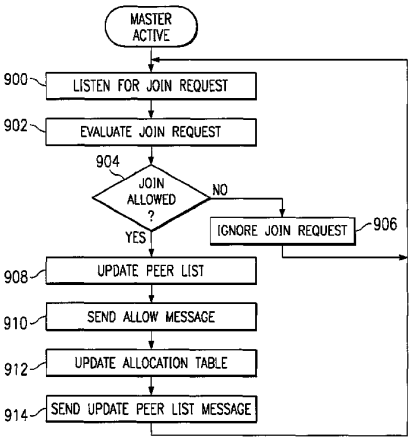
(Continued)

Primary Examiner—Saleh Najjar
Assistant Examiner—Michael Won
(74) *Attorney, Agent, or Firm*—Baker Botts L.L.P.

(57) **ABSTRACT**

A method and system for dynamic distributed data caching is presented. The method includes providing a cache community (**402**) comprising at least one peer (**413**). Each peer has an associated first content portion (**511**) indicating content to be cached by the respective peer. A client (**404**) may be allowed to join the cache community. A peer list (**426**) associated with the cache community is updated to include the client. The peer list indicates the peers in the cache community. A respective second content portion (**511**) is associated with each peer based on the addition of the client.

36 Claims, 11 Drawing Sheets



U.S. PATENT DOCUMENTS

6,112,279	A *	8/2000	Wang	711/119
6,122,629	A *	9/2000	Walker et al.	707/8
6,167,438	A *	12/2000	Yates et al.	709/216
6,167,490	A *	12/2000	Levy et al.	711/148
6,199,179	B1 *	3/2001	Kauffman et al.	714/26
6,205,481	B1 *	3/2001	Heddaya et al.	709/226
6,263,302	B1 *	7/2001	Hellestrand et al.	703/17
6,330,605	B1 *	12/2001	Christensen et al.	709/226
6,477,150	B1 *	11/2002	Maggenti et al.	370/312
6,487,583	B1 *	11/2002	Harvey et al.	709/204
6,542,926	B2 *	4/2003	Zalewski et al.	709/213
6,725,261	B1 *	4/2004	Novaes et al.	709/220
6,785,704	B1 *	8/2004	McCanne	718/105
2002/0026560	A1 *	2/2002	Jordan et al.	711/120
2002/0103972	A1 *	8/2002	Satran et al.	711/119

FOREIGN PATENT DOCUMENTS

WO	WO 98/22891	5/1998

OTHER PUBLICATIONS

Author: Corey Grice, Article entitled "Start-up taps swapping to ease Web bottlenecks", printed from web site www.cnet.com on Aug, 15, 2000.

Author: VTrails.com. Product information regarding company's Full Duplex Packet Cascading technology printed from web site www.vtrails.com on Aug. 15, 2000.

Author Vinod Valloppillil and Keith W. Ross, Internet-Draft document entitled "Cache Array Routing Protocol v1.0", printed from web site www.globecom.net on Nov. 14, 2000.

Author: Microsoft Corporation. Information regarding the Cache Array Routing Protocol (CARP) and Microsoft Proxy Server 2.0, printed from web site www.msdn.microsoft.com on Oct. 9, 2000.

Author: Geek.com. Information news article entitled "Cash for your unused CPU cycles", printed from web site www.geek.com on Jun. 30, 2000.

Author: Dcypher.net. Answers to frequently answered questions v1.5, printed from web site www.dcypher.net on Jun. 30, 2000.

Author: Popular Power. Information regarding the company's Popular Power for Windows software, printed from web site www.popularpower.com on Jun. 30, 2000.

Author: ProcessTree Network. Information regarding their "for-pay" distributed processing network, printed from web site www.distributedscience.com on Jun. 30, 2000.

Author: Keith Schultz, Article entitled "Pushing content to the Internet's Edge", printed from web site www.internetweek.com on Jan. 11, 2001.

U.S. Appl. No. 09/759,392, filed Jan. 12, 2001, entitled "*Method And System For Community Data Caching*,", 78 total pages.

Greg Barish, et al., "World Wide Web Caching: Trends and Techniques," XP-000949799, IEEE Communications Magazine, May 2000, pp. 178-185.

PCT/US 02-00886 Search Report, 6 pages, Oct. 2000.

PCT/US 02-00886 Search Report, 10 pages, Mar. 27, 2003.

Inohara, et al., "Self-Organizing Cooperative WWW Caching," © 1998 *IEEE* (pp. 74-83).

Zhang, et al., "Adaptive Web Caching," *Proceedings of the 1997 NLANR Web Cache Workshop*, Apr. 25, 1997 (9 pages).

Michel, et al., "Adaptive Web Caching: Towards a New Global Caching Architecture," © 1998 *Elsevier Science B.V.*, Computer Networks and ISDN Systems 30 (1998) (pp. 2169-2177).

Karger, et al., "Web Caching with Consistent Hashing," © 1999 *Elsevier Science B.V.*, Computer Networks 31 (1999)(pp. 1203-1213).

* cited by examiner

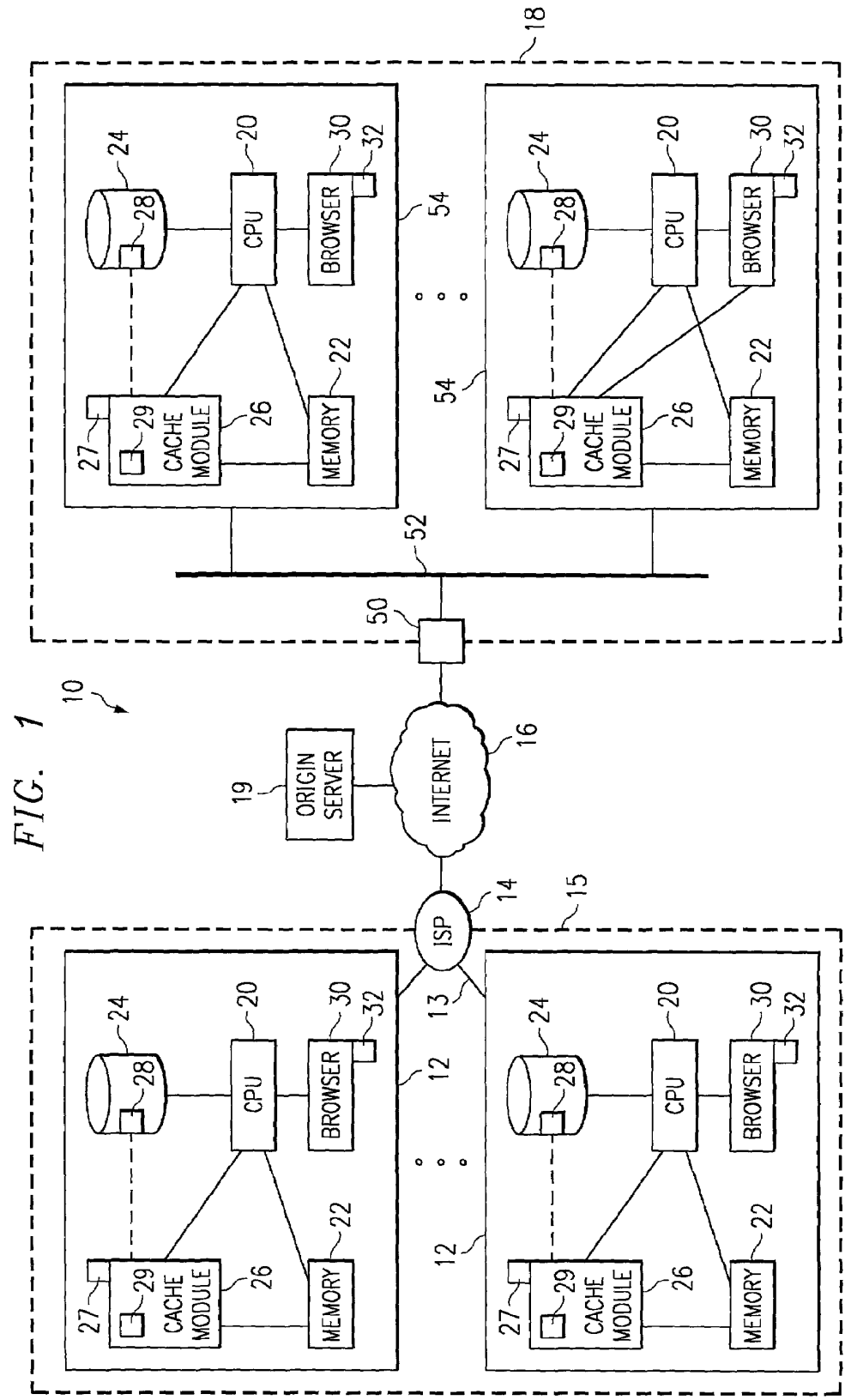

18
24
28
20
CPU
30
BROWSER
32
54
22
MEMORY
26
27
29
CACHE MODULE
52
50
10
19
ORIGIN SERVER
INTERNET
16
ISP
14
15
13
12

FIG. 1

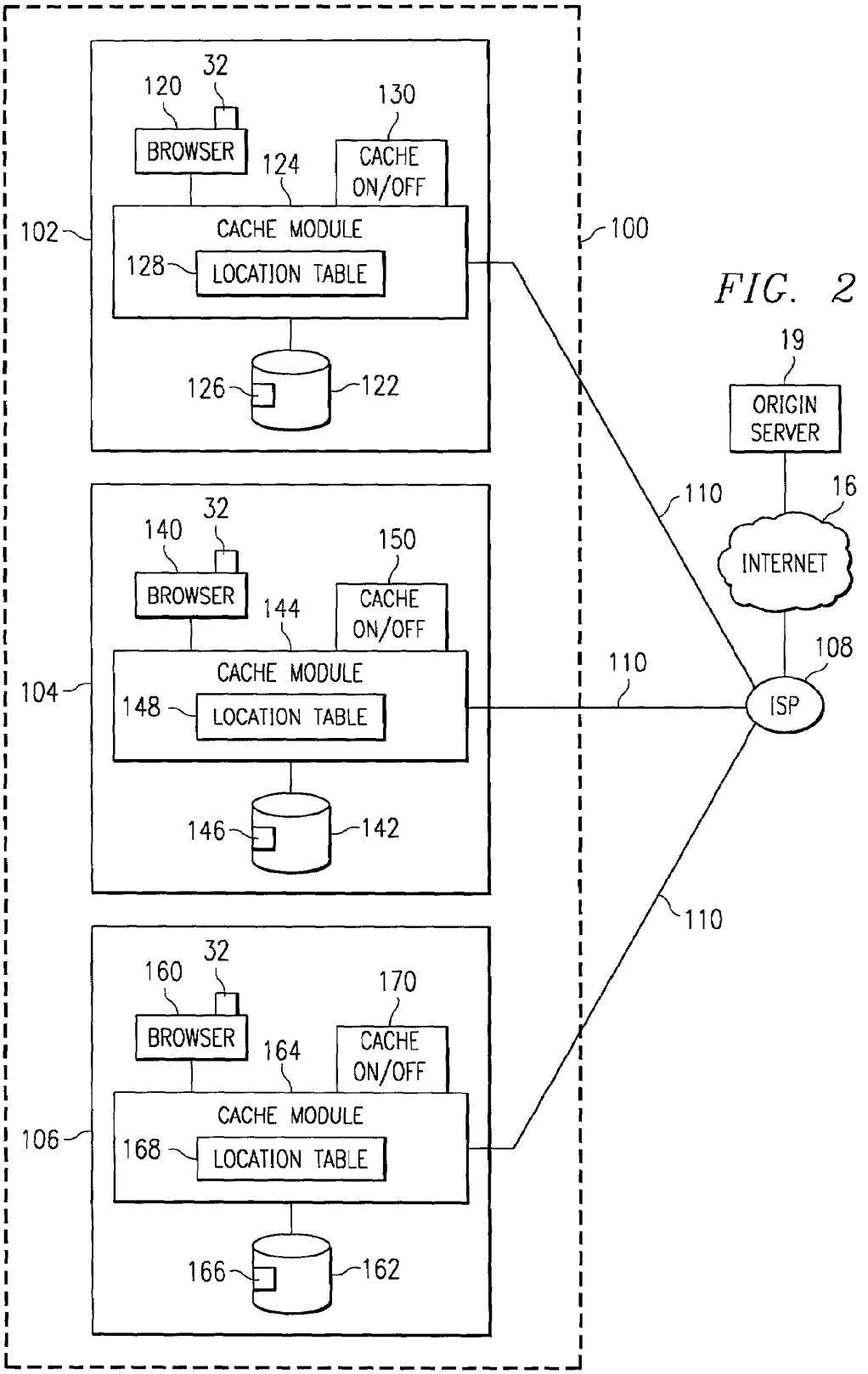

32
120
130
BROWSER
124
CACHE ON/OFF
102
CACHE MODULE
100
128
LOCATION TABLE
126
122
19
ORIGIN SERVER
110
16
32
140
150
BROWSER
144
CACHE ON/OFF
INTERNET
CACHE MODULE
110
108
104
148
LOCATION TABLE
ISP
146
142
110
32
160
170
BROWSER
164
CACHE ON/OFF
CACHE MODULE
106
168
LOCATION TABLE
166
162

FIG. 2

FIG. 3

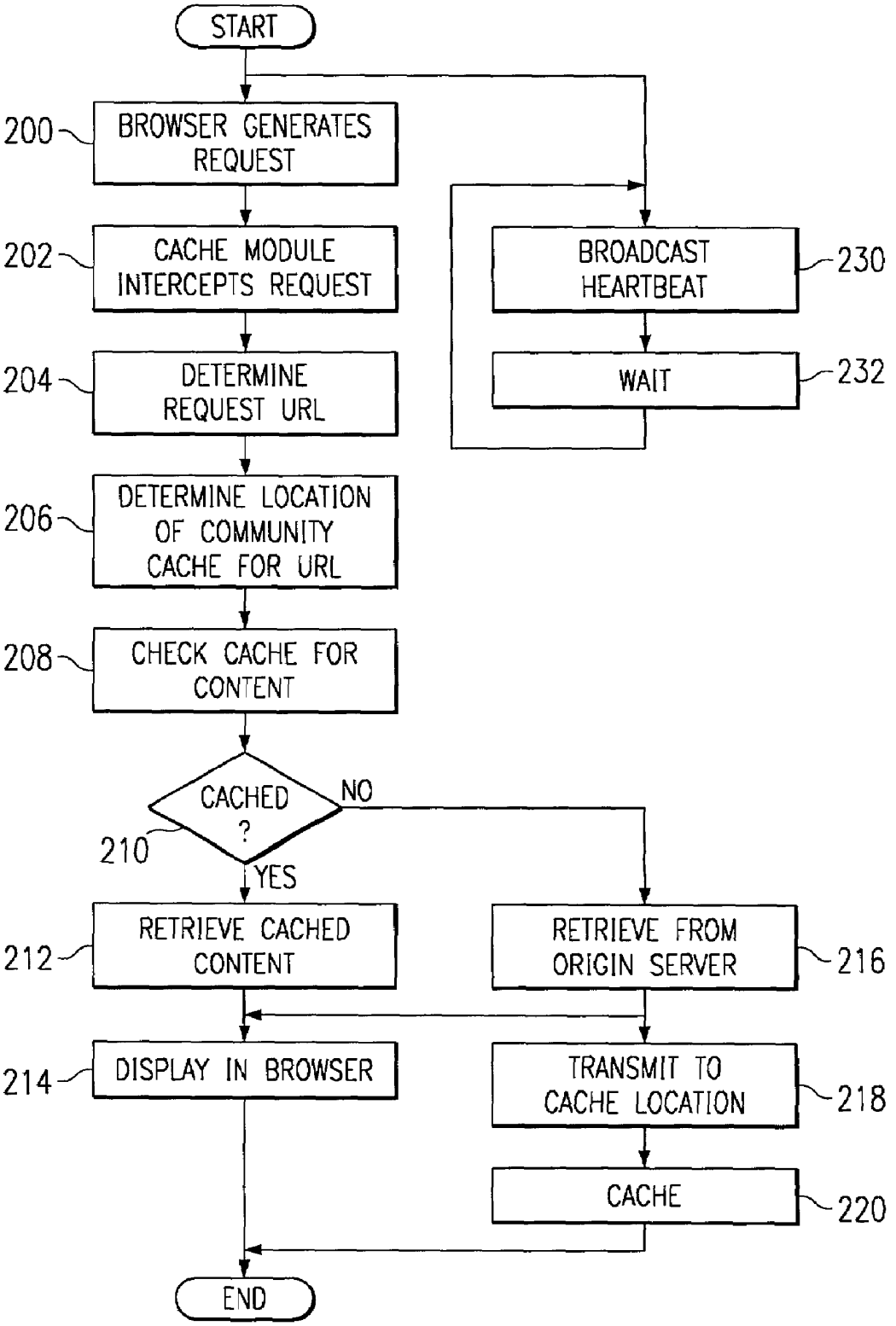

START
200
BROWSER GENERATES REQUEST
202
CACHE MODULE INTERCEPTS REQUEST
204
DETERMINE REQUEST URL
206
DETERMINE LOCATION OF COMMUNITY CACHE FOR URL
208
CHECK CACHE FOR CONTENT
CACHED ?
210
NO
YES
212
RETRIEVE CACHED CONTENT
214
DISPLAY IN BROWSER
END
BROADCAST HEARTBEAT
230
WAIT
232
RETRIEVE FROM ORIGIN SERVER
216
TRANSMIT TO CACHE LOCATION
218
CACHE
220

FIG. 4

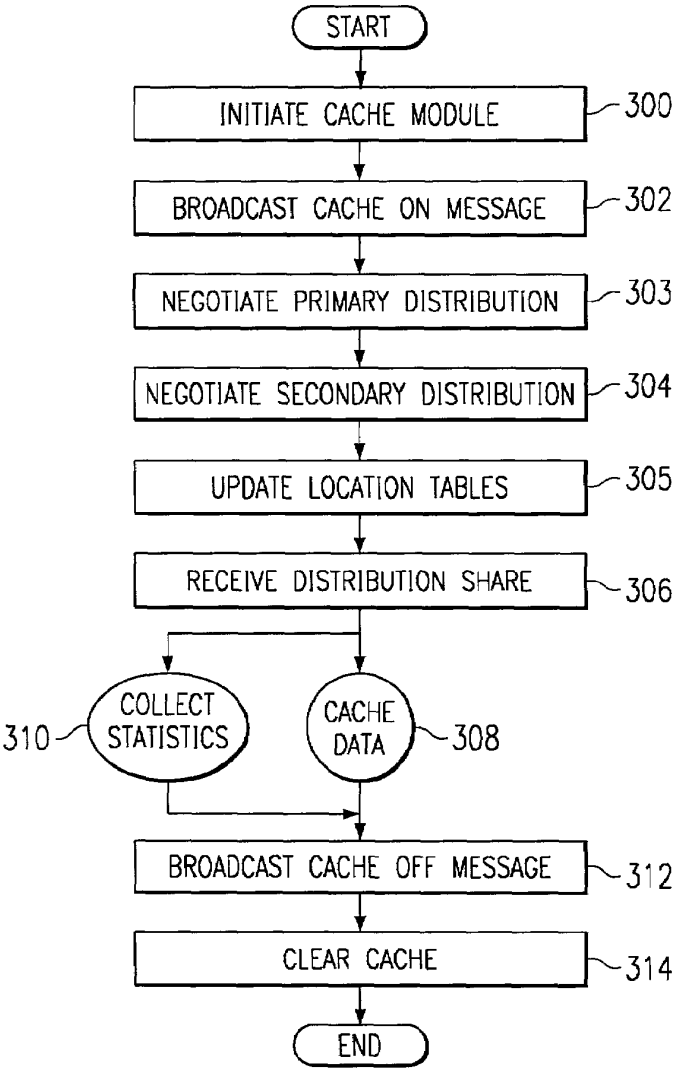

START
INITIATE CACHE MODULE
300
BROADCAST CACHE ON MESSAGE
302
NEGOTIATE PRIMARY DISTRIBUTION
303
NEGOTIATE SECONDARY DISTRIBUTION
304
UPDATE LOCATION TABLES
305
RECEIVE DISTRIBUTION SHARE
306
COLLECT STATISTICS
310
CACHE DATA
308
BROADCAST CACHE OFF MESSAGE
312
CLEAR CACHE
314
END

FIG. 5

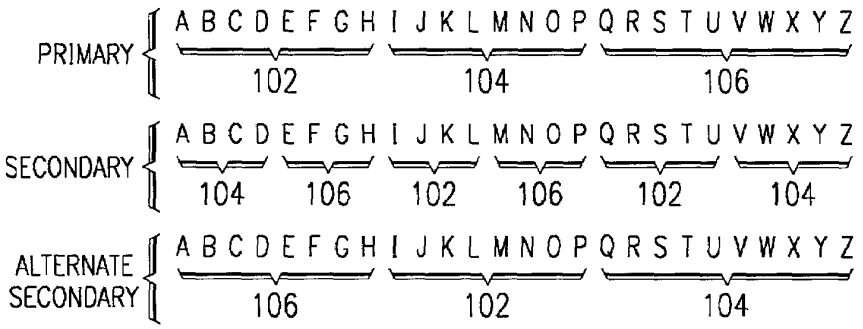

PRIMARY
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
102
104
106
SECONDARY
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
104
106
102
106
102
104
ALTERNATE SECONDARY
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
106
102
104

FIG. 6

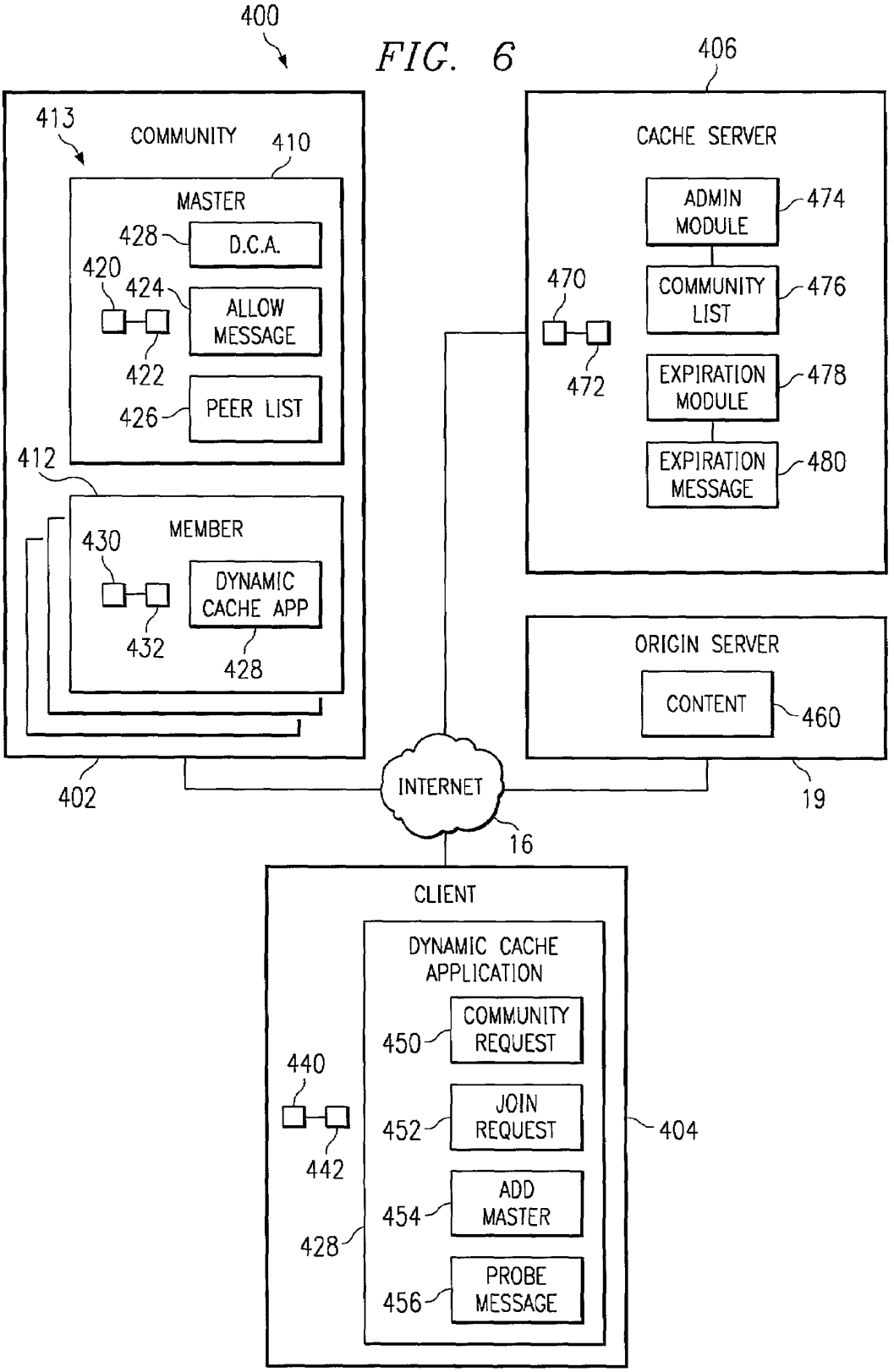

400
406
413
COMMUNITY
410
MASTER
428
D.C.A.
420
424
ALLOW MESSAGE
422
426
PEER LIST
412
MEMBER
430
DYNAMIC CACHE APP
432
428
CACHE SERVER
ADMIN MODULE
474
470
COMMUNITY LIST
476
472
EXPIRATION MODULE
478
EXPIRATION MESSAGE
480
ORIGIN SERVER
CONTENT
460
INTERNET
16
402
19
CLIENT
DYNAMIC CACHE APPLICATION
COMMUNITY REQUEST
450
440
JOIN REQUEST
452
404
442
ADD MASTER
454
428
PROBE MESSAGE
456

FIG. 7

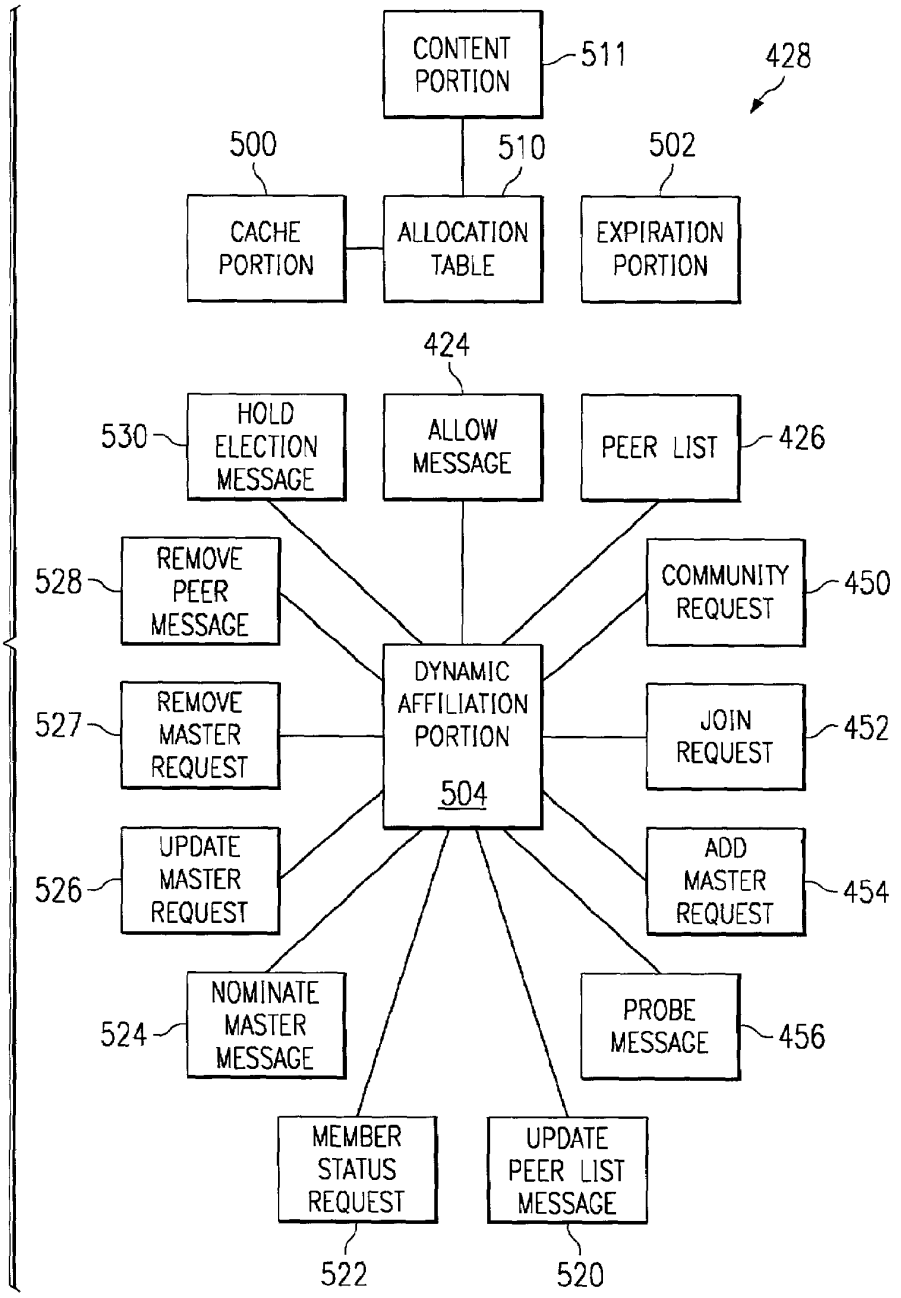
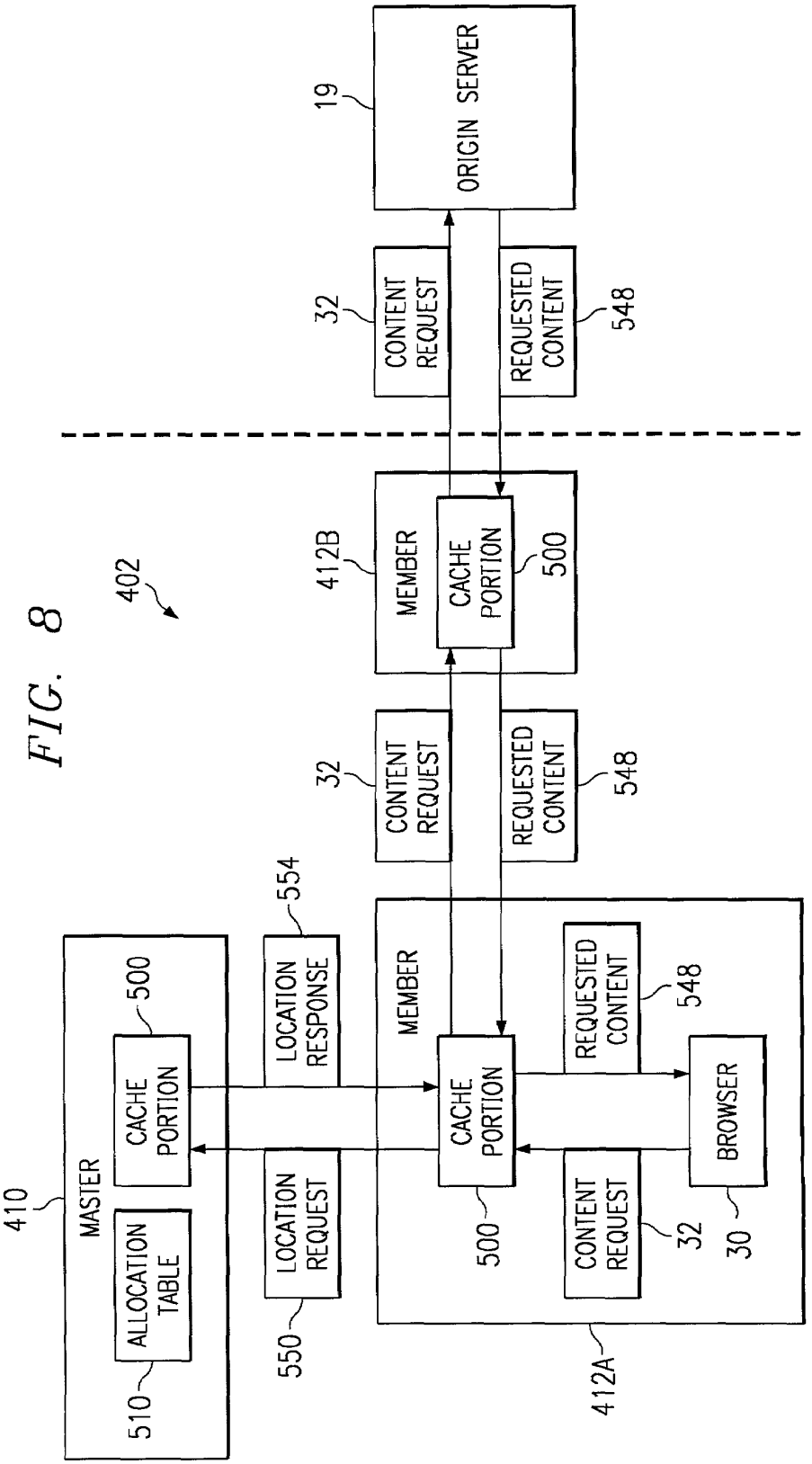

CONTENT PORTION
511
428
500
510
502
CACHE PORTION
ALLOCATION TABLE
EXPIRATION PORTION
424
530
HOLD ELECTION MESSAGE
ALLOW MESSAGE
PEER LIST
426
528
REMOVE PEER MESSAGE
COMMUNITY REQUEST
450
527
REMOVE MASTER REQUEST
DYNAMIC AFFILIATION PORTION
504
JOIN REQUEST
452
526
UPDATE MASTER REQUEST
ADD MASTER REQUEST
454
524
NOMINATE MASTER MESSAGE
PROBE MESSAGE
456
MEMBER STATUS REQUEST
UPDATE PEER LIST MESSAGE
522
520

FIG. 8


ORIGIN SERVER
19
CONTENT REQUEST
32
REQUESTED CONTENT
548
402
MEMBER
412B
CACHE PORTION
500
CONTENT REQUEST
32
REQUESTED CONTENT
548
554
LOCATION RESPONSE
MEMBER
REQUESTED CONTENT
548
500
CACHE PORTION
CACHE PORTION
BROWSER
MASTER
410
ALLOCATION TABLE
510
LOCATION REQUEST
550
500
CONTENT REQUEST
32
30
412A

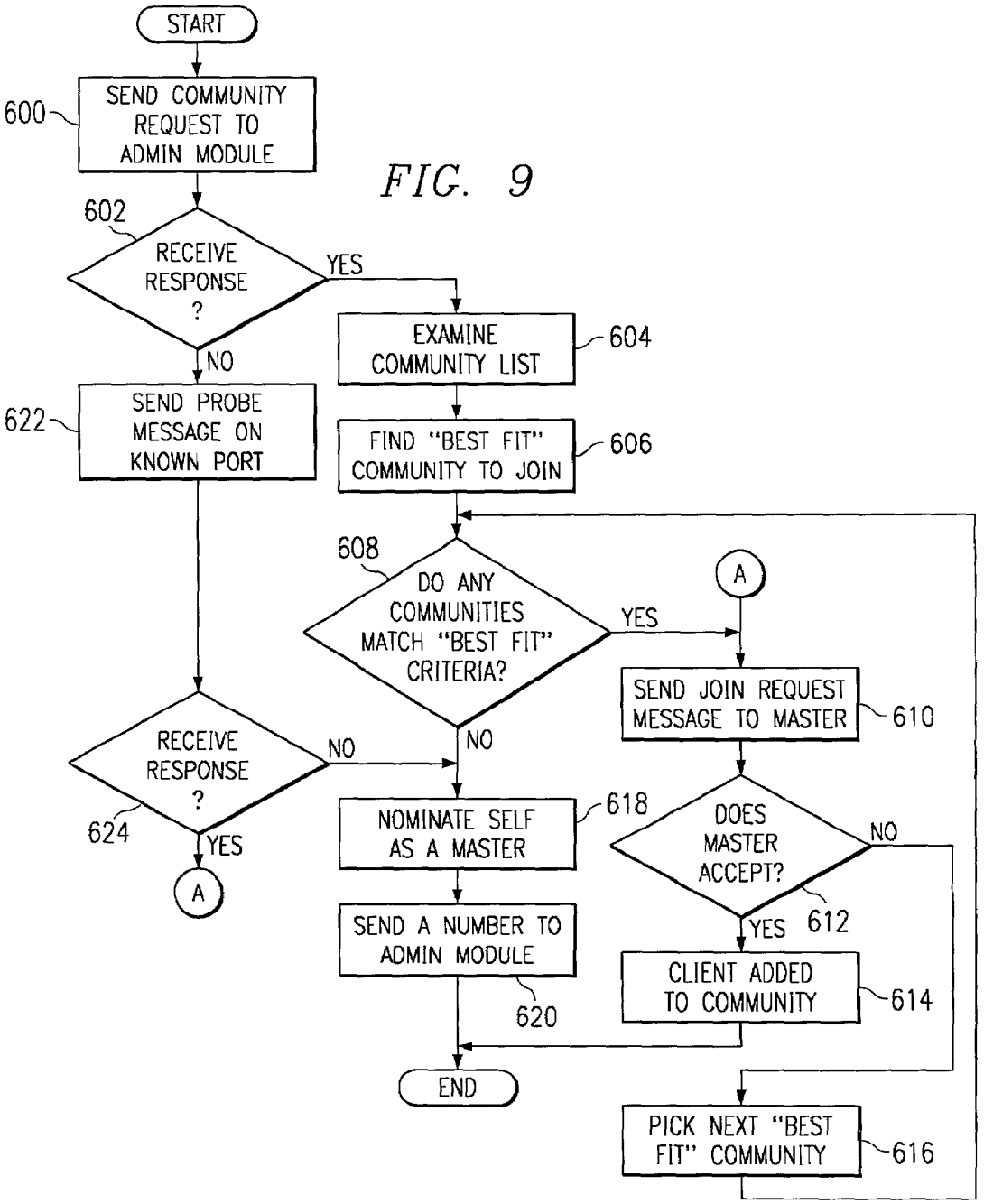

START
600
SEND COMMUNITY REQUEST TO ADMIN MODULE
602
RECEIVE RESPONSE ?
YES
NO
EXAMINE COMMUNITY LIST
604
622
SEND PROBE MESSAGE ON KNOWN PORT
FIND "BEST FIT" COMMUNITY TO JOIN
606
608
DO ANY COMMUNITIES MATCH "BEST FIT" CRITERIA?
YES
NO
A
SEND JOIN REQUEST MESSAGE TO MASTER
610
RECEIVE RESPONSE ?
NO
624
YES
A
NOMINATE SELF AS A MASTER
618
DOES MASTER ACCEPT?
NO
YES
612
SEND A NUMBER TO ADMIN MODULE
620
CLIENT ADDED TO COMMUNITY
614
END
PICK NEXT "BEST FIT" COMMUNITY
616

FIG. 9

FIG. 10

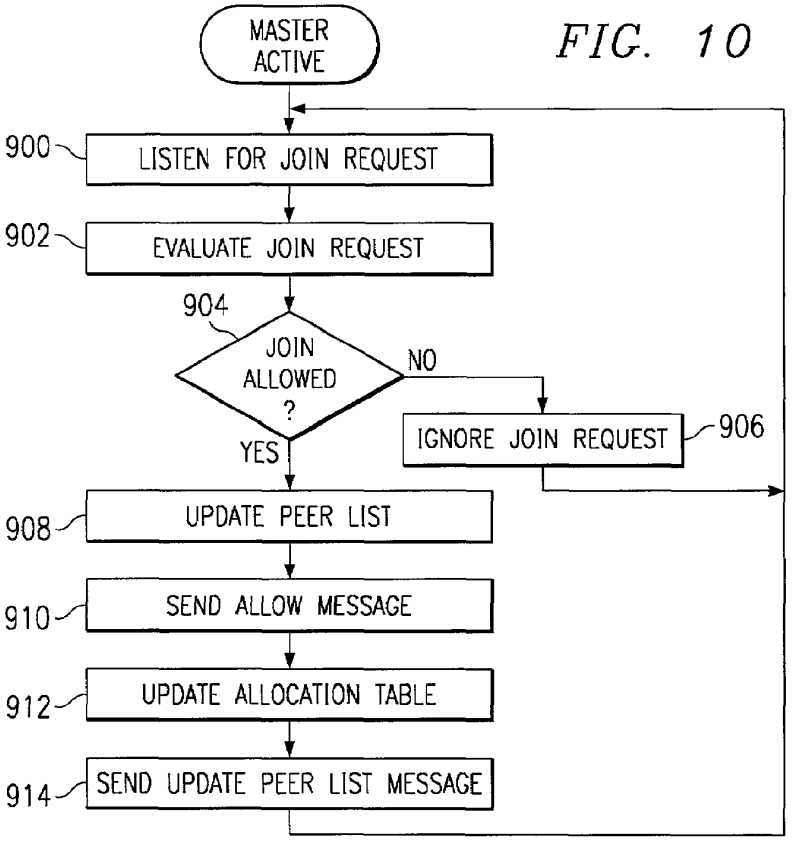

MASTER ACTIVE
900
LISTEN FOR JOIN REQUEST
902
EVALUATE JOIN REQUEST
904
JOIN ALLOWED ?
NO
YES
IGNORE JOIN REQUEST
906
908
UPDATE PEER LIST
910
SEND ALLOW MESSAGE
912
UPDATE ALLOCATION TABLE
914
SEND UPDATE PEER LIST MESSAGE

FIG. 11

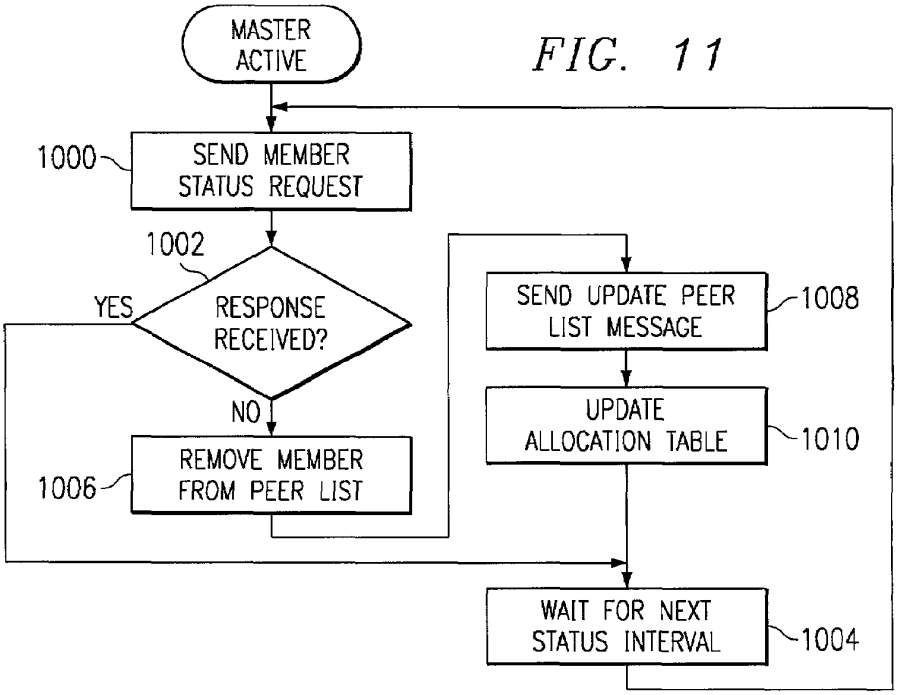

MASTER ACTIVE
1000
SEND MEMBER STATUS REQUEST
1002
RESPONSE RECEIVED?
YES
NO
1006
REMOVE MEMBER FROM PEER LIST
SEND UPDATE PEER LIST MESSAGE
1008
UPDATE ALLOCATION TABLE
1010
WAIT FOR NEXT STATUS INTERVAL
1004

FIG. 12

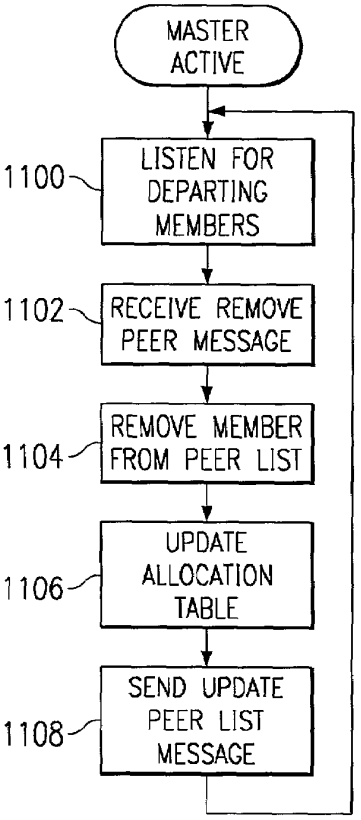

MASTER ACTIVE
1100
LISTEN FOR DEPARTING MEMBERS
1102
RECEIVE REMOVE PEER MESSAGE
1104
REMOVE MEMBER FROM PEER LIST
1106
UPDATE ALLOCATION TABLE
1108
SEND UPDATE PEER LIST MESSAGE

FIG. 13

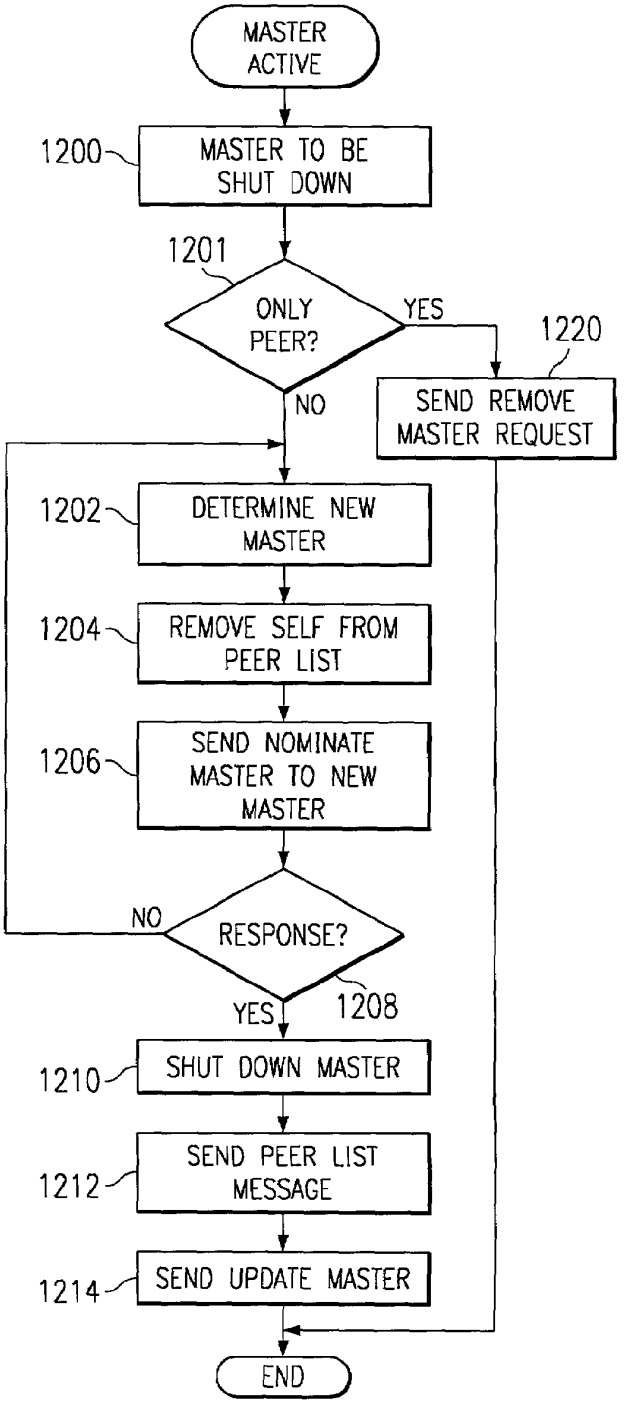
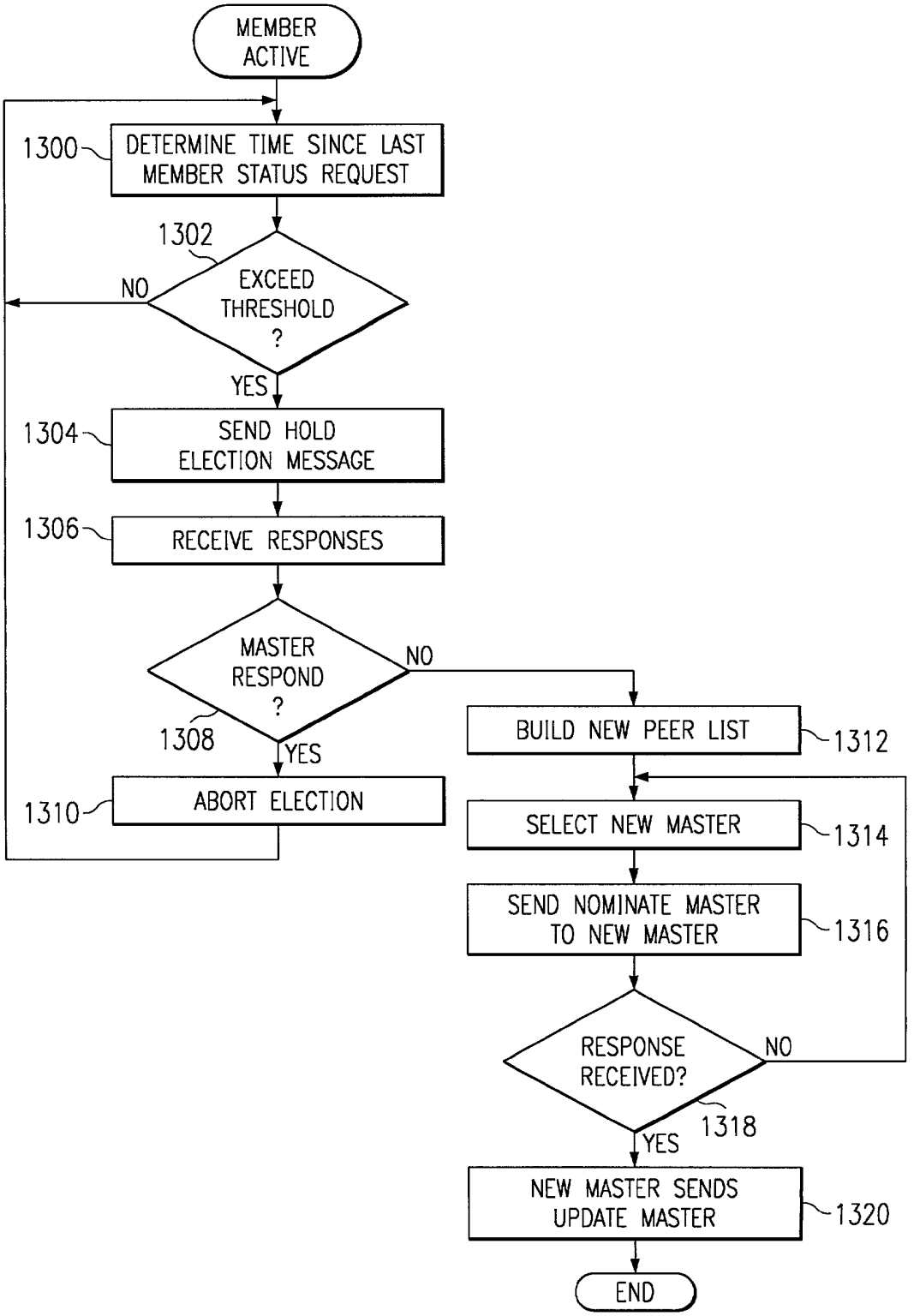

MASTER ACTIVE
1200
MASTER TO BE SHUT DOWN
1201
ONLY PEER?
YES
1220
SEND REMOVE MASTER REQUEST
NO
1202
DETERMINE NEW MASTER
1204
REMOVE SELF FROM PEER LIST
1206
SEND NOMINATE MASTER TO NEW MASTER
NO
RESPONSE?
1208
YES
1210
SHUT DOWN MASTER
1212
SEND PEER LIST MESSAGE
1214
SEND UPDATE MASTER
END

FIG. 14


MEMBER ACTIVE
1300
DETERMINE TIME SINCE LAST MEMBER STATUS REQUEST
1302
EXCEED THRESHOLD ?
NO
YES
1304
SEND HOLD ELECTION MESSAGE
1306
RECEIVE RESPONSES
MASTER RESPOND ?
NO
1308
YES
1310
ABORT ELECTION
BUILD NEW PEER LIST
1312
SELECT NEW MASTER
1314
SEND NOMINATE MASTER TO NEW MASTER
1316
RESPONSE RECEIVED?
NO
1318
YES
NEW MASTER SENDS UPDATE MASTER
1320
END

METHOD AND SYSTEM FOR DYNAMIC DISTRIBUTED DATA CACHING

TECHNICAL FIELD OF THE INVENTION

This invention relates in general to the field of data processing systems and, more particularly, to a method and system for dynamic distributed data caching.

BACKGROUND OF THE INVENTION

As computers have grown increasingly important in today's society, the importance of the Internet has also increased. As increasing numbers of users access the Internet, the need for efficient use of bandwidth has also increased. The increasing numbers of requests handled by the Internet are increasing the delay experienced by a user between generating a request and receiving a response to the request because of bandwidth limitations.

One traditional solution to decreasing bandwidth usage and decreasing the delay experienced by the user has involved caching previously requested content at the user's computer for faster retrieval. A related traditional solution has involved caching previously requested content for multiple users at a single cache server. Another traditional solution has involved increasing the bandwidth of the network connection between the Internet, the user and the web servers handling the requests. However, traditional solutions have often failed as the number of requests continue to increase and overload single cache servers and because of the expense associated with maintaining large numbers of high speed connections to the Internet. In addition, the traditional solutions have not utilized the "always-on" nature of newer broadband connections such as digital subscriber line and cable modems.

SUMMARY OF THE INVENTION

From the foregoing, it may be appreciated that a need has arisen for a method and system for dynamic distributed data caching to provide more efficient use of bandwidth.

According to one embodiment of the present invention, a method for dynamic distributed data caching is provided. The method comprises providing a cache community comprising at least one peer. Each peer has an associated first content portion indicating content to be cached by the respective peer. The method further comprises allowing a client to join the cache community, updating a peer list associated with the cache community to include the client, the peer list indicating the peers in the cache community, and associating a respective second content portion with each peer based on the addition of the client. The second content portion is distinct from the first content portion.

According to another embodiment of the present invention, a system for dynamic distributed data caching is presented. The system comprises logic encoded on storage. The logic is operable to provide a cache community comprising at least one peer. Each peer has an associated first content portion indicating content to be cached by the respective peer and allow a client to join the cache community. The logic is further operable to update a peer list associated with the cache community to include the client. The peer list indicates the peers in the cache community. The logic is further operable to associate a respective second content portion with each peer based on the addition of the client. The second content portion is distinct from the first content portion.

According to a further embodiment of the present invention, a method for dynamic distributed data caching is presented. The method comprises determining that a first master associated with a cache community is non-operational, electing a second master to replace the first master in the cache community, and allocating at least one content portion based on the loss of the first master.

According to a yet another embodiment of the present invention, a system for dynamic distributed data caching is presented. The system comprises logic encoded on storage. The logic is operable to determine that a first master associated with a cache community is non-operational, elect a second master to replace the first master in the cache community, and allocate at least one content portion based on the loss of the first master.

According to yet a further embodiment of the present invention, a method for dynamic distributed caching is presented. The method comprises requesting a list of cache communities from a cache server and determining whether at least one existing cache community exists. The method further comprises attempting to join a one of the existing cache communities when the existing cache communities are found and generating a new cache community when no existing cache communities are found.

According to another embodiment of the present invention, a system for dynamic distributed caching is presented. The system comprises logic encoded on storage. The logic is operable to request a list of cache communities from a cache server and determine whether at least one existing cache community exists. The logic is further operable to attempt to join a one of the existing cache communities when the existing cache communities are found and generate a new cache community when no existing cache communities are found.

According to a further embodiment of the present invention, a method for dynamic distributed data caching is presented. The method comprises generating a content request for requested content at a first peer in a cache community, determining a second peer associated with the requested content, the second peer being associated with the cache community, and retrieving, by the first peer, the requested content from the second peer.

According to yet another embodiment of the present invention, a system for dynamic distributed data caching is presented. The system comprises logic encoded on storage. The logic is operable to generate a content request for requested content at a first peer in a cache community, determine a second peer associated with the requested content, the second peer being associated with the cache community, and retrieve, by the first peer, the requested content from the second peer.

According to yet a further embodiment of the present invention, a method for dynamic distributed data caching is presented. The method comprises communicating a community request from a dynamic cache module to an administration module and receiving a community list from the administration module in response to the community request, the community list including a list of communities. The method further comprises generating a join request to attempt to join a one of the communities in the community list and receiving an allow message associated with the one of the communities. The method further comprises receiving a peer list associated with the one of the communities, receiving a content request, and storing content associated with the content request.

According to an additional embodiment of the present invention, a system for dynamic distributed data caching is

presented. The system comprises logic encoded on storage. The logic is operable to communicate a community request from a dynamic cache module to an administration module and receive a community list from the administration module in response to the community request. The community list includes a list of communities. The logic is further operable to generate a join request to attempt to join a one of the communities in the community list and receive an allow message associated with the one of the communities. The logic is further operable to receive a peer list associated with the one of the communities, receive a content request, and store content associated with the content request.

According to a further additional embodiment of the present invention, a system for dynamic distributed data caching is presented. The system comprises means for providing a cache community comprising at least one peer. Each peer has an associated first content portion indicating content to be cached by the respective peer. The system further comprises means for allowing a client to join the cache community and means for updating a peer list associated with the cache community to include the client. The peer list indicates the peers in the cache community. The system further comprises means for associating a respective second content portion with each peer based on the addition of the client. The second content portion is distinct from the first content portion.

According to yet a further additional embodiment of the present invention, a system for dynamic distributed data caching is presented. The system comprises means for determining that a first master associated with a cache community is non-operational, means for electing a second master to replace the first master in the cache community, and means for allocating at least one content portion based on the loss of the first master.

According to yet another further additional embodiment of the present invention, a system for dynamic distributed caching is presented. The system comprises means for requesting a list of cache communities from a cache server and means for determining whether at least one existing cache community exists. The system further comprises means for attempting to join a one of the existing cache communities when the existing cache communities are found and means for generating a new cache community when no existing cache communities are found. According to another additional embodiment of the present invention, a system for dynamic distributed data caching is presented. The system comprises means for generating a content request for requested content at a first peer in a cache community and means for determining a second peer associated with the requested content. The second peer is associated with the cache community. The system further comprises means for retrieving, by the first peer, the requested content from the second peer.

According to yet a further additional embodiment of the present invention, a system for dynamic distributed data caching is presented. The system comprises means for communicating a community request from a dynamic cache module to an administration module and means for receiving a community list from the administration module in response to the community request. The community list includes a list of communities. The system further comprises means for generating a join request to attempt to join a one of the communities in the community list and means for receiving an allow message associated with the one of the communities. The system further comprises means for receiving a peer list associated with the one of the communities, means for receiving a content request, and means for storing content associated with the content request.

The present invention provides a number of technical advantages. Various embodiments of the present invention may provide all, some or none of these advantages. One such technical advantage is the capability to support a dynamic distributed caching system. In addition, the distributed caching system is supportable without the use of specialized hardware as standard personal computers may be used to support the distributed caching system. A further technical advantage is decreased utilization of expensive connections to the Internet and increased utilization of cheaper local area network connections and broadband connections, such as digital subscriber line and cable modems. By caching content at local machines on a local area network or on broadband connections to an Internet Service Provider, response time to requests for content is decreased by retrieving the content from local machines. Additional benefits may be realized by allowing more client machines to utilize a single connection to the Internet by decreasing the amount of bandwidth needed by particular client machines.

Another technical advantage is the capability to dynamically add and remove members from a distributed cache community. In contrast to traditional distributed caching systems, which have typically required a human administrator to add and remove members from the distributed caching system, the present invention provides the capability to dynamically add and remove members from the distributed cache community. Also, members may be added or removed from the cache community without the intervention of a human administrator. The present invention also reallocates the data to be cached by particular members of the distributed cache community based on the addition and subtraction of members to the distributed cache community.

BRIEF DESCRIPTION OF THE DRAWINGS

A better understanding of the present invention will be realized from the detailed description that follows, taken in conjunction with the accompanying drawings, in which:

FIG. **1** is a block diagram illustrating a community cache system;

FIG. **2** is a block diagram illustrating an exemplary community cache constructed according to the teachings of the present invention;

FIG. **3** is a flowchart illustrating a method for community caching according to the teachings of the present invention;

FIG. **4** is a flowchart illustrating a method for generating a community cache according to the teachings of the present invention;

FIG. **5** is a diagram illustrating an exemplary distribution of cache shares according to the teachings of the present invention;

FIG. **6** is a block diagram illustrating a dynamic caching system according to one embodiment of the system of FIG. **1**;

FIG. **7** is a block diagram illustrating details of the dynamic cache application according to one embodiment of the present invention;

FIG. **8** is a flow diagram illustrating a method for retrieving and caching content within a cache community according to one embodiment of the present invention;

FIG. **9** is a flow chart illustrating a method for adding a client to the cache community according to one embodiment of the present invention;

FIG. **10** is a flow chart illustrating a method for allowing the client to join the cache community according to one embodiment of the present invention;

FIG. **11** is a flow chart illustrating a method for determining whether a member of the cache community has unexpectedly departed the cache community according to one embodiment of the present invention;

FIG. **12** is a flow chart illustrating a method for gracefully removing the member from the cache community according to one embodiment of the present invention;

FIG. **13** is a flow chart illustrating a method for gracefully removing a master from the cache community according to one embodiment of the present invention; and

FIG. **14** is a flow chart illustrating a method for allowing the master to unexpectedly depart the cache community according to one embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

FIG. **1** is a block diagram illustrating a community cache system **10**. System **10** comprises one or more Internet Service Provider (ISP) clients **12**, an ISP **14**, an ISP caching community **15**, a network **16**, an Intranet caching community **18** and an origin server **19**.

Client **12** comprises a processor **20**, a computer readable memory **22**, a computer readable storage device **24**, a cache module **26** and a browser **30**. Client **12** may be adapted to execute any of the well-known MS-DOS, PC-DOS, OS/2, UNIX, Linux, MAC-OS, mainframe, minicomputer and Windows operating systems or other operating systems. Processor **20** comprises any suitable general purpose or specialized electronic processing device, such as a central processing unit (CPU), operable to communicate with memory **22** and storage device **24**, and further to execute cache module **26** and browser **30**. Memory **22** comprises any suitable combination of transient or persistent memory operable to store cache module **26** and browser **30**, and to communicate with processor **20**. Storage device **24** comprises any suitable combination of optical, magnetic or other computer readable storage medium such as a floppy disk drive, a hard disk drive, a CD-ROM drive, a CD-RW drive, a magnetic tape drive or an optical drive. Storage device **24** may also represent multiple computer readable storage devices. Storage device **24** includes a cache portion **28**.

Cache portion **28** comprises a portion of storage device **24** used by cache module **26** for caching data. Access to cache portion **28** may be controlled by cache module **26** so as to prevent user modification of data stored in cache portion **28**. Cache portion **28** may comprise one or more directories, one or more logical partitions, one or more distinct physical devices and other suitable physical and logical elements.

Cache module **26** comprises a software application operable to manage cache portion **28** of storage device **24**. Cache module **26** is operable to monitor the activities of browser **30** and to cache content items retrieved by browser **30**. Cache module **26** is also operable to respond to content requests from browser **30** using content cached in cache portions **28** at clients **12** in community **15**. In one embodiment, cache module **26** may use the Cache Array Routing Protocol (CARP) to determining the location of content within community **15**. Cache module **26** is configurable such that limits may be placed on the size of cache portion **28** and the amount of processor time used on processor **20** by cache module **26**. For example, a user associated with a client **12** may configure the cache module **26** associated with that client **12** to use only 5% of the storage space and no more than 10% of the processor time. For another example, a user associated with a client **12** may configure the cache module **26** associated with that client **12** to only provide caching services when the system is idle, such as when a screen saver is active or when processor usage by the user of the client **12** is below a particular threshold. In general, resource limits may be associated with cache module **26** such that cache module **26** is prevented from consuming more than a predetermined amount of the resource. The resources may comprise any of an amount of processor time on processor **20**, an amount of bandwidth on link **13**, an amount of storage space on storage **24**, an amount of memory **22** and other computing resources associated with client **12**. Cache module **26** is further operable to collect statistical information associated with link **13**, browser **30**, client **12**, portion **28**, cache module **26** and other elements in community **15**.

Cache module **26** is further operable to encrypt data stored in cache portion **28**. Cache module **26** may use any suitable symmetric and/or asymmetric encryption system for encrypting data in cache portion **28**. For example, cache module **26** may use public-key/private-key encryption, the U.S. Data Encryption Standard (DES), the Twofish algorithm, the Blowfish algorithm and other suitable encryption systems. Encrypting data stored in cache portion **28** prevents a user associated with client **12** from unrestrictedly accessing and modifying cached content. Encryption also provides privacy as the user of any particular client **12** in community **15** is prevented from viewing the data retrieved by other users in community **15**.

The increasing use of "always-on" Internet connections with large bandwidth capacities allows for the use of a distributed caching system using non-specialized equipment. Note that as used herein, an "always-on" connection is defined as a data connection between a client computer, such as a personal computer, and a network, such as the Internet, which operates without competing with other devices associated with a user of the client computer. In addition, an "always-on" connection as used herein may be off and may cycle between being on and off at unpredictable intervals. Stated another way, an "always-on" connection has the capability to be continuously active without interfering with other devices usable by the user associated with the client computer, but the "always-on" connection is not required to be literally "always-on". For example, a traditional analog modem is typically active only when a user of the modem explicitly activates the analog modem and prevents the use of the phone line by the user of the analog modem for voice communication when the analog modem is active. Typically, an analog modem is not considered as creating an "always-on" connection. For another example, a digital subscriber line (DSL) modem, a cable modem and a local area network (LAN) have the capability to be continuously active without interfering with the use of the phone line or the cable television while active. Typically, a DSL modem, a cable modem and a LAN would be considered as creating "always-on" connection.

The growing availability of "always-on" connections provides the opportunity to utilize the processing power and storage capacity of computers connected via "always-on" connections to the Internet or other networks. Many computers with "always-on" connections utilize only a small portion of their processing power and storage space. For example, many DSL and cable modem connected personal computers remain completely unused during the day when their owner is at work and this unused processing and storage capacity may be used to form a distributed caching system. In addition, even when being used, many computers

with "always-on" connections may have significant unused storage and processing capability. System **10** provides the capability to harness the unused processing power and storage capacity of standard computers to create a distributed caching system.

In return for the use of processing power and storage capacity on computers to form the distributed caching system, a provider of system **10** may provide incentives to the users of clients **12**. For example, an ISP, such as **14**, may deploy cache modules **26** to subscribers of the ISP in return for a lower subscription cost. For another example, an ISP could use system **10** as part of a value-added service of faster internet service.

Each cache module **26** is further operable to generate a cache status message **27**. Cache status message **27** comprises a indication of whether a particular cache module **26** is active or inactive. More specifically, cache status message **27** includes a "cache on" or a "cache off" indication. The "cache on" indication indicates that the associated cache module **26** has gone from an inactive to an active state and is requesting to join community **15**. The "cache off" indication indicates that the associated cache module **26** is going from an active to an inactive state and is requesting removal from community **15**. Cache status message **27** may also include an "active" indication. The active indication indicates that the associated cache module **26** is currently active, and caching content and handling requests for content from browsers **30**. The active indication may operate as a heartbeat indicating that the associated cache module **26** is alive.

Each cache module **26** further comprises a distinct location table **29**. Location table **29** comprises one or more indications of the location within community **15** of cached content. More specifically, location table **29** indicates which client **12** is responsible for caching which content. In general, table **29** may use any suitable indication for indicating which clients **12** to cache content at, such as IP addresses, domain names, portions of URLs or a hash value based on a content request from browser **30**.

Cache modules **26** may be further operable to provide increased anonymity while surfing to users of clients **12**. More specifically, cache module **26** may remove specific lines in HTTP requests, such as requests **32**, sent to retrieve web pages from origin server **19**. This feature assists in preventing the sending of certain personal information to origin servers **19**. For example, browser **30** typically requests a web page from origin server **19** by generating a request **32**. Request **32** may include a header portion that may include information about client **12** and the associated user, such as an email address and what web sites that the user has visited. Cache module **26** may intercept request **32** and remove parts of the header portion which are not required to retrieve web pages. In one embodiment, cache module **26** comprises a Microsoft Windows application. The cache module **26** is operable to collect statistics such as the number of cache hits, the number of requests received by cache module **26**, the hit percentage, the amount of data cached, the maximum capacity of cache portion **28**, and how long cache module **26** has been active. Cache module **26** may also allow a user to configure various aspects of cache module **26**, such as the size of cache portion **28**, enable or disable anonymous surfing and purge cache portion **28**.

Browser **30** executes on client **12** and comprises any suitable Hypertext Transport Protocol (HTTP) client. In the disclosed embodiment, browser **30** comprises a web browser such as Internet Explorer® by Microsoft Corp. of Redmond, Washington, or Netscape Communicator by Netscape Communications Corp. Browser **30** is held in memory **22** and executed by processor **20**. Browser **30** transmits and receives data over link **13**. Each browser **30** is operable to generate one or more content requests **32**. Each request **32** is generated by a particular browser **30** in response to a desire for information by a user (not shown) associated with that particular browser **30**. Each request **32** comprises a request for one item of content from origin server **19**. In the disclosed embodiment, request **32** comprises an HTTP request. The HTTP request comprises a Uniform Resource Locator (URL) portion identifying the requested item of content and a header portion containing further information about the HTTP request. The requested item of content may further comprise multiple elements of content, for example, a web page with multiple graphic elements therein, but the request is directed to retrieving the single content item associated with the URL while the remaining elements of content in the main content item are retrieved as a function of the single content item.

Content comprises static information and dynamic information. Static information typically does not change once the content has been created or does not change except slowly over time. Stated another way, static information comprises content that, when created, is expected to remain the same for an indeterminate amount of time. Static information may comprise, for example, text files and preexisting Hypertext Markup Language (HTML) web pages. For example, a restaurant menu is the same regardless of the user who is accessing the menu because the menu is the same for all people. However, as chefs at the restaurant change over time, the menu may also change.

In contrast, dynamic information comprises content that is expected and designed to change. The dynamic content may change based on the data and criteria used for generating the dynamic content, such as a search result page. For example, a search for "mountain lions" will dynamically generate a search result page listing items related to "mountain lions." The dynamic content may also change based on information associated with the page, for example, a page in an online product catalog may change in response to pricing and availability changes, and dynamic content may further change for other suitable reasons. Dynamic information may comprise, for example, the output of Common Gateway Interface (CGI) scripts, search engine results, the output of Structured Query Language (SQL) searches and the SQL search itself, JavaScript, the output of JavaScript scripts, ActiveX controls and the output of ActiveX controls, Active Server Pages, applets and any other suitable dynamic content generation system. Often, the dynamic information will be expressed as a single HTML web page, but the information within the HTML web page has been generated dynamically based on some suitable criteria. For example, the result of a search using a search engine on the Internet returns different information based on the search terms provided by the user. The search results may also depend on one or more attributes associated with the search request, such as geographic location or the current date. Continuing the search engine example, when the user is searching for local businesses, the search results will vary not only on the search terms provided by the user, but also what "local" means with respect to the search request. For another example, a user searching for information about current events will want results that are tailored to the user's search terms, the user's location, and the current date. For yet another example, an online book retailer may provide price information and consumer reviews of books available from the online book retailer. The price information for a particular book may change unexpectedly in response to a sale or a sudden

interest in that particular book. Also, the web page listing the book and the consumer reviews changes in response to a consumer entering a new review of the book.

Referring again to FIG. **1**, communications link **13** comprises a data communications link operable to communicate data between clients **12** and ISP **14**. In the disclosed embodiment, link **13** comprises an "always-on" type connection, such as digital subscriber line (DSL) or a cable modem.

ISP **14** comprises a point of presence on network **16** for communicating data from clients **12** to remote locations. ISP **14** may also define the boundary of community **15**.

Community **15** comprises a plurality of clients **12** at whom content items retrieved by browsers **30** may be cached in cache portions **28**. Community **15** represents a group of clients **12** which cooperate to form a distributed caching system using cache module **26** and portion **28**. Requests by browsers **30** within community **15** for content cached within community **15** do not need to be propagated over network **16** since the requested content is available within community **15**.

Network **16** comprises any suitable data transport network, such as a Transmission Control Protocol/Internet Protocol (TCP/IP) network, an asynchronous transfer mode (ATM) network, an Ethernet network, or a frame-relay network. In the disclosed embodiment, network **16** comprises the Internet.

Community **18** represents an exemplary cache community based around a corporate Intranet. The distributed caching capabilities of system **10** are not limited to home computers. A corporation may utilize cache modules **26** in order to improve web performance for the corporation's employees and/or to decrease the amount of Internet connectivity the corporation purchases. The 10 megabit, 100 megabit, gigabit and faster LAN technologies used by corporations are well suited to the distributed cache of system **10**. LAN technologies may be considered as providing an "always-on" connection within the scope of the present invention. Other collections of computers may also form cache communities, communities **15** and **18** represent two examples of possible cache communities.

Community **18** may comprise a corporate intranet having a communications interface **50**, a LAN **52** and a plurality of intranet clients **54**. Interface **50** comprises a communication interface between LAN **52** and Internet **16**. For example, interface **50** may comprise a firewall, a router or other suitable communications interfaces. Interface **50** may also define the boundary of community **18**. LAN **52** comprises any suitable local area network, such as Ethernet, ATM, or TCP/IP. Intranet clients **54** are similar to ISP clients **12** except that clients **54** are members of an Intranet. Community **18** operates similarly to community **15**, except as otherwise noted.

Origin server **19** communicates data over network **16**. Origin server **19** comprises any suitable hardware and/or software executing on a computer for receiving and responding to requests **32**. Origin server **19** may comprise a single computer executing software or may comprise a plurality of computers each executing software. In the disclosed embodiment, origin server **19** comprises an HTTP server which may also be known as a web server. Origin server **19** may additionally support other protocols such as the file transfer protocol (FTP). Origin server **19** retrieves information from one or more data sources (not shown), such as a storage device coupled to server **19** or other origin servers, in response to requests **32**. Origin server **19** is operable to retrieve static content, such as prewritten text files, images, and web pages, from the data source in response to requests **32**. Origin server **19** is also operable to generate new, dynamic content, for example, by dynamically creating web pages based on content stored in the data source in response to requests **32**. For example, origin server **19** may generate a new web page using a common gateway interface (CGI) script, generate a new web page from the result of a structured query language (SQL) request and perform other suitable content generation functions. Origin server **19** may also be operable to generate executable software, such as applications and applets, in response to requests for data. For example, origin server **19** may generate a Java applet in response to an appropriate request **32**.

In operation, browser **30** generates request **32** for content. Operation of system **10** will be described with respected to cache community **15**, however, it should be noted that cache community **18** operates similarly using clients **54**. Cache module **26** then intercepts request **32** before request **32** is communicated to network **16**. Cache module **26** examines request **32** to determine whether the requested content is available in community **15**. If the requested content is available in community **15**, cache module **26** retrieves the requested content from the appropriate storage portion **28** within community **15** and returns the requested information to the browser **30** which requested it. If the requested content is not available within community **15**, then cache module **26** forwards request **32** over link **13** to ISP **14** for normal handling. Similarly, a request **32** generated by a browser on a client **54** is intercepted by cache module **26** to determine whether the requested content is available within community **18**.

Cache module **26** may be configured to control the amount of processor power, storage space and bandwidth of a particular client **12** used by community **15**. The client-by-client control of usage available to clients **12** allows for individual tailoring of community **15** to particular clients **12**. The client-by-client control of usage also allows for different incentive plans for subscribers if ISP **14**. For example, a subscriber to ISP **14** may have a second computer separate from the computer normally used by the subscriber. The subscriber with two computers could dedicate a large percentage of processor **20** and storage **24** to community **15** in exchange for ISP **14** providing a second IP address for the second computer over a DSL type link **13** for free. For another example, a subscriber of ISP **14** may dedicate increasing proportions of the bandwidth available over link **13** in return for decreasing subscription costs, such as 40% for a 40% discount or 50% for a 50% discount. Community **18**, representing a corporate intranet, may allow for centralized control of the percentage of the processing power, storage and bandwidth used by community **18**, such as by a corporate information technology (IT) department.

In one embodiment, cache module **26** may cache content using a conservative mode or an aggressive mode. When in the conservative mode, cache module **26** caches content received by browser **30** which is marked as cacheable. When in the aggressive mode, cache module **26** caches all content unless the content has been explicitly marked as non-cacheable. In general, by caching all content, unless the content is listed as non-cacheable, more content may be cached in comparison to conservative mode caching. Cache modules **26** using aggressive mode caching may further include communicating with a data center. More specifically, cache module **26** may communicate with the data center to inform the data center of data cached by cache module **26**.

Aggressive mode caching may use a content expiration protocol to avoid providing expired, but cached, content. The data expiration protocol may use data expiration com-

mands to inform cache modules **24** that data at an origin server **19** has changed. Alternatively, a single cache module within a community **15**, such as the master node discussed below, may communicate with the data center. By informing the data center of data cached within community **15**, the data center can send data expiration commands to community **15** so that cache modules **24** can mark cached content as expired.

The data expiration command comprises any suitable message for expiring data stored by cache module **24**. In one embodiment, the data expiration command comprises an Internet Cache Synchronization Protocol (ICSP) message. The ICSP message may expire any of a single web page, a plurality of web pages at a single web site, a plurality of web pages at a plurality of web sites, a plurality of sites within a single domain and one or more specific objects on a web page, such as an image. For example, the ICSP message may expire a single web page such as http://www.domain.com/web_page.html. The ICSP message may expire a plurality of web_pages at a single web site such as http://www.domain-.com/web_pages/* which would expire all web pages in the "web_pages" directory at the site www.domain.com. The ICSP message may expire a plurality of pages at a plurality of web sites such as http://*.domain.com/web_pages/* which would expire all web pages in the "web_pages" directory of each site in "domain.com". The ICSP message may expire a plurality of web sites such as http://*.domain-.com which would expire all web pages on all sites in "domain.com". For another example, a single active server page (ASP) may result in many individual cached pages. A single ASP page may generate large numbers of individual cached pages because a single ASP page can dynamically create multiple different specific pages, such as http://www.domain.com/product_description.asp?category=_&product=_may specify an ASP page that generates many specific product description pages based on a specified category and product. To cache module **24**, product_description.asp?category=5&product=7, product_description.asp?category=5&product=6 and product_description.asp?category=3&product=7 may specify three different cached pages. By expiring product_description.asp?category=*&product=7 ALL categories of product seven are expired, while not affecting product six related pages even though all of the pages were generated from a single ASP page.

In general, the data center may generate the data expiration command in response to a change in the content at origin server **19**. The data center may also generate the data expiration command in response to the elapsing of a predetermined time period. ICSP supports the synchronization of cached content in community **15** with updated content available at origin server **19**. Further details of ICSP are described in the patent application entitled "Method and Apparatus for Content Synchronization" by inventors Keith A. Lowery, et al., filed on Jun. 8, 2000, Ser. No. 09/590,760, which is incorporated herein by reference.

In addition, cache module **26** may provide a guaranteed click delivery capability. The guaranteed click delivery capability comprises the capability to regularly check whether a particular web page is available and to retrieve the web page when the web page becomes available. For example, a user of client **12** may attempt to retrieve a particular web page. The server providing that web page may be currently overloaded and unable to provide the requested web page. Cache module **26** may detect that the server is too busy, such as when the web site returns a "server too busy" error message, and then automatically attempt to retrieve the request web page from the busy server. For example, a busy server may comprise a server which is currently processing substantially all the requests **32** which the server is capable of handling. For another example, a busy server may comprise a server which is providing content and using substantially all of the bandwidth available to the server. In general, a busy server may comprise a server which is incapable of processing more requests **32** at a given time for one or more reasons. Cache module **26** may then display the retrieved web page in browser **30** or may abandon the attempt to retrieve the web page after a predetermined period of time has elapsed without successfully retrieving the requested web page. Cache module **26** may also ask the user whether the user wants cache module **26** to attempt to retrieve the requested web page from the busy server. Typically, cache module **26** would attempt to retrieve the requested web page from the busy server while the user retrieves and views other web pages from other origin servers **19**. Stated another way, cache module **26** would attempt to retrieve the requested web page in the background while the user may also be performing other tasks.

Yet another capability of cache module **26** is the ability to provide a screen saver to a user associated with client **12**. The screen saver displays a graphical representation of the user's response time to one or more origin servers **19**. For example, the response time between client **12** and a particular web site may be displayed in a graphical manner. More specifically, the screen saver displays a solar system-like graph with client **12** generally in the center and the distance between client **12** and other web sites displayed based on the round-trip ping time between client **12** and the other web sites.

FIG. **2** is a block diagram illustrating an exemplary community cache **100**. Community **100** comprises a first client **102**, a second client **104**, a third client **106** and an ISP **108**. In the exemplary embodiment of FIG. **2**, clients **102**, **104** and **106** represent specific examples of clients **12** of FIG. **1**. Clients **102**, **104** and **106** communicate with ISP **108** over respective communication links **110**. Client **102** comprises a browser **120**, storage **122**, and a cache module **124**. Client **104** comprises a browser **140**, storage **142**, and a cache module **144**. Client **106** comprises a browser **160**, storage **162**, and a cache module **164**.

Browsers **120**, **140** and **160** represent distinct examples of browsers **30** of FIG. **1**. Each of storage **122**, **142**, and **162** respectively support cache portions **126**, **146**, and **166**. Storage **122**, **142**, and **162** represent distinct examples of storage **24** of FIG. **1**. Cache portions **126**, **146**, and **166** represent distinct examples of cache portions **28** of FIG. **1**. Cache modules **124**, **144**, and **164** support respective location tables **128**, **148** and **168**. Each of cache module **124**, **144** and **164** are operable to generate respective cache status messages **130**, **150** and **170**. Cache modules **124**, **144**, and **164** represent distinct examples of cache modules **26** of FIG. **1**. Location tables **128**, **148** and **168** represent distinct examples of location table **29** of FIG. **1**. Cache status messages **130**, **150** and **170** represent distinct examples of cache status message **27** of FIG. **1**.

Location tables **128**, **148**, and **168** respectively comprise one or more indications of which client **102**, **104** or **106** to cache content in response to requests **32** from browsers **120**, **140**, and **160**. For example, location table **128** may indicate that content identified by URLs having a domain name beginning with A–D is cached at client **102**, while domain names E–H are cached at client **104** and domain names H–Z

are cached at client **106**. For another example, location tables **128**, **148** and **168** may indicate particular ranges of IP addresses to be cached at particular clients **102**, **104** and **106**. In general, tables **128**, **148** and **168** may use any suitable indication for indicating which clients **102**, **104** and **106** to cache content at, such as IP addresses, domain names, portions of URLs or a hash value based on request **32**.

Cache status messages **130**, **150** and **170** each comprise a message respectively generated by modules **124**, **144** and **164** to indicate to other modules **124**, **144** and **164** that the generating module is activating or deactivating its caching functionality. For example, when cache module **124** at client **102** is activated it may generate a cache status message **130** indicating caching is active at client **102**.

Communications link **110** comprises any suitable data communications system. In the exemplary embodiment of FIG. **3**, communications link **110** comprises a "always-on" link, such as DSL or a cable modem, similar to link **13** in FIG. **1**. ISP **108** comprises an Internet service provider operable to communicate data between clients **102**, **104** and **106** and network **16** similar to ISP **14** of FIG. **1**.

In operation, in one embodiment, community **100** may be formed by dynamically seeking out other active instances of cache module **26**. Then, based on a set of performance heuristics, clients **12** are bonded together under favorable conditions. Cache module **26** may use dynamic affiliation algorithms to build and manage communities **100**. More specifically, on startup, cache module **26** may communicate with a remote directory provider for assistance in finding other cache modules **26** with which to form a community **100**. Using the assistance from the remote directory provider, the client may attempt to contact and possibly join a currently existing community **100**. If no communities **100** are found or found communities **100** do not allow cache module **26**, then cache module **26** may attempt to start its own cache community. Alternatively, if no remote directory is available, cache module **26** searches for communities **100** itself. For example, cache module **26** may send a broadcast and/or a multicast message looking for communities **100**.

Each community **100** includes a master node and, optionally, one or more normal nodes. A master node comprises a cache module **26** on a particular client **12** which is responsible for supervising the addition and departure of clients from community **100**. The master node receives data associated with the addition of a client **12** to community **100** and the departure of a client **12** from community **100** and communicates the data to the other members of community **100**. Any cache module **26** may function as the master node. Any suitable method for electing the master node may be used by cache modules **26**. For example, a cache module **26** which has been activated the longest may be selected as the master, with ties being resolved randomly. The departure of the master node causes the election of a new master node by the remaining members of community **100**.

Community **100** handles the graceful and non-graceful departure of clients **12** from community **100**. A graceful departure comprises an intentional departure of a client **12** from community **100**. For example, a graceful departure may occur when a user deactivates cache module **26**. A non-graceful departure comprises an unexpected departure of a client **12** from community **100**. For example, a non-graceful departure may occur when a client **12** suddenly crashes and ceases operation. When an active cache module **26** shuts down, for example, the cache module **26** requests to leave community **100** and the request circulates through the remaining community members. The remaining community members would then discontinue forwarding requests to that client **12**. In a non-graceful scenario, a managing peer (known as a Master) watches for dead peers and notifies the rest of a community if this condition is detected.

Similarly, the managing peer may depart gracefully or non-gracefully. A graceful departure of the managing peer comprises the managing peer informing community **100** that the managing peer is leaving community **100**. An election is then held by the remaining members of the peer to select the new managing peer. When a non-graceful departure occurs, such as when the managing peer crashes, a cache module **26** may detect that the managing peer is not responding and call an election to create a new managing peer. In general, any suitable techniques may be used to handle the arrival and departure of cache modules **26** from community **100**, and to create and maintain the managing peer.

FIG. **3** is a flowchart illustrating a method for community caching. For increased clarity, the operation of the exemplary cache community **100** of FIG. **2** will be described in association with FIG. **3**. The method begins at step **200** where browser **120** generates a request **32** for content. The URL portion of the request **32** identifies the requested content as "http://www.server.com/jobs/listing.html". Next, at step **202** module **124** intercepts request **32** generated by browser **120**. Then, at step **204**, cache module **128** determines the URL associated with request **32**.

Proceeding to step **206**, cache module **124** determines the location where the content associated with the URL associated with request **32** would be cached. More specifically, cache module **124** determines which of the storage portions **126**, **146** or **166**, would store the requested content based on information in location table **128**. In the example of FIG. **2**, location table **128** indicates that the domain "server.com" would be cached at client **106**. Next, at step **208**, cache module **124** checks storage portion **166** for the requested content and at decisional step **210**, determines whether the requested content has been cached. More specifically, cache module **124** queries cache module **164** to determine whether the content associated with the URL in request **32** has been cached in portion **166** at client **106**. If cache module **164** replies that the requested content is cached in portion **166**, then the YES branch of decisional step **210** is followed to step **212** where the requested content is retrieved from storage portion **166** and, at step **214**, is displayed at browser **120**.

If the requested content is not cached at portion **166**, indicating that the requested content is not available within community **100**, then the NO branch of decisional step **210** leads to step **216**. At step **216**, the requested content is retrieved from origin server **19** since the requested content is not cached within community **100**. The requested content is then displayed on browser **120** at step **214** and, at step **218**, a copy of the requested content is communicated to cache module **164** for storage in portion **166**. Then, at step **220**, the retrieved content is stored in portion **166** by module **164**.

Returning to the start state of FIG. **3**, steps **230** and **232** are typically performed in parallel to the processing previously described for FIG. **3**. At step **230**, modules **26** generate cache status messages **27** with an "active" indication and broadcast messages **27** to clients **12** (or **54**) in community **15** (or **18**). Message **27** with an "active" indication may be used by modules **26** to indicate that the associated cache module **26** is alive. Then, at step **232**, modules **26** wait a predetermined period of time before broadcasting another message **27**. Modules **26** wait so as not to flood community **15** (or **18**) with messages **27** and waste bandwidth. In one embodiment, the predetermined wait period is five seconds. Modules **26**

which fail to broadcast the "active" indication may be removed from the cache community because the associated cache module **26** may have failed, other suitable actions may be taken in response to a failure to broadcast the "active" indication.

The method described by FIG. **3** is shown to end after steps **214** and **220**, this indicates that the processing for the request generated in step **200** has been completed. The method described by FIG. **3** is expected to be repeated a plurality of times within the scope of the invention.

FIG. **4** is a flowchart illustrating a method for generating a cache community, such as **15**, **18** or **100**. For increased clarity, the exemplary community **100** of FIG. **2** is discussed in association with FIG. **4**. The method begins at step **300** where cache module **144** is initiated. Initiating module **144** may comprise activating module **144**, such as by activating client **104**. Next, at step **302**, cache module **144** generates cache status message **150** with a "cache on" indication and broadcasts cache status message **150** to clients **102** and **106** in community **100**. Message **150** may be broadcast using any suitable method supported by ISP **108** and links **110**. For example, a TCP/IP broadcast or multicast message may be used to reach clients **102** and **106** associated with ISP **108**. For another example and referring back to FIG. **1**, community **18** may use an Ethernet broadcast message to reach all clients **54** in community **18**.

At step **303**, a primary distribution of portions of content to be cached ("cache shares") is negotiated between modules **124**, **144** and **164** within community **100**. The primary distribution of cache shares may be determined by splitting up URLs alphabetically by domain name, by generating a unique hash value for each URL and distributing the hash values among clients **102**, **104** and **106**, by IP address range or by any other suitable method or combination of methods. The distribution method to use may be configured for modules **124**, **144** and **164**, such as by an administrator. The actual primary distribution of cache shares may comprise a simple fractional split between the members of a cache community, such as **15**, **18** (FIG. **1**) or **100**, or may also consider other elements. For example, the actual distribution may consider the speed of links **13** (FIG. **1**) or **110**, the processing power of processors **20** (FIG. **1**) and/or the size of portions **28** (FIG. **1**), **126**, **146** and **166**. Historical and statistical information may be considered when determining the primary distribution. For example, a particular client, such as **12** or **54**, which regularly reboots may receive a smaller a cache share than a client which has a historically high up time.

With respect to the example in FIG. **2**, modules **124** and **164** may have already negotiated cache shares based on an alphabetical distribution where URLs having domains starting with A–M are cached at client **102** and domains starting with N–Z are cached at client **106**. Modules **124**, **144** and **164** may now renegotiate cache shares such that A–H domains are cached at client **102**, I–P domains are cached at client **104** and Q–Z domains are cached at client **106** using a simple fractional alphabetic distribution.

Proceeding to step **304**, a secondary distribution may optionally be negotiated. For increased reliability, cache modules **26** may be configured to negotiate the secondary distribution. The secondary distribution represents a distribution of cache shares to decrease the impact of the failure of clients **12**. The secondary distribution may be determined using any of the information and methods used in determining the primary distribution in step **303**. For example, cache modules may negotiate a secondary distribution of cache shares such that multiple clients **12** are caching the same content so that if a particular client **12** fails, a backup of the cached content exists within the community.

FIG. **5** is a diagram illustrating an exemplary distribution of cache shares. For increased clarity, FIG. **5** is discussed in association with FIGS. **2** and **4**. The primary distribution of cache shares described previously in associated with step **303** is shown in FIG. **5** as "primary" where domains A–H were allocated to client **102**, domains I–P were allocated to client **104** and domains Q–Z were allocated to client **106**. An overlapping secondary distribution may be used to allocate portions of the primary distribution to other clients for increased reliability. For example, an overlapping secondary distribution may allocate I–L and Q–U to client **102**, A–D and V–Z to client **104** and E–H and M–P to client **106** as shown in the "secondary" line of FIG. **5**. Alternatively, the secondary distribution may simply rotate cache shares, such as having A–H allocated to client **106**, I–P allocated to client **102** and Q–Z allocated to client **104** and is shown as "alt. secondary" in FIG. **5**. Any suitable secondary distribution may be used depending on the configuration of cache modules **26** (FIG. **1**), **124**, **144** and **164** (FIG. **2**). It should be noted that the cache shares shown in FIG. **5** are provided for increased clarity and are merely exemplary, various other primary and secondary distributions of cache shares are usable within the scope of the invention.

In addition, tertiary and further distributions may be negotiated to provide multiple layers of backup within cache communities **15**, **18** and **100** at step **304**. The determination of how many layers of backup to provide may be based on the reliability of clients **12**, **54** (FIG. **1**), **102**, **104** and **106** (FIG. **2**), the delay associated with checking backup cache modules **26** versus the delay associated with retrieving the data from origin server **19**, bandwidth considerations with community **15**, **18** or **100** and other criteria.

Then, at step **305**, once the primary, secondary and other distributions are determined, location tables **128**, **148** and **168** are updated to reflect which client **102**, **104** and **106** is caching which content.

Then, at step **306**, any redistribution of previously cached content for supporting a newly negotiated distribution occurs. Redistribution may comprise actually copying cached content from one client, such as **12** or **54** (FIG. **1**), to another, or removing or expiring content no longer cached at a particular server. Simply removing the cached items may cause a request that could have been satisfied by the community cache to be forwarded to the origin server, but avoids the bandwidth usage associated with copying. The decision whether to copy or remove may be configured at modules **124**, **144** and **164** and may consider bandwidth usage issues, the amount of content to copy and other criteria. For example, modules **124**, **144** and **164** may be configured to perform a copy of cached content with the content comprises less than five megabytes of data and to perform a removal when more than five megabytes of content are involved.

Continuing the example of FIG. **2**, the distribution has changed such that domains starting with I–M, previously cached at client **102**, and domains starting with N–P, previously cached at client **106**, are now being cached at client **104**. Thus, cached content may be moved or copied from clients **102** and **106** to portion **146** at client **104**.

At state **308**, modules **124**, **144** and **164** cache content in portions **126**, **146** and **166** and respond to requests **32** as described in FIG. **3**. At state **310**, modules **124**, **144** and **164** may also collect statistical data associated with clients **102**, **104** and **106**. Statistical data may comprise processor and/or bandwidth usage, such as with respect to the time of day,

time between crashes or reboots, web surfing habits and any other suitable statistical information associated with clients **102**, **104** and **106**. The statistical data may be stored by modules **124**, **144** and **164** for use in negotiating distributions and may also be communicated to a remote site. The statistical data may be communicated to the remote site for storage and use for marketing and advertising purposes. For example, a corporation may use the statistical information to learn about the surfing habits of its employees.

Then, at step **312**, cache module **168** begins to deactivate and broadcasts cache status message **170** indicating "cache off" to clients **102** and **104**. Modules **124** and **144** receiving cache status message **170** indicating a cache shutdown respond by renegotiating the cache distribution between the remaining active modules **124** and **144**. More specifically, modules **124** and **144** may redistribute cached content such that domains starting with A–M are cached at client **102**, while domains starting with N–Z are cached at client **104**. Location tables **128** and **148** are updated to indicate the new content distribution. Next, at step **314**, the module **164** clears portion **166**. Depending on the configuration of modules **164**, clearing portion **166** may include communicating cached content to the remaining clients **102** and **104**.

System **10** provides the capability to create a distributed caching system using standard personal computers and the cache module software. Previous caching systems have typically been limited to caching at a single computer, such as the caching performed by web browsers, or have concentrated caching efforts at a single firewall or proxy server. The increasing adoption of "always-on", high-bandwidth Internet connections, such as DSL modems, cable modems and LANs, allows for system **10** to group standard personal computers to form a cache community in order to decrease response time for content requests. In addition, since the cache modules use the existing storage capacity of the client computers, additional hardware does not need to be purchased for the cache community. Also, corporations may benefit from forming community caches over their corporate intranet LANs in order to decrease usage of expensive wide-area-network connections, such as T-1 lines, to the Internet.

The use of community caches also provides benefits to Internet Service Providers (ISPs) by decreasing the amount of data transferred between the ISP and the Internet. Since the community caching takes place entirely on the subscriber side of the ISP's infrastructure, an increased number of subscribers can be supported by a given amount of Internet connection bandwidth. For example, if 25% of the content requests from subscribers are handled by the community cache, an approximately 25% decrease in the use of the ISP's public network connection is realized.

FIG. **6** is a block diagram illustrating a dynamic caching system **400** comprising one embodiment of system **10**. System **400** comprises a cache community **402**, a client **404**, a cache server **406**, Internet **16**, and origin server **19**. Dynamic distributed caching system **400** provides the capability for allowing members to dynamically join and leave cache community **402** while continuing to provide a distributed caching system.

Community **402** comprises one or more peers **413**. A peer **413** comprises a computer operable to execute logic and/or an application. Peers **413** further comprise a master **410** and a member **412**. Community **402**, similar to community **15**, represents a group of peers **413** which cooperate to form a distributed caching system. Each community **402** includes one or more peers **413**. More specifically, each community **402** includes one master **410** and zero or more members **412**. A dynamic caching application **478** at each peer **413** provides functionality to support the distributed caching system **10**.

Master **410** comprises a processor **420** and computer readable storage **422**. Master **410** may be adapted to execute any of the well known MS-DOS, PC-DOS, OS/2, UNIX, Linux, MAC-OS, mainframe, minicomputer, Windows operating systems or other operating systems. Processor **420** comprises any suitable general purpose or specialized electronic or other processing device, such as a central processing unit (CPU), operable to communicate with storage **422** and execute applications stored in storage **422**. Processor **420** may comprise multiple processors. Storage **422** comprises any suitable combination of transient and/or persistent memory operable to store member list **426** and dynamic caching application **428**, and to communicate with processor **420**. For example, storage **422** may comprise any suitable combination of optical and/or magnetic storage, such as a floppy disk drive, a hard disk drive, a CD-ROM drive or a CD-RW drive. Storage **422** may also represent multiple computer readable storage devices in any combination.

Master **410** is operable to generate an allow message **424** and maintain a peer list **426**. Allow message **424** comprises a data message sent to client **404** to inform client **404** that client **404** is being allowed to join community **402**. Allow message **424** may also be sent to client **404** to inform client **404** that entry to community **402** is denied.

Peer list **426** comprises a list of peers **413** in community **402**. For example, peer list **426** may compromise a list of the Internet protocol (IP) addresses associated with peers **413**.

Dynamic caching application **428** comprises software and/or hardware operable to support caching of data and other content within community **402**. Application **428** is described in more detail in FIG. **7**.

Member **412** comprises a processor **430**, computer readable storage **432** and dynamic caching application **428**. Processor **430** comprises any suitable general purpose or specialized electronic processing device, such as a CPU, operable to communicate with storage **432**. Processor **430** may comprise multiple processors. Storage **432** comprises any suitable combination of transient and/or persistent memory operable to store peer list **426** and dynamic caching application **428**, and to communicate with processor **430**. For example, storage **432** may comprise any suitable combination of optical or magnetic computer readable storage mediums such as a floppy disk drive, a hard disk drive, a CD-ROM drive, an optical drive, or a CD-RW drive. Storage **432** may also represent multiple computer readable storage devices in any combination.

Community **402** is comprised of a master **410** and zero or more members **412**. Members **412** and master **410** operate to support the caching of content within community **402**. More specifically, both master **410** and members **402** provide for a distributed caching system within community **402**. In addition to the functionality provided by members **412**, master **410** is further responsible for providing administrative support to community **402**. In general, a member **412** is capable of being a master **410** as necessary and a member **412** may be designated as a master **410** at any time. Dynamic cache application **428**, which executes on both master **410** and members **412**, is typically fully capable of providing the functionality associated with being either a master **410** or a member **412**.

Client **404** comprises a processor **440**, computer readable storage **442** and dynamic caching cache application **428**. Client **404** comprises a computer executing dynamic cache application **428** which has not yet joined a community **402**.

Processor **440** comprises any suitable general purpose or specialized electronic processing device, such as a CPU, operable to communicate with storage **442**. Processor **440** may comprise multiple processors. Storage **442** comprises any suitable combination of transient and/or persistent memory operable to store member list **426** and dynamic caching application **428**, and to communicate with processor **440**. Storage **442** may further comprise any suitable combination of optical, magnetic or other computer readable storage mediums such as a floppy disk drive, a hard disk drive, a CD-ROM drive, an optical drive or a CD-RW drive. Storage **442** may also represent multiple computer readable storage devices in any combination.

Dynamic cache application **428** is operable to generate a community request **450** and a joined request **452**. Community request **450** comprises a data message generated by dynamic cache application **428**. Community request **450** requests the network location of communities **402** which client **404** may attempt to join. Community request **450** is communicating to cache server **406**.

Join request **452** comprises a data message indicating that client **404** wishes to join a particular community **402**. Join request **452** is communicated to a community **402**.

Origin server **19** is further operable to provide content **460**. Content **460** comprises static and/or dynamically generated information. In one embodiment, content **460** comprises one or more web pages, such as those formatted and linked using the hypertext markup language (HTML). For example, content **460** may comprise audio data, video data, text data, animations, applications, applets and other suitable content retrievable using the hypertext transport protocol (HTTP).

Cache server **406** comprises a processor **470**, computer readable storage **472**, an administration module **474**, a community list **476**, an expiration module **478**, and is operable to handle an expiration message **480**. Cache server **406** comprises one or more servers operable to provide content expiration and community tracking capabilities to system **400**. More specifically, cache server **460** maintains a list of communities **402** which a client **404** may attempt to join. Processor **470** comprises any suitable general purpose or specialized electronic processing device, such as a CPU, operable to communicate with storage **472**. Processor **470** may comprise multiple processors. Storage **472** comprises any suitable combination of transient and/or persistent memory operable to store member list **426** and dynamic caching application **428**, and to communicate with processor **470**. Storage **472** may further comprise any suitable combination of optical, magnetic or other computer readable storage mediums such as a floppy disk drive, a hard disk drive, a CD-ROM drive, an optical drive, or a CD-RW drive. Storage **472** may also represent multiple computer readable storage devices in any combination.

Administration module **474** comprises software and/or hardware operable to generate and maintain community list **476**. Administration module **474** is further operable to respond to community request **450** from client **404**.

Community list **476** comprises a list of communities **402** known to cache server **406** that client **404** may attempt to join. In one embodiment, cache server **406** tracks communities **402** which are close to cache server **406** in terms of network distance. Typically, on a computer network, geographic location is not necessarily a good indicator of response time between that two nodes on a network. The latency between two network nodes is often a better indication of performance. The latency represents the sum of the time the packet has spent waiting to be forwarded out of a particular network node plus the transit time over the link connecting one network node to another node. The latency between two network locations may be used to determine the network distance between the two network locations. Other suitable indications may also be used to determine network distance, such as the number of hops or nodes between two nodes. Administration module **474** may impose an upper and/or a lower limit on the network distance from cache server **406** of communities **402** that may be listed in community list **476**.

Expiration module **478** comprises software and/or hardware operable to receive and communicate data expiration messages **480** associated with content **460**. Expiration message **480** comprises an indication that content **460** at origin server **19** has changed and that cached copies of content **460** should be marked as stale or out-of-date. Expiration module **478** operates to communicate expiration message **480** to communities **402** listed in community list **476**.

In operation, dynamic cache application **428** is initiated at client **404**. Dynamic cache application **428** then generates a community request **450**. Community request **450** is a request for a list of communities **402** that client **404** may attempt to join. Community request **450** is communicated over Internet **60** to cache server **406**. In one embodiment, the cache server **406** contacted by dynamic cache application **428** has been previously determined, such as a particular URL. Alternatively, cache server **406** may be specified by a user associated with client **404**. Cache server **406** then receives community request **450** and returns community list **476** to client **404**.

Dynamic cache application **428** then examines community list **476** and selects one of the communities **402** listed in community list **476** to attempt to join. For example, dynamic cache application **428** may select a community **402** which has the lowest latency from client **404**. Alternatively, dynamic cache application **428** may select the first community **402** in list **476** or may select a community **402** randomly from list **476**. In general, dynamic cache application **428** may select a community **402** to attempt to join using any suitable technique.

Once dynamic cache application **428** has selected a particular community **402** to attempt to join, dynamic cache application **420** generates join request **452** and communicates join request **452** to community **402**. More specifically, join request **452** is communicated from client **404** to master **410** in community **402** since master **410** is responsible for handling the addition and removal of members **412** from community **402**.

Master **410** receives join request **452** and determines whether to allow client **404** to become a member **412** of community **402**. Master **410** may use any suitable criteria to determine whether to allow client **404** to join community **402**. For example, master **410** may consider whether the addition of client **404** would exceed the maximum number of members **412** that may be in community **402** and/or whether the round trip transit time for data between client **404** and members **412** of community **402** is within a threshold limit. The round trip transmission time may comprise the total length of time a data package from master **410** or members **412** takes to travel to client **404** and return back to the originating master **410** or member **412**. Master **410** may attempt to keep round trip transmission times within a certain threshold so as to avoid degrading the caching performance of community **402**. If master **410** determines that client **404** is appropriate to be a member of community **402**, then dynamic cache application **428** at master **410** generates allow message **424** and communicates allow mes-

sage **424** to client **404**. When allow message **424** indicates that client **404** may join community **402**, then dynamic cache application **428** also communicates peer list **426** along with allow message **424** to client **404**. If allow message **424** indicates that client **404** is allowed to join community **402**, then client **404** has joined community **402**.

Master **410** then communicates to members **412** that a new member **412** is joining community **402** by communicating an updated peer list **426** which includes client **404**. Dynamic cache application **428** then reallocates the content **460** to be cached by particular members **412** and master **410** so that the newly added client **404** is responsible for some subset of content **460** cached in community **402**.

If master **410** determines that client **404** should not be allowed to join community **402**, then dynamic cache application **428** may generate allow message **424** with the indication that client **404** has been denied entry to community **402**. Alternatively, dynamic cache application **428** may simply ignore join request **452** and allow client **404** to determine that it has rejected from community **402** because no response to join request **452** has been received.

If client **404** receives an allow message **424** indicating denial of entry to community **402** or if client **404** does not receive a response from master **410** within a predetermined period of time, then dynamic cache application **428** will determine that client **404** has been denied entry to community **402**. Dynamic cache application **428** will then select a new community **402** from community list **476** and communicate a join request **452** to the newly selected community **402**.

When cache server **406** determines that content **460** has changed at origin server **19** or is informed that content **460** has changed at origin server **19**, cache server **406** generates expiration message **480**. Expiration message **480** is then sent to each master **410** for each community **402** listed in community list **476**. Master **410** in community **402** then receives expiration message **480** and communicates expiration message **480** to members **412** so that expired, cached content **460** is marked as stale or expired. In one embodiment, expiration message **480** comprises an ICSP message. When expired content **460** is cached in community **402** and a request for the expired content **460** is received, a new copy of content **460** is retrieved from origin server **19**.

FIG. **7** is a block diagram illustrating details of dynamic cache application **428**. Dynamic cache application **428** comprises a cache portion **500**, an expiration portion **502**, and a dynamic affiliation portion **504**.

Cache portion **500** comprises software and/or hardware operable to receive any cached content **460**. Cache portion **500** is responsible for managing and organizing the actual cache content on the computer readable storage, such as storage **422**, **432**, or **442**. Cache portion **500** is further operable to maintain an allocation list **510** when cache application **428** is acting as a master **410**.

Allocation list **510** comprises an indication of which peer **413** caches which content **460**. In one embodiment, allocation list **510** comprises location table **148** and the cache shares previously described. In general, any suitable technique may be used for allocating content **468** among peers **413**. For example, the CARP protocol may be used to manage distribution of content within community **402**.

Expiration portion **502** comprises software and/or hardware operable to receive expiration messages **480** and mark content stored by cache portion **500** as expired. Alternatively, expiration portion **502** may request cache portion **500** to itself mark content as expired. Expiration portion **502** is further operable to propagate expiration message **480** to members **412** in member list **426** when dynamic cache application **428** is operating as a master **410**.

Dynamic affiliation portion **504** comprises software and/or hardware operable to support the dynamic joining and departure of members **412** and master **410** from community **402**. Dynamic affiliation portion **504** is further operable to generate and respond to various messages and requests. In particular, dynamic affiliation portion **504** is responsible for supporting allow message **424**, community request **450**, join request **452**, probe message **456** and add master request **454**. Dynamic affiliation portion **504** is further operable to provide a peer list message **520**, a member status request **522**, a nominate master message **524**, an update master request **526**, a remove peer message **528**, a hold election message **530**, a location request **550**, a location response **554**, a probe message **456** and a remove master request **527**.

Peer list message **520** comprises a data communication to members **412** of community **402** to update peer list **426** to add or remove peers **413** from community **402**. For example, peer list message **520** may include an updated peer list **426** from master **410** for use by members **412**. For another example, peer list message **520** may include an instruction to a member **412** to add or delete a particular peer **413** from peer list **426**.

Member status request **522** comprises data communicated from dynamic affiliation module **504** at master **410** to a member **412** to determine the status of the member **412**. More specifically, member status request **522** is used by master **410** to determine whether a member **412** has unexpectedly left community **402**, such as by crashing. In one embodiment, member status request **522** may comprise a "ping" type request, such as one based on the Internet Control Message Protocol (ICMP) used with the TCP/IP protocol.

Nominate master message **524** comprises data indicating that a particular peer **413** should be designated as master **410**. Update master request **526** comprises data indicating to administration module **474** that community list **476** should be updated to indicate that a particular peer **413** or client **404** has become the master **410** of community **402**.

Remove peer message **528** comprises a data message which indicates from master **410** to members **412** that a particular member **412** is leaving community **402** and should be removed from peer list **426**. Hold election message **530** comprises a data message which indicates that a member **412** has detected the possible departure of master **410** from community **402** in that an election should be held among the remaining members **412** to determine a new master **410**.

Location request **550** comprises a data message which indicates a request for the peer **413** which would cache content **460** requested by request **32**. Location response **554** comprises a data message which indicates an indication of a peer **413** in community **402** which is responsible for caching the requested content. Probe message **456** comprises a data message which indicates a broadcast and/or multicast request to determine the existence of one or more communities **402**.

Remove master request **527** comprises a data message which indicates a request by a master **410** that the cache server **406** remove the community **402** associated with the master **410**. More specifically, remove master request **527** is used by master **410** to indicate the shutdown of a cache community **402**.

FIG. **8** is a flow diagram illustrating a method for retrieving and caching content within community **402**. Initially, a request **32** for requested content **548** is generated by browser **30** in response to, for example, user input. Request **32** is

received at cache portion **500** of dynamic cache application **428** at member **412**A. Cache portion **500** at member **412**A then determines whether the requested content **548** is available in cache portion **500** at member **412**A. In general, content is considered "available" at a particular cache portion **500** when the content is stored and unexpired at the particular cache portion **500**. Content may be considered unavailable if the content is stored, but marked as expired, or when the content is not stored at the cache portion **500**.

If requested content **548** is available at cache portion **500** at member **412**A, then cache portion **500** returns requested content **548** to browser **30**.

If requested content **548** is unavailable at cache portion **500** at member **412**A, then cache portion **500** at member **412**A generates location request **550** which is communicated to cache portion **500** at master **410**. Cache portion **500** at master **410** examines allocation list **510** to determine which peer **413** would cache the requested content **460**.

Cache portion **500** at master **410** determines where the requested content **460** is located and generates a location response **554** which is communicated to member **412**A.

Cache portion **500** at member **412**A then receives location response **554** and forwards request **32** to the appropriate peer **413**. For example, the appropriate peer **413** may be member **412**B. Cache portion **500** at member **412**B receives request **32** and determines whether the requested content **548** is available and current in cache portion **500** at member **412**B. If the requested content **548** is available and current in cache portion **500**, then the requested content **548** is returned by member **412**B to cache portion **500** at member **412**A. Cache portion **500** at member **412**A then provides requested content **548** to browser **30** for display to a user associated with member **412**A.

If requested content **548** is not available at cache portion **500** at member **412**B, then member **412**B forwards request **32** to origin server **19**. Origin server **19** then returns requested content **548** to cache portion **500** at member **412**B. Member **412**B then caches requested content **548** at cache portion **500** and forwards requested content **548** to cache portion **500** at member **412**A. Cache portion **500** at member **412**A then communicates requested content **548** to browser **30** for display to the user.

FIG. **9** is a flow chart illustrating a method for adding a client **404** to community **402**. The method begins at step **600**, where client **404** sends community request **450** to administration module **474**. Next, at step **602**, client **404** determines whether a response has been received from administration module **474**. More specifically, dynamic cache application **428** determines whether a community list **476** has been received from cache server **406**.

If a response is received from administration module **474**, then the YES branch of decisional step **602** leads to step **604**. At step **604**, dynamic cache application **428** examines community list **476**. Then, at step **606**, dynamic cache application **428** attempts to find a community **402** which comprises a "best fit" community for client **404** to join. More specifically, dynamic cache application **428** may evaluate various factors associated with communities **402** to determine which community **402** or community list **476** client **404** should join. For example, dynamic cache application **428** may determine the round trip transmission time between client **404** and community **402**. The round trip transmission time may be useful in determining the overall performance of a community **402** and the performance of a community **402** with respect to client **404**. Proceeding to decisional step **608**, dynamic cache application **428** determines whether any communities **402** meet the best fit criteria. If one or more communities **402** meet the best fit criteria, then the YES branch of decisional step **608** leads to step **610**. At step **610**, dynamic cache application **428** sends join request **452** to one of the communities **402** which matched the best fit criteria in step **608**. The particular community selected by dynamic cache application **428** may comprise the community **402** which most closely met the best fit criteria. Alternatively, dynamic cache application **428** may use other methods for selecting the first community **402** to contact.

Then, at step **612**, dynamic cache application **428** at client **404** determines whether master **410** has accepted client **404** into community **402**. Dynamic cache application **428** at client **404** may determine that master **410** has denied client **404** admittance to community **402** by receiving an allow message **424** indicating that client **404** is not allowed to join community **402** or by receiving no response from master **410** within a predetermined time period. If client **404** receives an allow message **424** indicating that client **404** is allowed to join community **402**, then the YES branch of decisional step **612** leads to step **614**. At step **614**, client **404** is added to community **402** by master **410** and member list **426** is updated by master **410**.

If master **410** denies client **404** entrance to community **402**, then the NO branch of decisional step **612** leads to step **616**. At step **616**, dynamic cache application **428** at client **404** selects the next community **402** to attempt to join. Also, at step **616**, the criteria for the best fit for client **404** may optionally be modified by dynamic cache application **428** so as to change or increase the number of communities **402** that client **404** may join. The method then proceeds to step **608** where communities **402** and community list **476** are evaluated against the best fit criteria which may have changed at step **616**.

Returning to decisional step **608**, if no communities **402** in community list **476** match the best fit criteria for client **404**, then the NO branch of decisional step **608** leads to step **618**. At step **618**, dynamic cache application **428** at client **404** nominates itself as a master **410**. When client **404** nominates itself as a master **410**, client **404** creates a new cache community **402**. Then, at step **620**, add master **454** is sent from client **404** to administration module **474** at cache server **406**. Add master **454** indicates to administration module **474** that a new cache community **402** is being generated with client **404** as master **410** and that the new community **402** should be added to community list **476**.

Returning to decisional step **602**, if client **404** does not receive a response from administration module **474**, then the NO branch of decisional step **602** leads to step **622**. At step **622**, probe message **456** is sent from client **404** to a known port to one or more network nodes. More specifically, dynamic cache application **428** may be configured to listen at a particular location, such as a transmission control protocol/Internet protocol (TCP/IP) port, for clients **404** which wish to join community **402**. This capability provides, for example, the capability to add members to a community **402** in the absence of a cache server **406** or when cache server **406** cannot be contacted. Proceeding to decisional step **624**, if client **402** receives a response to probe message **456** from a master **410** and a community **402**, then the YES branch of decisional **624** leads to step **610** where client **404** will attempt to join the responding community **402**. If no response is received to probe message **456**, then the NO branch of decisional step **624** leads to step **618** and client **404** will attempt to form its own community **402**.

FIG. **10** is a flow chart illustrating a method for allowing a client **404** to join a cache community **402**. The method begins at step **900**, where a master **410** in a community **402**

listens for join request **452** from clients **404**. Next, at step **902**, dynamic affiliation portion **504** at master **410** evaluates join request **452** to determine whether client **404** will be allowed to join community **402**. Master **410** may use any suitable criteria in determining whether to allow client **404** to join community **402**, such as whether community **402** has reached a maximum number of peers **413** and the roundtrip transmission time between client **404** and community **402**. Next, at decisional step **904**, master **410** decides whether client **404** is allowed to join community **402**. If client **404** is not allowed to join community **402**, then the NO branch of decisional step **904** leads to **906** where join request **452** is ignored by master **410**. Alternatively, master **410** may communicate an allow message **424** to client **404** with the indication that client **404** has been denied entry to community **402**.

If client **404** is allowed to join community **402**, then the YES branch of decisional step **904** leads to step **908**. At step **908**, peer list **426** is updated to include client **404**. Proceeding to step **910**, allow message **424** is communicated to client **404** along with the updated peer list **426**. Then, at step **912**, allocation list table **510** is updated to include client **404** and community **402**. More specifically, the distribution of content **460** to be cached within community **402** is adjusted to include the additional capacity provided by client **404**. Then, at step **914**, update peer list message **520** is sent to members **412** by master **410** to inform members **412** to update peer list **426** at each of the members **412** to include client **404**. The method then returns to step **900** where master **410** listens for additional join requests **452**.

FIG. **11** is a flow chart illustrating a method for determining whether a member **412** has unexpectedly departed community **402**. The method begins at step **1000**, where master **410** sends member status request **522** to a selected member **412**. The member **412** to send member status request **522** to may be determined using any suitable method. For example, master **410** may sequentially traverse peer list **426** and send member status request **522** to each member **412** listed in peer list **426**. Next, at step **1002**, master **410** determines whether a response has been received to member status request **522**. If a response is received then the YES branch of decisional step **1002** leads to step **1004**. At step **1004**, master **410** resets the time-out for the member **412** from which the response is received. More specifically, master **410** may have a predetermined time interval after which a particular member **412** is contacted again and the countdown to the expiration of that interval is reset. For example, a member **412** may be contacted once every minute. If no response is received from member **412**, then the NO branch of decisional step **1002** leads to step **1006**. At step **1006**, master **410** removes the member **412** which did not respond from peer list **426**. Proceeding to step **1008**, master **410** sends update peer list message **512** with the indication that the non-responding member **412** should be removed from the peer list **426** of other members **412**. Then, at step **1010**, master **410** updates allocation table **510** to account for the loss of the member **412**.

FIG. **12** is a flow chart illustrating a method for removing a member **412** from community **402**. The method begins at step **1100**, where master **410** listens for members which wish to leave community **402**. Next, at step **1102**, master **410** receives remove peer message **528**. Remove peer message **528** indicates that a member **412** wishes to leave community **402**. Remove peer message **528** is generated by the member **412** which desires to depart community **402** and is communicated to master **410**. Then, at step **1104**, master **410** removes the departing member **412** from peer list **426**. Proceeding to step **1106**, master **410** updates allocation table **510** to account for the departing member **412**. Then, at step **1108**, master **410** sends update peer list message **520** to the remaining members **412** in community **402** to update their peer lists **426** to remove the departing member **412**. The departing member **412** is then removed from community **402** and the method returns to step **1100** where master **410** listens for further departing members.

FIG. **13** is a flow chart illustrating a method for gracefully removing master **410** from community **402**. The method begins at step **1200** where master **410** determines that it is going to leave community **402**. Then, at decisional step **1201**, master **410** determines if it is the only peer **413** in community **402**. If master **410** is not the only peer **413** in community **402**, then NO branch of decisional step **1201** leads to step **1202**. Next, at step **1202**, master **410** selects a new master from members **412** in peer list **426**. Master **410** may use any suitable method for determining who the new master **410** is to be. For example, master **410** may select the new master **410** based on the amount of bandwidth available at a member **412**, the processing capability of member **412**, or other suitable criteria. Then, at step **1204**, master **410** removes itself from peer list **426**. Proceeding to step **1206**, master **410** sends nominate master message **524** to the selected member **412** whose become the new master **410** along with the updated peer list **426** which no longer includes the old master **410**. Then, at decisional step **1208**, the old master determines whether a response has been received from member **412** which is to become the new master **410**. If no response is received from the member **412** which is to become the new master, then the NO branch of decisional step **1208** returns to step **1202** where the current master **410** will select a new master **410** from the other members **412** of community **402**. If a response is received from member **412** which is to become the new master **410** then the YES branch of decisional step **1208** leads to step **1210**. At step **1210**, the existing master **410** shuts itself down. Next, at step **1212**, the member **412** which was selected as the new master **410** becomes the master **410** and sends update peer list message **520** to the remaining members **412** of community **402**. The update peer list message **520** indicates that the remaining members **412** of community **402** should update their peer lists **426** to indicate that the old master **410** is no longer in community **402** and that the new master **410** is the present master **410**. Proceeding to step **1214**, the new master **410** sends update master request **526** to cache server **406**. Update master request **526** indicates to cache server **406** and administration module **474** that community list **476** should be updated to indicate that the new master **410** is the master for community **402**.

Returning to decisional step **1201**, if master **410** determines that it is the only peer **413** in community **402**, then YES branch of decisional step **1201** leads to step **1220**. At step **1220**, master **410** sends remove master request **527** to cache server **406**. Cache server **406** then removes community **402** associated with master **410**. Master **410** then shuts down. The method then ends.

FIG. **14** is a flow chart illustrating a method for allowing a master **410** to unexpectedly depart community **402**. The method begins at step **1300**, where a member **412** determines how much time has elapsed since the last member status request **522** was received by member **412**. Then, at decisional step **1302**, member **412** determines whether the time since the last member status request **522** was received at member **412** exceeds a predetermined threshold. For example, the predetermined threshold may be one minute since the last member status request **522** was received. If the

time since the last member status request **522** was received does not exceed the predetermined threshold, then the NO branch of decisional step **1302** returns to step **1300** where member **412** checks the elapsed time again.

If the elapsed time exceeds the predetermined threshold, then the YES branch of decisional step **1302** leads to step **1304**. At step **1304**, member **412** generates hold election message **530** and communicates hold election message to members **412** in peer list **426**. Member **412** generates hold election message **530** to inform other members **412** in community **402** that master **410** may have shut down and that an election for a new master **410** should be held. Proceeding to step **1306**, member **412** which generated hold election message **530** may receive one or more responses from members **412** and master **410** in community **402**. Then, at step **1308**, member **412** which generated hold election message **530** determines whether any response has been received from master **410**. If master **410** has responded to hold election message **530**, then the YES branch of decisional step **1308** leads to step **1310** where member **412** aborts the attempt to elect a new master **410**. Master **410** may respond because master **410** is still operating and was only temporarily unable to generate a member status request **522** for member **412** which generated hold election message **530**. If master **410** was only temporarily unavailable, then it is unnecessary for a new master **410** to be elected for community **402**.

If no master **410** responds in step **1308**, then the NO branch decisional step **1308** leads to step **1312**. At step **1312**, member **412** generates a new peer list **426** based on responses from members **412** received in step **1306**. A new peer list **426** may be generated so as to determine the remaining members of community **402**. For example, a communications link connecting master **410** and one or more members **412** to other members **412** in community **402** may unexpectedly shut down. The unexpected loss of a communications link may force community **402** to divide into two separate communities **402** and result in the election of a new master **410** for one of the two new communities **402**. Then, at step **1314**, a new master **410** is selected from the remaining peers in **413** in the newly generated peer list **426** from step **1312**. The new master **410** may be selected in any suitable manner, such as by selecting member **412** which detected that master **410** had ceased to respond. Next, at step **1316**, the nominate master message **524** is communicated to member **412** which has been selected to be the new master **410** in step **1314** along with the newly generated peer list **426**. Nominate master message **524** is generated by member **412** which originally detected that master **410** had ceased to respond in step **1302**. Then, at decisional step **318**, member **412** which originally detected that master **410** had ceased to respond determines whether the responses received from the newly nominated master. If no response is received from the newly nominated master, then the NO branch decisional step **1318** returns to step **1314** where another new master is selected from the remaining members **412**. If a response is received then the YES branch decisional step **1318** leads to step **1320**. At step **1320**, the newly nominated master **410** sends update master request **526** to cache server **406** to update community list **476**. The method then ends.

In one embodiment, dynamic affiliation portion **504** supports the addition and removal of members **412** and master **410** using the Dynamic Reef Protocol (DRP). In the DRP, community request **450** comprises a CRMSG_WAKEUP data message, add master request **454** comprises a CRMSG_ADDMASTER data message, update master request **526** comprises a CRMSG_REPLACEMASTER data message, remove master request **527** comprises a CRMSG_REMOVEMASTER data message, join request **452** comprises a CRMSG_REQUESTTOJOIN data message, nominate master message **524** comprises a CRMSG_NOMINATEMASTER data message, update peer list message **520** comprises a CRMSG_UPDATEPEERLIST data message, remove peer message **528** comprises a CRMSG_REMOVEPEER data message, hold election message **530** comprises a CRMSG_HOLDELECTION data message, and member status request **522** comprises a CRMSG_PEERPING data message. In addition, probe message **456** may use the CRMSG_WAKEUP data message. The DRP data messages may be organized using any suitable format. For example, a bit pattern may be used in a type field of a data message to indicate the type of data message being sent. For another example, the DRP may be built on top of or as part of the ICSP protocol.

By providing the capability to cache data in a cache community, system **400** provides the ability to cache large amounts of content to the user. Traditionally, caching of Internet content has been done only at an individual user's computer. This provides the benefits of caching data only after the user has retrieved the data from a remote web server. By providing the capability for computers to join a cache community, content stored in the cache community can be quickly provided to the members without having to retrieve the content from an origin server over a possibly slow Internet. As the use of high-speed connections to the Internet, such as Digital Subscriber Line (DSL) and cable modem technologies, increases, the increased responsiveness of a cache community using high-speed connections may be significant in comparison to the responsiveness of the Internet in general.

In addition, traditional distributed caching solutions have used static, predefined groups cache servers. In contrast, system **400** provides the capability to dynamically add and remove members from a distributed caching community. By allowing dynamic addition and subtraction of members from a distributed caching community, a distributed caching system is supported which does not require a centralized administrator. Also, system **400** supports creating a distributed caching system with heterogeneous computers that may be added and removed from the caching community in a dynamic manner.

Other changes, substitutions and alterations are also possible without departing from the spirit and scope of the present invention, as defined by the following claims.

What is claimed is:

1. A method for dynamic distributed data caching comprising:

providing a cache community on a first side of a point of presence, the cache community comprising at least one peer, the cache community being associated with content obtained from a second side of the point of presence, the content being cached by the at least one peer;

allowing a client to join the cache community;

updating a peer list associated with the cache community to include the client, the peer list indicating the peers in the cache community;

associating the content with the client based on joinder of the client;

re-allocating the cache storage of the content among the peers in the cache community in response to allowing the client to join the community.

2. The method for dynamic distributed data caching according to claim **1** and further comprising:

receiving a join request from the client; and

determining whether to allow the client to join the cache community.

3. The method dynamic distributed data caching according to claim **2**, wherein the join request comprises a CRMSG_REQUESTTOJOIN data message.

4. The method for dynamic distributed data caching according to claim **1**, wherein allowing the client to join the cache community comprises:

generating an allow message;

associating the peer list with the allow message; and

communicating the allow message to the client.

5. The method for dynamic distributed data caching according to claim **4**, wherein allowing the client to joint the cache community comprises:

generating an allow message comprising the peer list updated to include the client;

communicating the allow message to the client; and

communicating the allow message to at least one member associated with the cache community.

6. The method for dynamic distributed data caching according to claim **4**, wherein the allow message comprises a CRMSG_UPDATEPEERLTST data message.

7. The method for dynamic distributed caching according to claim **4**, wherein the peer list associated with the allow message comprises the updated peer list which includes the client.

8. The method for dynamic distributed data caching according to claim **1**, wherein the point of presence is an ISP.

9. The method for dynamic distributed data caching according to claim **1**, wherein a one of the peers comprises a member.

10. The method for dynamic distributed data caching according to claim **1**, wherein a one of the peers comprises a master.

11. The method for dynamic distributed data caching according to claim **1**, wherein associating the content with the client comprises:

allocating the content to peers in the peer list; and

updating an allocation table to indicate the content associated with the peers.

12. The method for dynamic distributed data caching according to claim **11**, wherein the content is cached at more than one peer.

13. The method for dynamic distributed data caching according to claim **11**, wherein the content includes a plurality of internet protocol domain names.

14. The method for dynamic distributed data caching according to claim **1** and further comprising removing the association between the content and the peers.

15. A system for dynamic distributed data caching comprising:

logic encoded on storage and operable to:

provide a cache community on a first side of a point of presence, the cache community comprising at least one peer, the cache community being associated with content obtained from a second side of the point of presence, the content being cached by the at least one peer;

allow a client to join the cache community;

update a peer list associated with the cache community to include the client, the peer list indicating the peers in the cache community;

associate the content with the client based on joinder of the client;

re-allocate the cache storage of the content among the peers in the cache community in response to allowing the client to join the community.

16. The system for dynamic distributed data caching according to claim **15**, wherein the logic is further operable to:

receive a join request from the client; and

determine whether to allow the client to join the cache community.

17. The system for dynamic distributed data caching according to claim **16**, wherein the join request comprises a CRMSG_REQUESTTOJOIN data message.

18. The system for dynamic distributed data caching according to claim **15**, wherein the logic is further operable to:

generate an allow message;

associate the peer list with the allow message;

communicate the allow message to the client.

19. The system for dynamic distributed data caching according to claim **18**, wherein the logic is further operable to:

generate an allow message comprising the peer list updated to include the client;

communicate the allow message to the client; and

communicate the allow message to at least one member associated with the cache community.

20. The system for dynamic distributed data caching according to claim **18**, wherein the allow message comprises a CRMSG_UPDATEPEERLIST data message.

21. The system for dynamic distributed caching according to claim **18**, wherein the peer list associated with the allow message comprises the updated peer list which includes the client.

22. The system for dynamic distributed data caching according to claim **15**, wherein the point of presence is an ISP.

23. The system for dynamic distributed data caching according to claim **15**, wherein a one of the peers comprises a member.

24. The system for dynamic distributed data caching according to claim **15**, wherein a one of the peers comprises a master.

25. The system for dynamic distributed data caching according to claim **15**, wherein the logic is further operable to:

allocate the content to peers in the peer list; and

update an allocation table to indicate the content associated with the peers.

26. The system for dynamic distributed data caching according to claim **25**, wherein the content is cached at more than one peer.

27. The system for dynamic distributed data caching according to claim **25**, wherein the content includes a plurality of internet protocol domain names.

28. The system for dynamic distributed data caching according to claim **15**, wherein the logic is further operable to remove the association between the content and the peers.

29. A method for dynamic distributed data caching comprising:

communicating a community request to an administration module;

receiving a community list from the administration module in response to the community request, the community list including a list of communities;

selecting one of the communities to attempt to join;

generating a join request to attempt to join the selected one of the communities;

receiving an allow message associated with the selected one of the communities;

receiving a peer list associated with the selected one of the communities;

receiving content allocated for storage in caches of peers in the peer list for cache storage re-allocation in response to joining the selected one of the communities; and

providing content for cache storage re-allocation to peers in the peer list in response to joining the selected one of the communities.

30. The method for dynamic distributed data caching according to claim **29**, wherein the community request comprises a CRMSG_WAKEUP data message.

31. The method for dynamic distributed data caching according to claim **29**, wherein the join request comprises a CRMSG_REQUESTTOJOTN data message.

32. A system for dynamic distributed data caching comprising:

logic encoded on storage and operable to:

communicate a community request to an administration module;

receive a community list from the administration module in response to the community request, the community list including a list of communities;

select one of the communities to attempt to join;

generate a join request to attempt to join the selected one of the communities;

receive an allow message associated with the selected one of the communities;

receive a peer list associated with the selected one of the communities;

receive content allocated for storage in caches of peers in the peer list for cache storage re-allocation in response to joining the selected one of the communities; and

provide content for cache storage re-allocation to peers in the peer list in response to joining the selected one of the communities.

33. The system for dynamic distributed data caching according to claim **32**, wherein the community request comprises a CRMSG_WAKEUP data message.

34. The system for dynamic distributed data caching according to claim **32**, wherein the join request comprises a CRMSG_REQUESTTOJOTN data message.

35. A system for dynamic distributed data caching comprising:

means for providing a cache community on a first side of a point of presence, the cache community comprising at least one peer, the cache community being associated with content obtained from a second side of the point of presence, the content being cached by the at least one peer;

means for allowing a client to join the cache community;

means for updating a peer list associated with the cache community to include the client, the peer list indicating the peers in the cache community;

means for associating the content with the client based on joinder of the client;

means for re-allocating the cache storage of the content among the peers in the cache community in response to allowing the client to join the community.

36. A system for dynamic distributed data caching comprising:

means for communicating a community request to an administration module;

means for receiving a community list from the administration module in response to the community request, the community list including a list of communities;

means for selecting one of the communities to attempt to join;

means for generating a join request to attempt to join the selected one of the communities;

means for receiving an allow message associated with the selected one of the communities;

means for receiving a peer list associated with the selected one of the communities;

means for receiving content allocated for storage in caches of peers in the peer list for cache storage re-allocation in response to joining the selected one of the communities; and

means for providing content for cache storage re-allocation to peers in the peer list in response to joining the selected one of the communities.

* * * * *

FORM 30. Certificate of Service

Form 30
Rev. 03/16

UNITED STATES COURT OF APPEALS FOR THE FEDERAL CIRCUIT

CERTIFICATE OF SERVICE

I certify that I served a copy on counsel of record on December 13, 2016 by:

- ☐ U.S. Mail
- ☐ Fax
- ☐ Hand
- ☒ Electronic Means (by E-mail or CM/ECF)

Thomas R. Waskom	/s/ Thomas R. Waskom
Name of Counsel	Signature of Counsel

Law Firm	Hunton & Williams, LLP
Address	951 East Byrd Street
City, State, Zip	Richmond, VA 23219
Telephone Number	(804) 788-8403
Fax Number	(804) 343-4868
E-Mail Address	twaskom@hunton.com

NOTE: For attorneys filing documents electronically, the name of the filer under whose log-in and password a document is submitted must be preceded by an "/s/" and typed in the space where the signature would otherwise appear. Graphic and other electronic signatures are discouraged.

Reset Fields

FORM 19. Certificate of Compliance With Rule 32(a)

Form 19
Rev. 12/16

UNITED STATES COURT OF APPEALS FOR THE FEDERAL CIRCUIT

CERTIFICATE OF COMPLIANCE WITH TYPE-VOLUME LIMITATION, TYPEFACE REQUIREMENTS, AND TYPE STYLE REQUIREMENTS

1. This brief complies with the type-volume limitation of Federal Rule of Federal Circuit Rule 32(a) or Federal Rule of Federal Circuit Rule 28.1.

☒ This brief contains [*state the number of*] 8,891 words, excluding the parts of the brief exempted by Federal Rule of Appellate Procedure 32(f), or

☐ This brief uses a monospaced typeface and contains [*state the number of*] _____ lines of text, excluding the parts of the brief exempted by Federal Rule of Appellate Procedure 32(f).

2. This brief complies with the typeface requirements of Federal Rule of Appellate Procedure 32(a)(5) or Federal Rule of Federal Circuit Rule 28.1 and the type style requirements of Federal Rule of Appellate Procedure 32(a)(6).

☒ This brief has been prepared in a proportionally spaced typeface using

[*state name and version of word processing program*] Microsoft Word Professional Plus 2010 in

[*state font size and name of type style*] 14 pt. Cambria , or

☐ This brief has been prepared in a monospaced typeface using

[*state name and version of word processing program*] _____ with

[*state number of characters per inch and name of type style*] _____ .

/s/ Thomas R. Waskom

(Signature of Attorney)

Thomas R. Waskom

(Name of Attorney)

Appellant Parallel Networks, LLC

(State whether representing appellant, appellee, etc.)

Dec 13, 2016

(Date)

Reset Fields