UNITED STATES DISTRICT COURT
EASTERN DISTRICT OF VIRGINIA
Alexandria Division

LIMELIGHT NETWORKS, INC.,

Plaintiff,

v.

XO COMMUNICATIONS, LLC., AND AKAMAI TECHNOLOGIES, INC.,

Defendants.

Case No. 1:15CV720-JAG

JURY TRIAL DEMANDED

COMPLAINT

Limelight Networks, Inc. ("Limelight" or "Plaintiff") hereby alleges for its Complaint against Defendants XO Communications, LLC ("XO") and Akamai Technologies, Inc. ("Akamai") (collectively, "Defendants") on personal knowledge as to its own actions and on information and belief as to the actions of others, as follows:

NATURE OF THE ACTION

1. This is a patent infringement action by Limelight to end Defendants' unauthorized and infringing manufacture, use, sale, offering for sale, and/or importation of products and methods incorporating Limelight's patented inventions.

2. Limelight holds all substantial rights and interest in the Patents-in-Suit described below, including the exclusive right to sue Defendants for infringement and recover damages.

3. Plaintiff Limelight seeks monetary damages, prejudgment interest and injunctive relief for Akamai's and XO's past and on-going infringement of the Patents-in-Suit.

THE PARTIES

4. Limelight Networks, Inc. ("Limelight") is a corporation organized and existing under the laws of Delaware with its principal place of business at 222 South Mill Ave., Suite 800, Tempe, Arizona, 85281.

5. On information and belief, Defendant XO Communications, LLC. ("XO") is a corporation existing and organized under the laws of Delaware and has its principal place of business at 13865 Sunrise Valley Drive, Herndon, VA 20171.

6. On information and belief, Defendant Akamai Technologies, Inc. ("Akamai," or "Defendant") is a corporation existing and organized under the laws of Delaware and has its principal place of business at 150 Broadway, Cambridge, Massachusetts, 02142.

7. Founded in 2001, Limelight is a leader in digital content delivery. Its content acceleration technologies and services enable publishers to deliver their digital content (*e.g.,* web pages, videos, full-length movies and television shows, operating system updates, and online games) on any device, anywhere in the world.

8. Akamai also sells products and services for digital content delivery. As such, numerous Limelight products and services compete with those offered by Akamai. For example, Limelight and Akamai each operate a global Content Delivery Network ("CDN")—a geographically distributed network of servers that their customers, such as web sites, software applications, video-on-demand and streaming media providers, can use to accelerate content delivery to their end users. Such CDNs accelerate content delivery through a variety of techniques, such as caching content at numerous servers so that the content can be delivered to end users from locations close to the user. XO is a telecommunications company that is engaged in an extensive partnership with Akamai, including as a reseller of Akamai services and as a partner in deployment and operation of hardware and software components of a CDN.

9. While Akamai was one of the first to market with a CDN solution, newer entrants such as Limelight have rapidly innovated and developed new technology contributions—and obtained patent protection for those contributions—which Akamai has then implemented in order to remain competitive.

JURISDICTION AND VENUE

10. This action for patent infringement arises under the patent laws of the United States, Title 35 of the United States Code.

11. This Court has subject matter jurisdiction pursuant to 28 U.S.C. §§ 1331 and 1338(a).

12. This Court has general and specific personal jurisdiction over Defendant XO. XO has substantial contacts with the forum as a consequence of establishing its headquarters in Virginia and in this District, and XO conducts substantial business in Virginia. XO sells, makes, uses, and offers for sale its products and services, including products and services that infringe Plaintiff's patents, within the state of Virginia, including to customers in Virginia. Such customers include USA Today, a customer it shares with Akamai in connection with use of systems that infringe the asserted patents. In addition, on information and belief, XO has established data centers for use in infringing the asserted Limelight Patents in this district, including at 12100 Sunrise Valley Drive Reston, VA, and at 8613 Lee Highway, Fairfax, VA 22031.

13. XO has committed and continues to commit acts of patent infringement, including making and using infringing systems, and performing infringing methods, within this district, including in conjunction with Akamai.

14. This Court has general and specific personal jurisdiction over Defendant Akamai. Akamai has substantial contacts with the forum as a consequence of conducting substantial

business in Virginia and in this District, including establishment of offices at 11111 Sunset Hills Road, Suite 250, Reston, VA 20190. In addition, Akamai has established a significant presence in this forum by locating its content delivery servers accused of infringing the patents asserted in this action in Virginia and in this District. For example, according to publicly available documentation, Akamai has placed more than 200 racks of its accused content delivery servers and associated hardware and software at a data center located at 12100 Sunrise Valley Drive Reston, VA 20191, and has placed an additional 170 racks of its accused content delivery servers and associated hardware and software at a data center located at 1780 Business Center Drive, Reston, VA 20190. Akamai has also located its accused content delivery servers in data centers in Sterling, VA, Manassas, VA, Ashburn, VA, and Vienna, VA, each of which are located in this District. The operation of these content delivery servers in Virginia and in this District constitutes infringement of the asserted Limelight patents in this District. In addition, this Court has jurisdiction over Akamai because Akamai has conducted business with a Virginia-based corporation, XO, for the purpose of infringing the patents.

15. Akamai has committed and continues to commit acts of patent infringement, including making and using infringing systems, and performing infringing methods, within this district, including in conjunction with XO.

16. Venue is proper for XO in this District under 28 U.S.C. §§ 1391(b) and (c), and 1400(b) because, as described above, a substantial part of the events giving rise to Limelight's claims occurred in this district, and because XO, which is headquartered in Herndon, Virginia, resides within this district.

17. Venue is proper for Akamai in this District under 28 U.S.C. §§ 1391(b) and (c), and 1400(b) because, as described above, Akamai has a regular and established practice of

business in this district and has committed acts of infringement in this district, including by virtue of its far-reaching relationship with XO, whose headquarters are in this district. In 2001, Akamai and XO entered into a strategic agreement under which "XO will provide co-location space in its data centers for the deployment of additional Akamai servers," and "XO's interconnection bandwidth related services and hosting capabilities [will] help Akamai to expand its reach to enable users to benefit from improved performance and accelerated delivery of the Web's most popular streaming media, software applications and content served on Akamai's globally distributed network"—in short, to co-locate at XO-owned or XO-administered data centers the products and services accused of infringing the Limelight patents asserted in this action. "XO Communications and Akamai Announce Strategic Alliance," dated May 17, 2001, at http://www.akamai.eu/html/about/press/releases/2001/press_051701.html (last visited November 29, 2015). On information and belief, one such data center is located at 12100 Sunrise Valley Drive, Reston, VA. XO also resells Akamai's accused content delivery services to its customers.

JOINDER

18. Joinder is proper under 35 U.S.C. § 299 because questions of fact common to each Defendant will arise in the action. As detailed below, Limelight alleges patent infringement by Defendants in connection with their making and using systems, and their practice of methods, for accelerating the delivery of digital content based on hardware and software developed by Defendant Akamai. As such, factual issues regarding the operation of that hardware and software are common to Akamai and to XO.

19. Joinder is further proper because some of Defendants' infringement arises out of the same transaction, occurrence, or series of transactions or occurrences relating to the making, using, importing into the United States, offering for sale, or selling of the same accused product

or process. For example, as described below, while each of XO and Akamai can directly infringe the asserted claims, when the infringing system includes a combination of XO-deployed and Akamai servers, Akamai and XO act jointly or in concert to perform the infringing acts, and in that instance, the infringement is not complete until both XO and Akamai have provided or performed their respective parts.

THE ASSERTED PATENTS

20. As a global leader in digital content delivery, Limelight has sought patent protection for many of its innovations in this field, including the patents asserted in this matter.

THE CONDITIONAL PROTOCOL CONTROL PATENTS

21. On May 11, 2010, the United States Patent and Trademark Office duly and legally issued U.S. Patent No. 7,715,324 ("the 324 Patent"), entitled "Conditional Protocol Control." A copy of the 324 Patent is attached to the Complaint as Exhibit A.

22. On December 10, 2014, the United States Patent and Trademark Office duly and legally issued U.S. Patent No. 8,750,155 ("the 155 Patent"), entitled "Conditional Protocol Control." A copy of the 155 Patent is attached to the Complaint as Exhibit B.

23. The 324 and 155 Patents arose out of the innovative work performed by Limelight engineers to utilize selective optimizations of the Transport Control Protocol ("TCP"), a core Internet protocol that governs how content is delivered over the web, in order to accelerate their customers' delivery of Internet content, including web pages, downloadable files, and media content such as images or audio/video, to their end users. The inventors of the 324 and 155 Patents developed ways to use TCP optimizations to accelerate such content conditionally, such as on a customer-by-customer, or file-by-file basis, in order to optimize this content delivery for any given set of circumstances.

24. In October 2008, Limelight licensed the basic technology and software for optimizing TCP connections from FastSoft, Inc., a startup company that developed an algorithm known as FastTCP, which allowed for accelerating TCP connections on one end (the server end) of an Internet connection. FastSoft had no experience in content acceleration in the context of CDNs such as those provided by Limelight and by Akamai, and instead pursued a business model whereby it sought to sell hardware appliances that implemented its algorithm. On top of the elementary technology supplied by FastSoft, Limelight engineers developed a complete TCP optimization solution for CDNs that could analyze a request for content received by a content server and, based on information obtained from the request, such as the identity of the customer or the type of content requested, conditionally apply a set of transport protocol optimizations on a connection by connection basis. Because optimization could be applied conditionally on a connection-by-connection basis under the Limelight solution, each connection could be optimized differently, according to a configurable profile.

25. Limelight sought and obtained patent protection for its conditional protocol control innovations, including the 324 and 155 Patents.

26. Limelight also shared its conditional protocol control innovations with FastSoft, including providing FastSoft with the functional requirements for its Deliver XD service that implemented these innovations, and collaborating with FastSoft on the improvement of its technology for use within a Content Delivery Network.

27. In September 2012, Akamai announced that it had acquired FastSoft, and had integrated FastSoft's engineering team—a team that had been exposed to Limelight's innovations—into Akamai. Shortly thereafter, Akamai communicated to Limelight that all

FastSoft products were entering their End Of Life ("EOL") phase, and support for these products would be discontinued within one year, or earlier if allowed under the license agreement.

28. At least by September 2013, Akamai had deployed FastSoft-based TCP protocol optimization in its own Content Delivery Network in a manner strikingly similar to the implementation created and patented by Limelight. Like Limelight, Akamai's TCP optimization does not utilize FastSoft hardware appliances, but instead deploys TCP optimizations in software at content servers in the Content Delivery Network. Like Limelight, Akamai's TCP optimization is conditional, highly configurable via a configuration profile, and can be set connection-by-connection. Like Limelight, Akamai's TCP optimization parameters are based on analysis of the received content request. Each of these aspects is described in Limelight's conditional protocol control patents prior to Akamai's deployment.

OTHER LIMELIGHT PATENTS

29. On October 7, 2014, the United States Patent and Trademark Office duly and legally issued U.S. Patent No. 8,856,263 ("the 263 Patent"), entitled "Systems and methods thereto for acceleration of web pages access using next page optimization, caching and pre-fetching techniques." A copy of the 263 Patent is attached to the Complaint as Exhibit C.

30. On March 25, 2014, the United States Patent and Trademark Office duly and legally issued U.S. Patent No. 8,683,002 ("the 002 Patent"), entitled "Content delivery network cache grouping." A copy of the 002 Patent is attached to the Complaint as Exhibit D.

31. On April 21, 2015, the United States Patent and Trademark Office duly and legally issued U.S. Patent No. 9,015,348 ("the 348 Patent"), entitled "Dynamically selecting between acceleration techniques based on content request attributes." A copy of the 348 Patent is attached to the Complaint as Exhibit E.

32. On December 24, 2013, the United States Patent and Trademark Office duly and legally issued U.S. Patent No. 8,615,577 ("the 577 Patent"), entitled "Policy based processing of content objects in a content delivery network using mutators." A copy of the 577 Patent is attached to the Complaint as Exhibit F.

33. Limelight owns all substantial right, title, and interest in the 324, 155, 002, 263, 348, and 577 Patents, and holds the right to sue and recover damages for infringement thereof, including past infringement.

COUNT I AGAINST AKAMAI:

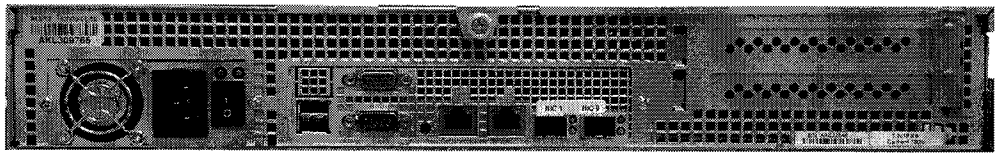
INFRINGEMENT OF U.S. PATENT NO. 8,750,155

34. Limelight incorporates and realleges paragraphs 1-33 above as if fully set forth herein.

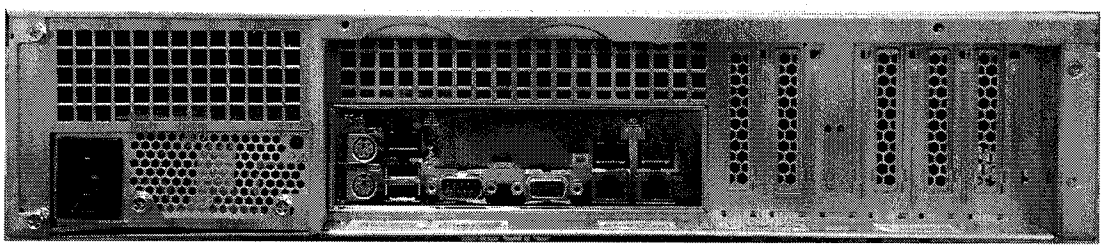
35. On information and belief, Akamai has infringed and continues to infringe one or more claims of the 155 Patent, including but not limited to claims 1, 3, 8, 9, 10, 11, 12, 13, 15, 18, 19, and 20 pursuant to 35 U.S.C. § 271(a), literally or under the doctrine of equivalents, by making, using, selling, and/or offering to sell in the United States without authority and/or importing into the United States without authority, the Akamai Intelligent Platform, including a content delivery network with edge servers running Akamai's TCP optimization functionality, as well as services associated therewith (the 155 Infringing Products). Based on information and belief, and publicly available documentation, the 155 Infringing Products perform TCP optimization by modifying pre-existing TCP settings based upon parameters that are determined at least in part with reference to information in the URLs of end-user requests processed by Akamai.

36. Further, Akamai's edge servers with TCP optimization meet the requirements of the claimed content distribution server, as reflected by publicly available Akamai documentation.

On information and belief, to perform their basic role, Akamai's edge servers, including edge servers that are co-located with XO, have multiple network ports to send and receive data. For example, Akamai publishes the following images showing Akamai servers having two Ethernet ports:



Akamai Hands And Eyes Guide, *available at* https://fieldtech.akamai.com/heguide/images/Server_Hardware/X4i_1-5x18_10G_Rear_Large.jpg (last visited November 29, 2015).



Akamai Hands And Eyes Guide, *available at* https://fieldtech.akamai.com/heguide/images/Server_Hardware/X1_2x8_CacheH_Rear_Large.jpg (last visited November 29, 2015). Further information about Akamai's deployed network interfaces can be found in Akamai Hands And Eyes Guide, *available at* https://fieldtech.akamai.com/heguide/Network_Packages.html (last visited November 29, 2015).

37. Further, Akamai's geographically distributed deployments of its edge servers, as deployed and operated for example by XO in its data centers, include routers that "allow Akamai to direct traffic between Akamai's equipment and the providers that Akamai connects to."

[Akamai Hands And Eyes Guide], *available at* https://fieldtech.akamai.com/heguide/Router_Hardware.html. When such equipment is deployed in a manner that connects Akamai's edge servers to the Internet, for example by XO in its data centers, it likewise has multiple ports configured to send and receive data over a connecting network.

38. Likewise, Akamai's servers, including on information and belief servers that are co-located with XO, include processors:

Server Generations

The following server generations are listed from most recently deployed to no longer being deployed. This a brief overview of each generation's specifications.

Generation	Specifications (Partial List)
X8	Intel Broadwell DE
X7	Intel Skylake Intel Haswell E3-1200 V5 series, up to 64 GB Unbuffered DDR4
X6m	Intel Haswell E5-2600 V3 series, 16G to 256G DDR4 Registered
X6	Intel Haswell E3-1200 V3 series, 16G or 32G DDR3
X5	Intel Sandy Bridge E5 2600 EP series, 128GB or 256GB DDR3
X4i	Intel Ivy Bridge E3-1270 V2, 16G DDR3
X4	Intel Sandy Bridge E3-1270, 16G DDR3
X2	Intel Nehalem X3470, 8G to 32G DDR3
X1	Intel Nehalem X3470, 8G DDR3
G10	AMD Opteron 6204 Quad Core, 64G DDR3
G9	AMD Opteron 4184 Hexa Core, 8G DDR3
G7	AMD Opteron 1389 Quad Core, 8G DDR2
G6	AMD Opteron 2381 Quad Core, 8G DDR2
G5	AMD Athlon II 240e Dual Core, 8G DDR3
G4	AMD Opteron 1218 Dual Core, 4G DDR2
G3	AMD Opteron 244, 2G or 4G M DDR1
G2	AMD Opteron 244, 1G or 2G DDR1
G1	AMD Opteron 244, 512M DDR1

Akamai Hands And Eyes Guide, *available at* https://fieldtech.akamai.com/heguide/Server_Hardware.html (last visited November 29, 2015). The 155 Infringing Products include a protocol handler, such as the TCP/IP protocol stack implementation, that establishes and maintains connections with end-users. The 155 Infringing Products have storage that they use to store customer content to serve to end-users.

39. The 155 Infringing Products perform TCP optimization in a manner that infringes the asserted claims. Specifically, the 155 Infringing Products monitor connections with end-users for requests. When they receive end-user requests the 155 Infringing Products determine parameters that relate to processing and memory capabilities in the TCP protocol, such as

maximum buffer space and socket buffer values. These determinations are made by the edge server, including on information and belief in those cases where the edge server is co-located with XO, based at least in part on reference to information in the URL of the request (such as for example, the hostname field or the customer ID). This information is utilized, in addition to other information, for the Akamai server to determine how aggressive the TCP optimization should be for that connection. Once that determination is made, the TCP settings are altered to put that new optimization into effect by changing pre-existing TCP values to new values that are consistent with the correct level of TCP optimization. As Akamai's documentation explains in detail:

> At a high-level, it operates in two modes: slow-start and congestion-avoidance. Those are different phases in the protocol that attempt to probe the network for available bandwidth using slightly different approaches. TCP maintains what's referred to as a congestion window, which determines how many packets can be in-flight on the network at any point in time. The higher the congestion window, the greater TCP believes its fair share of the available bandwidth is. In slow-start, for every packet that is correctly received (i.e., acknowledged), the congestion window is expanded by a factor of 2; which is an aggressive rate of increase despite the "slow-start" misnomer. In congestion-avoidance, TCP believes it is much closer to its fair share and probes the network much less aggressively. Instead of expanding the congestion window by a factor of 2, the congestion window is only expanded by a single packet after an entire congestion window worth of packets is acknowledged by the receiver. In both cases, once loss is detected, the congestion window is shrunk and the probing starts again.
>
> Akamai optimizes TCP by tuning knobs that control where we start probing from (i.e., the initial congestion window), how quickly we expand the congestion window in both the slow-start (factor of 2 or 3 or higher) and congestion-avoidance (increase by 1 or 2 or higher) phases, as well as how much we back off when a loss is detected (shrink window by 50%, 30% or even less). That allows us to control how aggressive the protocol is in acquiring bandwidth. A TCP instance that probes aggressively and does not back off as much will acquire a larger share of the available bandwidth, under most network conditions.

TCP Optimizations, *available at* https://developer.akamai.com/stuff/Optimization/TCP_Optimizations.html (last visited November 29, 2015).

40. Akamai's TCP optimization has at least medium and low settings, which determine how aggressively TCP is optimized for the connection. Further, the selection of a level

of TCP optimization results in the timing of data transmission at the transport layer being modified as a function of the rate at which the congestion window is changed. On information and belief, Akamai's TCP optimization also results in changing the burst size of the connection.

41. Further, Akamai utilizes latency estimates to select the correct level of TCP optimization. As Akamai explains: "It's a reactive protocol. FastTCP, the Akamaized version of FastSoft's solution, attempts to estimate the correct transmission rate by utilizing latency estimates, among other things, without actually inducing loss. It's a proactive protocol." TCP Optimizations, *available at* https://developer.akamai.com/stuff/Optimization/TCP_Optimizations.html (last visited November 29, 2015).

42. Further, on information and belief, Akamai makes TCP optimization determinations based on a predetermined performance profile, for instance, based on the identity of the customer or the specific customer content provided by Akamai. On information and belief, this performance profile is stored on at least a customer-by-customer basis and is used to set the level of TCP optimization (such as medium and low).

43. Further, when an Akamai edge server with TCP optimization does not have content requested by an end-user in its own cache, the edge server can obtain that content from the cache or caches of neighboring or "parent" Akamai edge servers, provide that content to the end-user, and also store that same content in its own cache for future use. When an Akamai edge server obtains the missing content from the cache of an edge server hosted by XO, or vice versa, Akamai and XO act in concert or jointly to practice the claimed inventions and the infringement is not complete until both Akamai and XO have provided or performed their respective parts.

44. On information and belief, Akamai's TCP optimization, which infringes the asserted claims, utilizes technology that Akamai received from Limelight by way of its

acquisition of FastSoft, as discussed above at ¶¶ 24-28. As Akamai explains in its public documentation:

> There has been a lot of research on TCP over the last 10–15 years, much of which has focused on improving some aspect of TCP's behavior. The key finding is that TCP does not work well under all types of network characteristics, including loss/latency patterns, cross-traffic, how quickly the available bandwidth changes over time, and so on. In 2012 Akamai acquired FastSoft, a company that developed a novel transport solution that does not rely on detecting loss to adapt the congestion window. In general, TCP induces loss, by constantly probing for more available bandwidth, in order to estimate the correct transmission rate. It then reacts to the occurrence of loss. It's a reactive protocol. FastTCP, the Akamaized version of FastSoft's solution, attempts to estimate the correct transmission rate by utilizing latency estimates, among other things, without actually inducing loss. It's a proactive protocol.

TCP Optimizations, *available at* https://developer.akamai.com/stuff/Optimization/TCP_Optimizations.html.

45. The making, and operation, of the 155 Infringing Products as described above constitutes infringement of at least the above-mentioned claims of the 155 Patent pursuant to 35 U.S.C. § 271(a).

46. Unless enjoined by this Court, Akamai will continue to infringe the 155 Patent.

47. As a result of Akamai's conduct, Limelight has suffered and will continue to suffer irreparable injury, for which it has no adequate remedy at law. Limelight has also suffered damages as a result of Akamai's infringement of the 155 Patent and will continue to suffer such damages, until an injunction issues, in an amount and manner yet to be determined.

COUNT II AGAINST AKAMAI:

INFRINGEMENT OF U.S. PATENT NO. 7,715,324

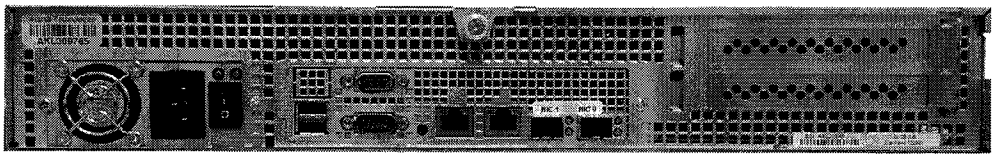
48. Limelight incorporates and realleges paragraphs 1-33 above as if fully set forth herein.

49. On information and belief, Akamai has infringed and continues to infringe one or more claims of the 324 Patent, including but not limited to claims 1, 2, 4, 5, 6, 7, 8, 10, and 11, pursuant to 35 U.S.C. § 271(a), literally or under the doctrine of equivalents, by making, using, selling, and/or offering to sell in the United States without authority and/or importing into the United States without authority, the Akamai Intelligent Platform, including a content delivery network with edge servers running Akamai's TCP optimization functionality, as well as services associated therewith (the 324 Infringing Products). Based on information and belief, and publicly available documentation, the 324 Infringing Products perform TCP optimization by modifying pre-existing TCP settings based upon parameters that are determined at least in part with reference to information in the URLs of end-user requests processed by Akamai.

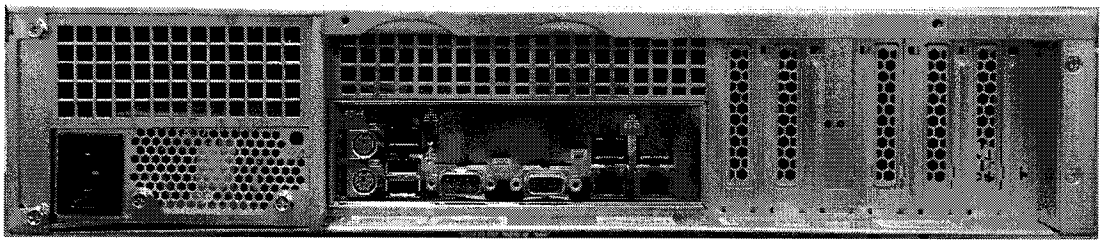
50. Specifically, Akamai's edge servers with TCP optimization, including on information and belief, edge servers that are co-located with XO, manage the delivery of content over network connections in satisfaction of the asserted claims, on information and belief and based on publicly available documentation.

51. The 324 Infringing Products include a protocol handler, such as the TCP/IP protocol stack implementation, that establishes and maintains connections with end-users.

52. On information and belief, to perform their basic role, Akamai's edge servers, including edge servers that are co-located with XO, include network ports used to receive and send communications over a network. For example, Akamai publishes the following images showing Akamai servers having Ethernet ports:



Akamai Hands And Eyes Guide, *available at* https://fieldtech.akamai.com/heguide/images/Server_Hardware/X4i_1-5x18_10G_Rear_Large.jpg (last visited November 29, 2015).



Akamai Hands And Eyes Guide, *available at* https://fieldtech.akamai.com/heguide/images/Server_Hardware/X1_2x8_CacheH_Rear_Large.jpg (last visited November 29, 2015). Further information about Akamai's deployed network interfaces can be found in Akamai Hands And Eyes Guide, *available at* https://fieldtech.akamai.com/heguide/Network_Packages.html (last visited November 29, 2015).

53. Further, Akamai's geographically distributed deployments of its edge servers, as deployed and operated for example by XO in its data centers, include routers that "allow Akamai to direct traffic between Akamai's equipment and the providers that Akamai connects to." [Akamai Hands And Eyes Guide], *available at* https://fieldtech.akamai.com/heguide/Router_Hardware.html. When such equipment is deployed in a manner that connects Akamai's edge servers to the Internet, for example by XO in its data centers, it likewise has multiple ports configured to send and receive data over a connecting network.

54. The 324 Infringing Products perform TCP optimization in a manner that infringes the asserted claims. Specifically, the 324 Infringing Products monitor connections with end-users for requests. When they receive end-user requests Akamai's 324 Infringing Products determine a level of TCP optimization in part on reference to information in the URL of the request (such as for example, the hostname field, or the customer ID, which constitute alphanumeric strings). This information is utilized, in addition to other information, for the Akamai server, including on information and belief, cases where the edge server is co-located with XO to determine how aggressive the TCP optimization should be for that connection. Once that determination is made the TCP settings are altered to put that new optimization into effect by changing pre-existing TCP values to new values that are consistent with the correct level of TCP optimization. As Akamai's documentation explains in detail:

> At a high-level, it operates in two modes: slow-start and congestion-avoidance. Those are different phases in the protocol that attempt to probe the network for available bandwidth using slightly different approaches. TCP maintains what's referred to as a congestion window, which determines how many packets can be in-flight on the network at any point in time. The higher the congestion window, the greater TCP believes its fair share of the available bandwidth is. In slow-start, for every packet that is correctly received (i.e., acknowledged), the congestion window is expanded by a factor of 2; which is an aggressive rate of increase despite the "slow-start" misnomer. In congestion-avoidance, TCP believes it is much closer to its fair share and probes the network much less aggressively. Instead of expanding the congestion window by a factor of 2, the congestion window is only expanded by a single packet after an entire congestion window worth of packets is acknowledged by the receiver. In both cases, once loss is detected, the congestion window is shrunk and the probing starts again.
>
> Akamai optimizes TCP by tuning knobs that control where we start probing from (i.e., the initial congestion window), how quickly we expand the congestion window in both the slow-start (factor of 2 or 3 or higher) and congestion-avoidance (increase by 1 or 2 or higher) phases, as well as how much we back off when a loss is detected (shrink window by 50%, 30% or even less). That allows us to control how aggressive the protocol is in acquiring bandwidth. A TCP instance that probes aggressively and does not back off as much will acquire a larger share of the available bandwidth, under most network conditions.

TCP Optimizations, *available at* https://developer.akamai.com/stuff/Optimization/TCP_Optimizations.html (last visited November 29, 2015).

55. Thus, Akamai's TCP optimization has at least medium and low settings, which determine how aggressively TCP is optimized for the connection. Further, the selection of a level of TCP optimization results in the timing of data transmission at the transport layer being modified as a function of the rate at which the congestion window is changed.

56. This process of TCP optimization is performed, on information and belief, on multiple connections, including multiple simultaneous connections, including from different end-users, where the multiple connections are used to serve different content. On information and belief, the TCP optimization process employed by Akamai can apply different levels of TCP optimization to these different connections.

57. Further, Akamai utilizes other attributes such as latency estimates to select the correct level of TCP optimization. As Akamai explains: "FastTCP, the Akamaized version of FastSoft's solution, attempts to estimate the correct transmission rate by utilizing latency estimates, among other things, without actually inducing loss. It's a proactive protocol." TCP Optimizations, *available at* https://developer.akamai.com/stuff/Optimization/TCP_Optimizations.html (last visited November 29, 2015).

58. Further, on information and belief, Akamai makes TCP optimization determinations based at least in part on attributes associated with the identity of the customer or the specific customer content provided by Akamai. On information and belief these attributes and the information is stored on at least a customer-by-customer basis and is used to set the level of TCP optimization (such as medium and low).

59. On information and belief, Akamai's TCP optimization, which infringes the asserted claims, utilizes technology that Akamai received from Limelight by way of its acquisition of FastSoft, as discussed above at ¶¶ 24-28. As Akamai explains in its public documentation:

> There has been a lot of research on TCP over the last 10–15 years, much of which has focused on improving some aspect of TCP's behavior. The key finding is that TCP does not work well under all types of network characteristics, including loss/latency patterns, cross-traffic, how quickly the available bandwidth changes over time, and so on. In 2012 Akamai acquired FastSoft, a company that developed a novel transport solution that does not rely on detecting loss to adapt the congestion window. In general, TCP induces loss, by constantly probing for more available bandwidth, in order to estimate the correct transmission rate. It then reacts to the occurrence of loss. It's a reactive protocol. FastTCP, the Akamaized version of FastSoft's solution, attempts to estimate the correct transmission rate by utilizing latency estimates, among other things, without actually inducing loss. It's a proactive protocol.

TCP Optimizations, *available at* https://developer.akamai.com/stuff/Optimization/TCP_Optimizations.html.

60. The making and operation of the 324 Infringing Products as described above constitutes infringement of at least the above-mentioned claims of the 324 Patent pursuant to 35 U.S.C. § 271(a).

61. Unless enjoined by this Court, Akamai will continue to infringe the 324 Patent.

62. As a result of Akamai's conduct, Limelight has suffered and will continue to suffer irreparable injury, for which it has no adequate remedy at law. Limelight has also suffered damages as a result of Akamai's infringement of the 324 Patent and will continue to suffer such damages, until an injunction issues, in an amount and manner yet to be determined.

COUNT III AGAINST AKAMAI:

INFRINGEMENT OF U.S. PATENT NO. 8,683,002

63. Limelight incorporates and realleges paragraphs 1-33 above as if fully set forth herein.

64. On information and belief, Akamai has infringed and continues to infringe one or more claims of the 002 Patent, including but not limited to claims 1, 2, 3, 4, 5, 7, 8, 9, 10, 13, 15, 16, 17, 18, and 20, pursuant to 35 U.S.C. § 271(a), literally or under the doctrine of equivalents, by making, using, selling, and/or offering to sell in the United States without authority and/or importing into the United States without authority, the Akamai Intelligent Platform, including a content delivery network with edge servers, as well as services associated therewith (the 002 Infringing Products). Akamai's content delivery network includes a plurality of points of presence that are distributed geographically. These points of presence include edge servers. Based on information and belief, and publicly available documentation, edge servers that do not have user-requested content in their own caches can ask other edge servers whether they have the requested content in their caches and if so the content is provided to the user.

65. Specifically, the 002 Infringing Products include edge servers that receive end-user requests for content in the form of URLs. When such requests are received, if the content is not in the cache of the edge server, the edge server contacts neighboring edge servers to determine whether the neighboring edge servers have the user-requested content in their own caches. If the neighboring edge server has the requested content the content is served to the end user. As Akamai explains:

> The edge server will check its local cache as well as the caches of other machines in the server deployment to see if the requested object has been seen before. If the object is found, the edge server will verify that the object is not stale and will serve it to the user.

> If the object is found in the cache but it is stale, the edge server will contact another Akamai deployment or the origin to see if a newer version has been uploaded.

Client to Edge Servers to Origin, *available at* https://developer.akamai.com/stuff/Overview/Client_Edge_Servers_Origin.html (last visited November 29, 2015).

66. Further, if the edge server receiving the request for content that it does not have in its own cache is also unable to get that content from a neighboring edge server, the edge server requests the content from a server higher in Akamai's distribution hierarchy, including in some instances, the origin server, until it is able to retrieve the requested content. On information and belief, this process is based in part on analysis of the URL of the content request. As Akamai explains:

> When an edge server gets a request for an object that it hasn't yet seen, it will download it from either another Akamai deployment or the origin. The customer's metadata determines whether the edge contacts the origin directly, or if it applies some sort of tiered distribution hierarchy.
>
> Tiered distribution is used to provide greater origin offload by allowing many Akamai edge deployments to go forward to a smaller set of deployments which in turn go forward to the origin. In the case of Akamai's Site Shield product, the Customer's IT department can program the IP addresses of these top-tier machines into their firewall and block access to their network from all other Internet hosts.
>
> At this point, caching rules are applied to the object and the requested bytes are delivered to the user.

Client to Edge Servers to Origin, *available at* https://developer.akamai.com/stuff/Overview/Client_Edge_Servers_Origin.html (last visited November 29, 2015).

67. In addition, the edge servers in a given instance of infringement can both be located within the same point of presence, and the infringement can involve all of the caches in a given point of presence. Likewise, the servers higher in the distribution hierarchy can also be located within different points of presence.

68. Further the edge server that received the request can serve the content to an end user acting as a proxy for the other edge server.

69. Also, on information and belief, the edge server that receives the request can query more than one edge server in overlapping time.

70. On information and belief, both Akamai and XO make and use infringing systems with respect to each of the acts of infringement described above. Akamai makes infringing systems that consist entirely of Akamai servers. Likewise on information and belief, XO makes and uses infringing systems where all of the servers are XO-hosted or operated servers. Further, when the infringing system includes a combination of Akamai and XO-hosted or operated servers, Akamai and XO act jointly or in concert to practice the claimed inventions, and the infringement is not complete until both Akamai and XO have provided or performed their respective parts.

71. The making, and operation, of the 002 Infringing Products as described above constitutes infringement of at least the above-mentioned claims of the 002 Patent pursuant to 35 U.S.C. § 271(a).

72. Unless enjoined by this Court, Akamai will continue to infringe the 002 Patent.

73. As a result of Akamai's conduct, Limelight has suffered and will continue to suffer irreparable injury, for which it has no adequate remedy at law. Limelight has also suffered damages as a result of Akamai's infringement of the 002 Patent and will continue to suffer such damages, until an injunction issues, in an amount and manner yet to be determined.

COUNT IV AGAINST AKAMAI:

INFRINGEMENT OF U.S. PATENT NO. 8,856,263

74. Limelight incorporates and realleges paragraphs 1-33 above as if fully set forth herein.

75. On information and belief, Akamai has infringed and continues to infringe one or more claims of the 263 Patent, including but not limited to claims 1, 2, 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, and 20 pursuant to 35 U.S.C. § 271(a), literally or under the doctrine of equivalents, by making, using, selling, and/or offering to sell in the United States without authority and/or importing into the United States without authority, the Akamai Intelligent Platform, including a content delivery network with edge servers performing prefetching additional web pages and content to cache, prior to their being requested by an end user, as well as services associated therewith, also known as the "Akamai Instant" feature (the 263 Infringing Products). Based on information and belief, and publicly available documentation, Akamai's edge servers accelerate delivery of web content by parsing requested web pages to identify additional web pages that are likely to be requested by a user system, and storing them to cache.

76. Specifically, Akamai's edge servers with the Akamai Instant feature meet the requirements of the claimed systems and methods for accelerating access to resources of web pages, as reflected by publicly available Akamai documentation. On information and belief, Akamai's edge servers include a cache for storing web content that can be used to store web content that has been "prefetched"—obtained before an end user client has asked for them. For example, Akamai states the following about the prefetching capabilities of its edge servers:

> Before a base page (e.g. home page html) is served from the origin to the client, the Akamai edge server parses the content and prefetches predefined assets from the origin before the response is sent to the client, so they can be served from the edge cache when the client requests them.

"Of Preconnect, Prefetch and Preload," https://community.akamai.com/community/web-performance/blog/2015/09/24/of-preconnect-prefetch-and-preload.

77. Akamai's edge servers with the Akamai Instant feature can parse requested web pages to identify additional web pages that are likely to be requested by the user system, to

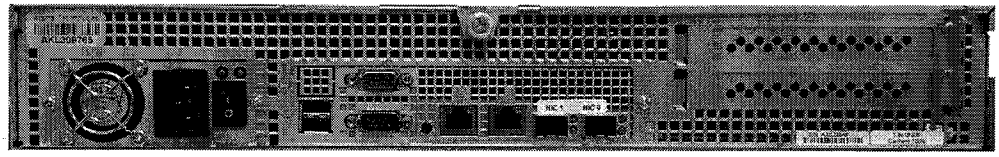
request those web pages and their specific content resources from another server before an end user requests them from the edge server, and store them in its cache. On information and belief, this ability includes the ability to obtain and store in cache static (non-dynamic) resources, such as image files. For example, Akamai states the following about the prefetching capabilities of its edge servers with Akamai Instant:

> In the past, when our customers have had long think-time applications due to database lookups, Web services calls, or other processing components that slow down origin response times, there wasn't much we could do to help other than speed the content once it was ready to be delivered. But by then it is usually too late. **The new Terra Alta feature, Akamai Instant, now lets us tackle that delivery challenge head on. By designating the most likely next pages to be visited by users, Terra Alta is able to start the process of gathering content, making Web service calls, or doing database lookups, before the page is requested by the user, and pre-fetching that content to the edge of the Internet, close to users, prior to the user requesting it.** We've seen this improve the performance of these applications by up to 100% over origin delivery.

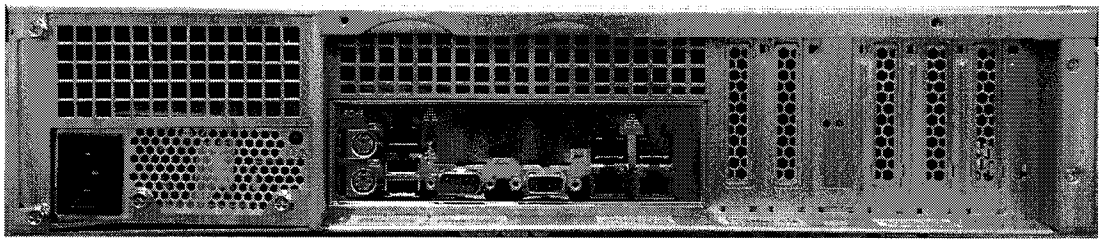
(emphasis added). "A Few More Tricks From Terra Alta," https://blogs.akamai.com/2012/03/a-few-more-tricks-from-terra-alta.html (last visited November 29, 2015).

78. For example, edge servers with Akamai Instant can prefetch web resources identified in a first web page with the <a> or <link> HTML elements, which can include additional web pages, as described at https://community.akamai.com/community/web-performance/blog/2015/09/24/of-preconnect-prefetch-and-preload.

79. As deployed and operated, as for example by XO in its data centers, Akamai edge servers with Akamai Instant include interfaces that enable communication of one or more user nodes with one or more web servers. For example, Akamai publishes the following images showing Akamai servers having Ethernet ports:



Akamai Hands And Eyes Guide, *available at* https://fieldtech.akamai.com/heguide/images/Server_Hardware/X4i_1-5x18_10G_Rear_Large.jpg (last visited November 29, 2015).



Akamai Hands And Eyes Guide, *available at* https://fieldtech.akamai.com/heguide/images/Server_Hardware/X1_2x8_CacheH_Rear_Large.jpg (last visited November 29, 2015). Further information about Akamai's deployed network interfaces can be found in Akamai Hands And Eyes Guide, *available at* https://fieldtech.akamai.com/heguide/Network_Packages.html (last visited November 29, 2015).

80. Further, Akamai's geographically distributed deployments of its edge servers, as deployed and operated for example by XO in its data centers, include routers that "allow Akamai to direct traffic between Akamai's equipment and the providers that Akamai connects to." Akamai Hands And Eyes Guide, *available at* https://fieldtech.akamai.com/heguide/Router_Hardware.html. When such equipment is deployed in a manner that connects Akamai's edge servers to the Internet, for example by XO in its data centers, it likewise has multiple ports configured to send and receive data over a connecting network.

81. On information and belief, because Akamai edge servers with Akamai Instant are designed and intended to respond to repeated requests for web content, both from the same end user device, and from different end user devices, these servers are able to perform the described prefetching functions for additional requests from end users, including where common resources are shared between pages.

82. The making, and operation, of Akamai edge servers with Akamai Instant as described above constitutes infringement of at least the above-mentioned claims of the 263 Patent pursuant to 35 U.S.C. § 271(a).

83. Unless enjoined by this Court, Akamai will continue to infringe the 263 Patent.

84. As a result of Akamai's conduct, Limelight has suffered and will continue to suffer irreparable injury, for which it has no adequate remedy at law. Limelight has also suffered damages as a result of Akamai's infringement of the 263 Patent and will continue to suffer such damages, until an injunction issues, in an amount and manner yet to be determined.

COUNT V AGAINST AKAMAI:

INFRINGEMENT OF U.S. PATENT NO. 9,015,348

85. Limelight incorporates and realleges paragraphs 1-33 above as if fully set forth herein.

86. On information and belief, Akamai has infringed and continues to infringe one or more claims of the 348 Patent, including but not limited to claims 1, 2, 3, 7, 10-16, and 18 pursuant to 35 U.S.C. § 271(a), literally or under the doctrine of equivalents, by making, using, selling, and/or offering to sell in the United States without authority and/or importing into the United States without authority, the Akamai Intelligent Platform, including a content delivery network with edge servers that perform automated front end optimization ("FEO"). Based on information and belief, and publicly available documentation, Akamai's edge servers perform

front end optimization by using attributes associated with content requests to select a set of content acceleration techniques that will be applied to deliver the requested content, and utilize performance metrics obtained regarding the delivered content to dynamically improve the subsequent selection of content acceleration techniques for similar content.

87. Specifically, Akamai's edge servers with automated front end optimization meet the requirements of the claimed systems and method for dynamically selecting from among a plurality of acceleration techniques implemented in a Content Delivery Network (CDN) using attributes associated with content requests, as reflected by publicly available Akamai documentation.

88. Based on information and belief, and publicly available documentation, Akamai's Accused 348 Products can apply numerous techniques to accelerate the delivery of digital content to end users, including: combining, compressing, rewriting or otherwise "minifying" Javascript and CSS elements in web pages; optimizing (including compressing) image and other media files; running Javascript asynchronously; and reordering web resource delivery. As Akamai describes:

> Front-end optimization reduces the number of requests, makes responses smaller, and reorders things to optimize rendering in the browser. There are dozens of different FEO optimization methods available in our service and the list continues to grow. FEO can reduce the number of requests by combining multiple JavaScript or CSS files into one download and by embedding small images directly into CSS. FEO can make responses smaller by minifying JavaScript and CSS, and by optimizing images. FEO also can unblock rendering of your page by running JavaScript asynchronously. Images can be made to load on demand, only as they scroll into view.

"FEO Fundamentals," *available at* https://developer.akamai.com/stuff/FEO/index.html (last visited November 29, 2015).

89. Other content acceleration techniques the Accused 348 Products perform include prefetching web content, optimizing TCP connections, caching the static portions of dynamically

rendered web pages (a feature known as EdgeStart), file versioning, domain sharding, and DNS prefetching. These and other content acceleration techniques the Accused 348 Products perform are described in Akamai public documentation, including at https://www.akamai.com/jp/ja/multimedia/documents/white-paper/front-end-optimization-on-the-akamai-intelligent-platform-white-paper.pdf (last visited November 29, 2015).

90. On information and belief, and as described in Akamai public documentation, the Accused 348 Products apply content acceleration techniques to requested content selectively, based in part on configuration files that are maintained by Akamai and its customers:

> Akamai's edge servers are responsible for processing end user requests and serving the requested content, as well as for acting as intermediaries in our overlay network. The platform offers a rich set of functionality and content-handling features, developed over a decade of experience working with and supporting many of the most sophisticated websites and applications on the Internet. These controls not only ensure correct application behavior as experienced by the end user, but also optimize the performance of Applications under different scenarios.
>
> An important feature of the edge server platform is its tremendous configurability via metadata configuration, which allows enterprises to retain fine-grained control in applying the platform's various capabilities to the handling of their content.

"The Akamai Network: A Platform for High-Performance Internet Applications," *available at* https://www.akamai.com/us/en/multimedia/documents/technical-publication/the-akamai-network-a-platform-for-high-performance-internet-applications-technical-publication.pdf (last visited November 29, 2015).

91. On information and belief, and as described in Akamai public documentation, the Accused 348 Products match attributes of content requests—such as the URL path, or header data in the request, or other attributes of the request such as end-user location or device type—to configuration data, to selectively apply content acceleration techniques to requested content:

> The metadata system allows these features to be separately configured based on matching request and response attributes. While the simplest matches are on URL

> path components, file extensions, and request methods, more advanced metadata matches can change behavior based on attributes including end-user geographic location, connection speed, HTTP request and response headers, cookie values, and many others....Metadata configuration can be set across an entire website, a portion of the site, a specific category of content, or even for individual files.

"The Akamai Network: A Platform for High-Performance Internet Applications," *available at* https://www.akamai.com/us/en/multimedia/documents/technical-publication/the-akamai-network-a-platform-for-high-performance-internet-applications-technical-publication.pdf (last visited November 29, 2015).

92. On information and belief, and as described in Akamai public documentation, the Accused 348 Products obtain metrics regarding the performance of content acceleration techniques applied to specific content requests, and use those metrics to improve the selection and configuration of acceleration techniques that will subsequently be used for similar requests. For example, Akamai's public documentation describes its "automated FEO solution" as follows:

> For every end user request, Akamai's proven technologies are dynamically applied in a way that optimizes performance for that unique scenario, taking into account real-time website, network, and end user conditions. Akamai's FEO capabilities are an integrated part of these solutions, working in concert with our other performance, security, and availability offerings to deliver the best possible experience for every user, on every device, every time.

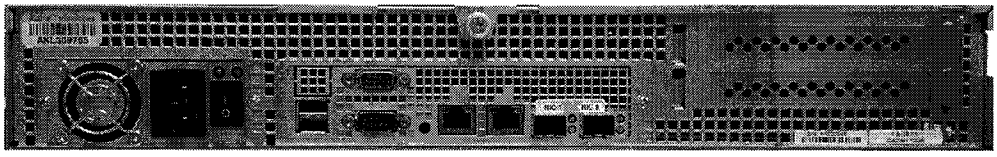
Front-End Optimization on the Akamai Intelligent Platform, https://www.akamai.com/jp/ja/multimedia/documents/white-paper/front-end-optimization-on-the-akamai-intelligent-platform-white-paper.pdf

93. The Accused 348 Products include hardware and software, such as a router, that provide an interface to a network. For example, Akamai's geographically distributed deployments of its edge servers, as deployed and operated for example by XO in its data centers,

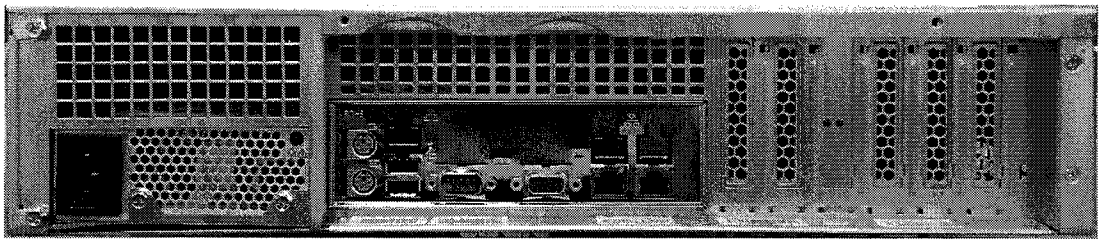
include routers that "allow Akamai to direct traffic between Akamai's equipment and the providers that Akamai connects to." [Akamai Hands And Eyes Guide], *available at* https://fieldtech.akamai.com/heguide/Router_Hardware.html. When such equipment is deployed in a manner that connects Akamai's edge servers to the Internet, for example by XO in its data centers, it is configured to receive requests from end users, such as from a device browser.

94. The Accused 348 Products include edge servers distributed throughout the United States and globally, such as those as deployed and operated by XO in its data centers. These edge servers include memory, storage devices, a processor, and interfaces to connect with a network interface, and to other edge servers and Akamai hardware and software located elsewhere in its content distribution network, and to apply selected content acceleration techniques as described above.

95. For example, on information and belief, to perform their basic role, Akamai's edge servers have multiple network ports to send and receive data. As a further example, Akamai publishes the following images showing Akamai servers having two Ethernet ports:



Akamai Hands And Eyes Guide, *available at* https://fieldtech.akamai.com/heguide/images/Server_Hardware/X4i_1-5x18_10G_Rear_Large.jpg (last visited November 29, 2015).



Akamai Hands And Eyes Guide, *available at* https://fieldtech.akamai.com/heguide/images/Server_Hardware/X1_2x8_CacheH_Rear_Large.jpg (last visited November 29, 2015). Further information about Akamai's deployed network interfaces can be found in Akamai Hands And Eyes Guide, *available at* https://fieldtech.akamai.com/heguide/Network_Packages.html (last visited November 29, 2015). Likewise, Akamai's servers include processors:

Server Generations

The following server generations are listed from most recently deployed to no longer being deployed. This a brief overview of each generation's specifications.

Generation	Specifications (Partial List)	
X8	Intel Broadwell DE	
X7	Intel Skylake	Intel Haswell E3-1200 V5 series, up to 64 GB Unbuffered DDR4
X6m	Intel Haswell E5-2600 V3 series, 16G to 256G DDR4 Registered	
X6	Intel Haswell E3-1200 V3 series, 16G or 32G DDR3	
X5	Intel Sandy Bridge E5 2600 EP series, 128GB or 256GB DDR3	
X4i	Intel Ivy Bridge E3-1270 V2, 16G DDR3	
X4	Intel Sandy Bridge E3-1270, 16G DDR3	
X2	Intel Nehalem X3470, 8G to 32G DDR3	
X1	Intel Nehalem X3470, 8G DDR3	
G10	AMD Opteron 6204 Quad Core, 64G DDR3	
G9	AMD Opteron 4184 Hexa Core, 8G DDR3	
G7	AMD Opteron 1389 Quad Core, 8G DDR2	
G6	AMD Opteron 2381 Quad Core, 8G DDR2	
G5	AMD Athlon II 240e Dual Core, 8G DDR3	
G4	AMD Opteron 1218 Dual Core, 4G DDR2	
G3	AMD Opteron 244, 2G or 4G M DDR1	
G2	AMD Opteron 244, 1G or 2G DDR1	
G1	AMD Opteron 244, 512M DDR1	

Akamai Hands And Eyes Guide, *available at* https://fieldtech.akamai.com/heguide/Server_Hardware.html (last visited November 29, 2015).

96. The making, and operation, of the Accused 348 products as described above constitutes infringement of at least the above-mentioned claims of the 348 Patent pursuant to 35 U.S.C. § 271(a).

97. Unless enjoined by this Court, Akamai will continue to infringe the 348 Patent.

98. As a result of Akamai's conduct, Limelight has suffered and will continue to suffer irreparable injury, for which it has no adequate remedy at law. Limelight has also suffered damages as a result of Akamai's infringement of the 348 Patent and will continue to suffer such damages, until an injunction issues, in an amount and manner yet to be determined.

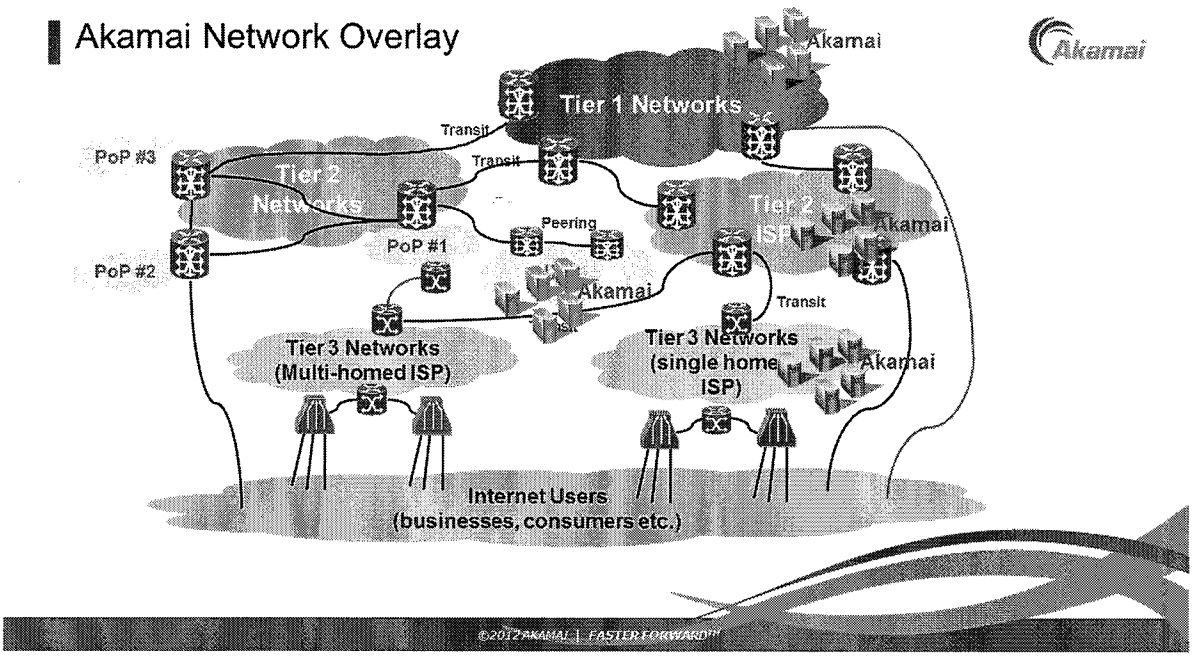
COUNT VI AGAINST AKAMAI:

INFRINGEMENT OF U.S. PATENT NO. 8,615,577

99. Limelight incorporates and realleges paragraphs 1-33 above as if fully set forth herein.

100. On information and belief, Akamai has infringed and continues to infringe one or more claims of the 577 Patent, including but not limited to claims 1, 2, 3, 4, 5, 6, 8, 9, 11, 16, and 19, pursuant to 35 U.S.C. § 271(a), literally or under the doctrine of equivalents, by making, using, selling, and/or offering to sell in the United States without authority and/or importing into the United States without authority, Akamai's Image Converter and Image Manager products, as well as Akamai's cloud-based video transcoding products (the 577 Infringing Products).

101. The 577 Infringing Products constitute parts of the Akamai content delivery network, which Akamai illustrates as follows:



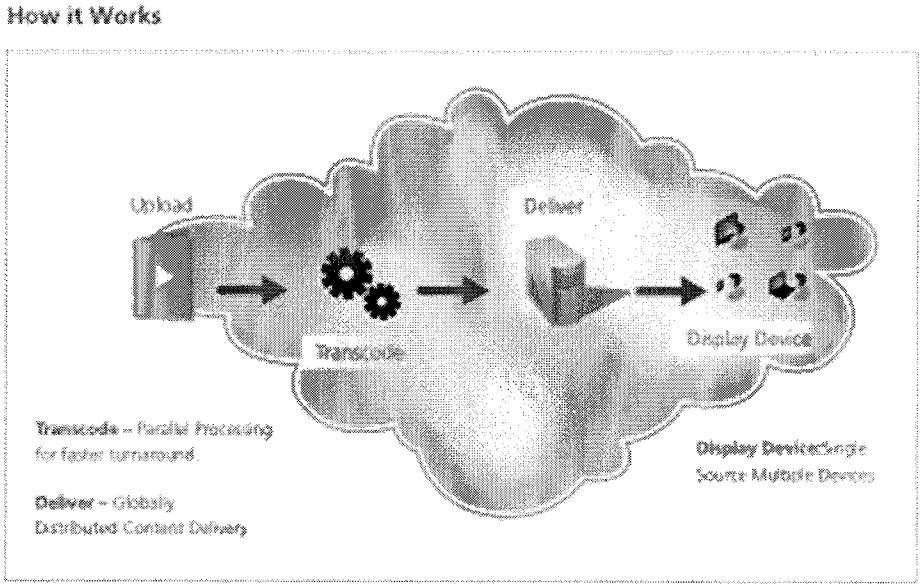
"Object Delivery," available at https://developer.akamai.com/stuff/Content_Delivery/Object_Delivery.html (last visited November 29, 2015).

102. On information and belief, and from publicly available Akamai documentation, the 577 Infringing Products allow Akamai's customers to upload image and video content (a process known as "ingest"). When the content is uploaded, it is determined (based on criteria that can be set by the customer) whether policies, that can be defined or customized by Akamai's customers, apply to the ingested content. When the policies apply, they determine what kind of processing will be performed to the ingested content, such as video transcoding, or formatting and alteration of ingested images. The 577 Infringing Products maintain numerous processing functions that are matched with content by these (premade and customer-defined) policies. Moreover, the 577 Infringing Products maintain numerous policies that can be applied to different ingested content. These policies can be triggered based on the processing to be performed on the content, the location of the content itself, or both.

103. On information and belief, the application of these policies can be based on metadata (which can be stored in a database) of the content itself, information about the end user that subsequently requests the content, or information related to the provider of the content using specialized function calls that the patent refers to as "mutators." Once it is determined which policy applies to the ingested content, such as a video or image file, the appropriate processing, such as video transcoding, or image formatting and alteration, is selected for processing that content. As Akamai explains with respect to video transcoding:

> With Akamai, you simply set the initial configuration in the easy-to-use Luna Control Center and after that, the workflow is a completely automated process. Upload content to pre-defined watch folders and Akamai handles the rest. Whether you're processing one media file or 20,000, the same automated processes apply. You can also customize advanced transcoding parameters including number of renditions, video/audio bitrates, bitrate types (VBR/CBR), frame rate, keyframe rate, and resolution.

"Media Services On Demand Product Brief," *available at* https://www.akamai.com/us/en/multimedia/documents/product-brief/media-services-on-demand-product-brief.pdf (last visited November 29, 2015). Akamai illustrates its video transcoding services as as follows:

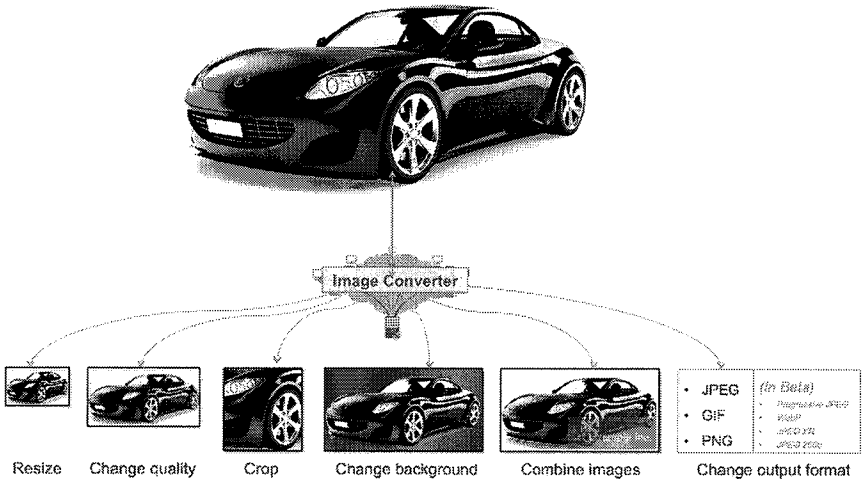


"Video On Demand Transcoding Product Brief," https://www.akamai.com/us/en/multimedia/documents/product-brief/vod-transcoding-product-brief.pdf (last visited November 29, 2015).

104. Akamai provides the following explanation of Image Converter capabilities:

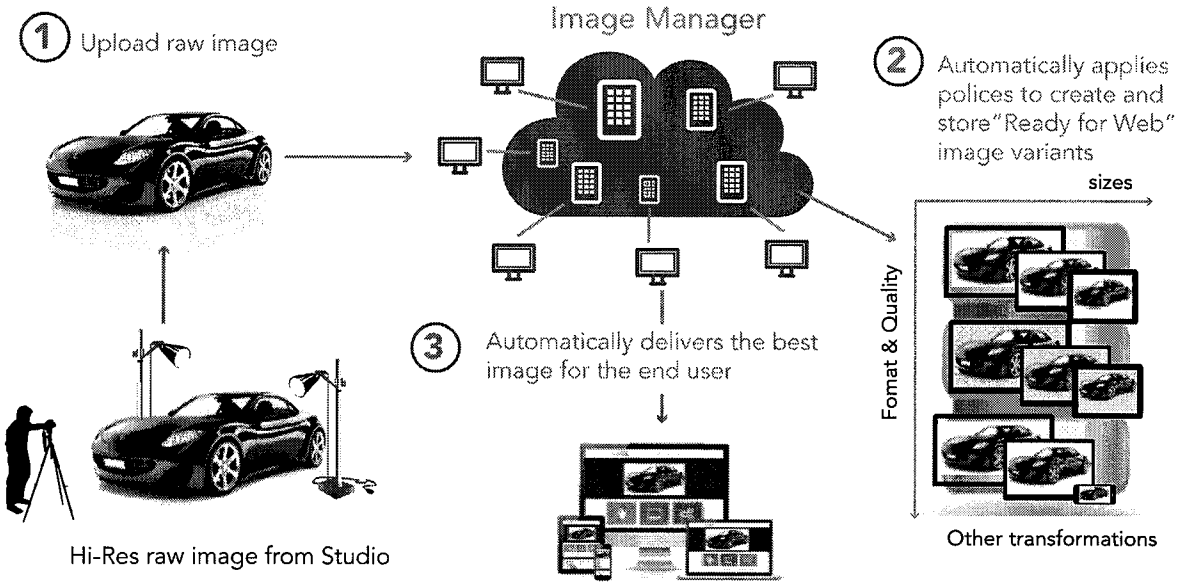
> Image Converter supports real-time API commands including:
> - **Downsize** – reduce an image's dimensions.
> - **Resize** – scale images to a specific width and height.
> - **Crop** – crop, or cut out, a section of an image based on dimension and axis parameters.
> - **Change Output Quality** – compress JPEG images based on a 1 to 100 scale.
> - **Change Output Format** – change JPEG, PNG, GIF & TIFF images to a specific file type such as JPEG, PNG & GIF.
> - **Background Color** – set the background color for transparent images using HTML or Hex colors.
> - **Compose Images** – place an image in a specific location on top of another image e.g. for watermarking.

"Image Converter" *available at* https://www.akamai.com/us/en/solutions/intelligent-platform/cloudlets/image-converter.jsp (last visited November 29, 2015). Akamai illustrates Image Converter as follows:



Id.

105. As Akamai explains with respect to Image Manager: "Akamai provides developers with highly customizable policies to accommodate a wide range of image transformations. Begin with high-quality master images and quickly derive ready-for-web images that adapt to business, artistic and technical requirements." "Image Manager Product Brief," *available at* https://www.akamai.com/us/en/multimedia/documents/product-brief/image-manager-product-brief.pdf (last visited November 29, 2015). Akamai illustrates functionality of Image Manager as follows:



106. "Image Manager Product Brief," https://www.akamai.com/us/en/multimedia/documents/product-brief/image-manager-product-brief.pdf (last visited November 29, 2015).

107. In addition, on information and belief the policies can be triggered by function calls that are built into template URLs. As Akamai explains: "Image Converter harnesses the power of the Akamai Intelligent Platform™ to enable organizations to dynamically manipulate images in the cloud through appending application programming interface (API) commands to

image URLs." "Image Converter" *available at* https://www.akamai.com/us/en/solutions/intelligent-platform/cloudlets/image-converter.jsp (last visited November 29, 2015).

108. Likewise, as shown above, the functions that process the ingested content can be an HTTP-based application programming interface (API). *See id.*

109. The making, and operation, of the 577 Infringing Products as described above constitutes infringement of at least the above-mentioned claims of the 577 Patent pursuant to 35 U.S.C. § 271(a).

110. Unless enjoined by this Court, Akamai will continue to infringe the 577 Patent.

111. As a result of Akamai's conduct, Limelight has suffered and will continue to suffer irreparable injury, for which it has no adequate remedy at law. Limelight has also suffered damages as a result of Akamai's infringement of the 577 Patent and will continue to suffer such damages, until an injunction issues, in an amount and manner yet to be determined.

COUNT VII AGAINST XO:

INFRINGEMENT OF U.S. PATENT NO. 8,750,155

112. Limelight incorporates and realleges paragraphs 1-47 above as if fully set forth herein.

113. On information and belief, XO has infringed and continues to infringe one or more claims of the 155 Patent, including but not limited to claims 1, 3, 8, 9, 10, 11, 12, 13, 15, 18, 19, and 20, pursuant to 35 U.S.C. § 271(a), literally or under the doctrine of equivalents, by making, using, selling, and/or offering to sell in the United States without authority and/or importing into the United States without authority, hardware and software, content delivery servers and networks, and data centers that constitute or include 155 Infringing Products because of their inclusion and performance of the functionality described above with respect to Count I and the Akamai Intelligent Platform.

114. Specifically, XO provides services, networks, and data centers, that host servers that constitute 155 Infringing Products because they form all or part of CDNs including a plurality of points of presence that perform functionality related to the Akamai Intelligent Platform. These points of presence include edge servers that are operated or hosted by XO. Based on information and belief, and publicly available documentation, XO-operated or hosted edge servers perform TCP optimization by modifying pre-existing TCP settings based upon parameters that are determined at least in part with reference to information in the URLs of end-user requests as described above with respect to Count I including the specific variations described therein.

115. On information and belief, when XO assembles or configures a server, network, or data center that includes this functionality, and when it uses such server, network, or data center to provide services to its customers, these acts constitute acts of direct infringement of the 155 Patent for the same technical reasons explained above with respect to Count I except that in such instances XO is the direct infringer.

116. Both XO and Akamai make and use infringing systems. On information and belief, XO makes infringing systems that consist entirely of XO-hosted or operated servers. Likewise Akamai makes and uses infringing systems where all of the servers are Akamai servers. Further, when the infringing system includes a combination of XO and Akamai servers, Akamai and XO act jointly or in concert to perform the infringing acts, and the infringement is not complete until both XO and Akamai have provided or performed their respective parts.

117. Unless enjoined by this Court, XO will continue to infringe the 155 Patent.

118. As a result of XO's conduct, Limelight has suffered and will continue to suffer irreparable injury, for which it has no adequate remedy at law. Limelight has also suffered

damages as a result of XO's infringement of the 155 Patent and will continue to suffer such damages, until an injunction issues, in an amount and manner yet to be determined.

COUNT VIII AGAINST XO:

INFRINGEMENT OF U.S. PATENT NO. 7,715,324

119. Limelight incorporates and realleges paragraphs 1-33 and 48-62 above as if fully set forth herein.

120. On information and belief, XO has infringed and continues to infringe one or more claims of the 324 Patent, including but not limited to claims 1, 2, 4, 5, 6, 7, 8, 10, and 11, pursuant to 35 U.S.C. § 271(a), literally or under the doctrine of equivalents, by making, using, selling, and/or offering to sell in the United States without authority and/or importing into the United States without authority, hardware and software, content delivery servers and networks, and data centers that constitute or include 324 Infringing Products because of their inclusion and performance of the functionality described above with respect to Count II and the Akamai Intelligent Platform.

121. Specifically, XO provides services, networks, and data centers, which host servers that constitute 324 Infringing Products because they form all or part of CDNs including a plurality of points of presence that perform functionality related to the Akamai Intelligent Platform. These points of presence include edge servers that are operated or hosted by XO. Based on information and belief, and publicly available documentation, XO-operated or hosted edge servers perform TCP optimization by modifying pre-existing TCP settings based upon parameters that are determined at least in part with reference to information in the URLs of end-user requests as described above with respect to Count II including the specific variations described therein.

122. On information and belief, when XO assembles or configures a server, network, or data center that includes this functionality, and when it uses such server, network, or data center to provide services to its customers, these acts constitute acts of direct infringement of the 324 Patent for the same technical reasons explained above with respect to Count II except that in such instances XO is the direct infringer.

123. Unless enjoined by this Court, XO will continue to infringe the 324 Patent.

124. As a result of XO's conduct, Limelight has suffered and will continue to suffer irreparable injury, for which it has no adequate remedy at law. Limelight has also suffered damages as a result of XO's infringement of the 324 Patent and will continue to suffer such damages, until an injunction issues, in an amount and manner yet to be determined.

COUNT IX AGAINST XO:

INFRINGEMENT OF U.S. PATENT NO. 8,683,002

125. Limelight incorporates and realleges paragraphs 1-33 and 63-73 above as if fully set forth herein.

126. On information and belief, XO has infringed and continues to infringe one or more claims of the 002 Patent, including but not limited to claims 1, 2, 3, 4, 5, 7, 8, 9, 10, 13, 15, 16, 17, 18, and 20, pursuant to 35 U.S.C. § 271(a), literally or under the doctrine of equivalents, by making, using, selling, and/or offering to sell in the United States without authority and/or importing into the United States without authority, hardware and software, content delivery servers and networks, and data centers that constitute or include 002 Infringing Products because of their inclusion and performance of the functionality described above with respect to Count III and the Akamai Intelligent Platform.

127. Specifically, XO provides services, networks, and data centers, that host servers that constitute 002 Infringing Products because they form all or part of CDNs including a

plurality of points of presence that are distributed geographically and perform functionality related to the Akamai Intelligent Platform. These points of presence include edge servers that are operated or hosted by XO. Based on information and belief, and publicly available documentation, XO-operated or hosted edge servers that do not have user-requested content in their own caches can ask other edge servers whether they have the requested content in their caches and if so the content is provided to the user, including the specific variations described above with respect to Count III.

128. On information and belief, when XO assembles or configures a server, network, or data center that includes this functionality, and when it uses such server, network, or data center to provide services to its customers, these acts constitute acts of direct infringement of the 002 Patent for the same technical reasons explained above with respect to Count III except that in such instances XO is the direct infringer.

129. Both XO and Akamai make and use infringing systems. On information and belief, XO makes infringing systems that consist entirely of XO-hosted or operated servers. Likewise Akamai makes and uses infringing systems where all of the servers are Akamai servers. Further, when the infringing system includes a combination of XO and Akamai servers, Akamai and XO act jointly or in concert to perform the infringing acts, and the infringement is not complete until both XO and Akamai have provided or performed their respective parts.

130. Unless enjoined by this Court, XO will continue to infringe the 002 Patent.

131. As a result of XO's conduct, Limelight has suffered and will continue to suffer irreparable injury, for which it has no adequate remedy at law. Limelight has also suffered damages as a result of XO's infringement of the 002 Patent and will continue to suffer such damages, until an injunction issues, in an amount and manner yet to be determined.

COUNT X AGAINST XO:

INFRINGEMENT OF U.S. PATENT NO. 8,856,263

132. Limelight incorporates and realleges paragraphs 1-33 and 74-84 above as if fully set forth herein.

133. On information and belief, XO has infringed and continues to infringe one or more claims of the 263 Patent, including but not limited to claims 1, 2, 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, and 20, pursuant to 35 U.S.C. § 271(a), literally or under the doctrine of equivalents, by making, using, selling, and/or offering to sell in the United States without authority and/or importing into the United States without authority, hardware and software, content delivery servers and networks, and data centers that constitute or include 263 Infringing Products because of their inclusion and performance of the functionality described above with respect to Count IV and the Akamai Intelligent Platform.

134. Specifically, XO provides services, networks, and data centers, that host servers that constitute 263 Infringing Products because they form all or part of CDNs including a plurality of points of presence that perform functionality related to the Akamai Intelligent Platform. These points of presence include edge servers that are operated or hosted by XO. Based on information and belief, and publicly available documentation, XO-operated or hosted edge servers perform prefetching of additional web pages and content to cache, prior to their being requested by an end user, as well as services associated therewith, also known as the "Akamai Instant" feature as described above with respect to Count IV including the specific variations described therein.

135. On information and belief, when XO assembles or configures a server, network, or data center that includes this functionality, and when it uses such server, network, or data center to provide services to its customers, these acts constitute acts of direct infringement of the

263 Patent for the same technical reasons explained above with respect to Count IV except that in such instances XO is the direct infringer.

136. Unless enjoined by this Court, XO will continue to infringe the 263 Patent.

137. As a result of XO's conduct, Limelight has suffered and will continue to suffer irreparable injury, for which it has no adequate remedy at law. Limelight has also suffered damages as a result of XO's infringement of the 263 Patent and will continue to suffer such damages, until an injunction issues, in an amount and manner yet to be determined.

COUNT XI AGAINST XO:

INFRINGEMENT OF U.S. PATENT NO. 9,015,348

138. Limelight incorporates and realleges paragraphs 1-33 and 85-98 above as if fully set forth herein.

139. On information and belief, XO has infringed and continues to infringe one or more claims of the 348 Patent, including but not limited to claims 1, 2, 3, 7, 10-16, and 18, pursuant to 35 U.S.C. § 271(a), literally or under the doctrine of equivalents, by making, using, selling, and/or offering to sell in the United States without authority and/or importing into the United States without authority, hardware and software, content delivery servers and networks, and data centers that constitute or include 348 Infringing Products because of their inclusion and performance of the functionality described above with respect to Count V and the Akamai Intelligent Platform.

140. Specifically, XO provides services, networks, and data centers, that host servers that constitute 348 Infringing Products because they form all or part of CDNs related to the Akamai Intelligent Platform including a content delivery network with edge servers that perform automated front end optimization ("FEO"). Based on information and belief, and publicly available documentation, XO-operated or hosted edge servers perform front end optimization by

using attributes associated with content requests to select a set of content acceleration techniques that will be applied to deliver the requested content, and utilize performance metrics obtained regarding the delivered content to dynamically improve the subsequent selection of content acceleration techniques for similar content as described above with respect to Count V including the specific variations described therein.

141. On information and belief, when XO assembles or configures a server, network, or data center that includes this functionality, and when it uses such server, network, or data center to provide services to its customers, these acts constitute acts of direct infringement of the 348 Patent for the same technical reasons explained above with respect to Count V except that in such instances XO is the direct infringer.

142. Unless enjoined by this Court, XO will continue to infringe the 348 Patent.

143. As a result of XO's conduct, Limelight has suffered and will continue to suffer irreparable injury, for which it has no adequate remedy at law. Limelight has also suffered damages as a result of XO's infringement of the 348 Patent and will continue to suffer such damages, until an injunction issues, in an amount and manner yet to be determined.

COUNT XII AGAINST XO:

INFRINGEMENT OF U.S. PATENT NO. 8,615,577

144. Limelight incorporates and realleges paragraphs 1-33 and 99-111 above as if fully set forth herein.

145. On information and belief, XO has infringed and continues to infringe one or more claims of the 577 Patent, including but not limited to claims 1, 2, 3, 4, 5, 6, 8, 9, 11, 16, and 19, pursuant to 35 U.S.C. § 271(a), literally or under the doctrine of equivalents, by making, using, selling, and/or offering to sell in the United States without authority and/or importing into the United States without authority, hardware and software, content delivery servers and

networks, and data centers that constitute or include 577 Infringing Products because of their inclusion and performance of the functionality described above with respect to Count VI and the Akamai Intelligent Platform.

146. Specifically, on information and belief, XO provides services, networks, and data centers, that host servers that constitute 577 Infringing Products because they form all or part of CDNs related to the Akamai Intelligent Platform including a content delivery network including the Image Converter, Image Manager, and cloud-based video transcoding products, which store and apply the claimed policies to ingested content in the various manners described above with respect to Count VI including the specific variations described therein.

147. On information and belief, when XO assembles or configures a server, network, or data center that includes this functionality, and when it uses such server, network, or data center to provide services to its customers, these acts constitute acts of direct infringement of the 577 Patent for the same technical reasons explained above with respect to Count VI except that in such instances XO is the direct infringer.

148. Unless enjoined by this Court, XO will continue to infringe the 577 Patent.

149. As a result of XO's conduct, Limelight has suffered and will continue to suffer irreparable injury, for which it has no adequate remedy at law. Limelight has also suffered damages as a result of XO's infringement of the 577 Patent and will continue to suffer such damages, until an injunction issues, in an amount and manner yet to be determined.

PRAYER FOR RELIEF

150. Limelight respectfully prays for relief as follows:

(a) A judgment that Akamai and XO have infringed and continue to infringe one or more claims of the Asserted Patents;

(b) A judgment awarding Limelight all damages adequate to compensate for Akamai's and XO's infringement, and in no event less than a reasonable royalty for Akamai's and XO's acts of infringement, including all pre-judgment and post-judgment interest at the maximum rate allowed by law;

(c) A permanent injunction enjoining Akamai, and its directors, officers, employees, attorneys, agents, and all persons in active concert or participation with any of the foregoing, from further acts of infringement of the Asserted Patents;

(d) A permanent injunction enjoining XO, and its directors, officers, employees, attorneys, agents, and all persons in active concert or participation with any of the foregoing, from further acts of infringement of the Asserted Patents; and

(e) A judgment awarding Limelight such other relief as the Court may deem just and equitable.

DEMAND FOR JURY TRIAL

Pursuant to Rule 38(b) of the Federal Rules of Civil Procedure, Plaintiff Limelight demands a trial by jury in this action.

Date: November 30, 2015

Respectfully submitted,

Maya M. Eckstein (Va. Bar No. 41413)
HUNTON & WILLIAMS LLP
951 E. Byrd St.
Richmond, Virginia 23219
Telephone: (804) 788-8788
Facsimile: (804) 343-4630
meckstein@hunton.com

Matthew D. Powers (CA Bar No. 104795) (*pro hac vice* motion to be filed)
Paul T. Ehrlich (Cal Bar No. 228543) (*pro hac vice* motion to be filed)
William P. Nelson (Cal Bar No. 196091) (*pro hac vice* motion to be filed)
Aaron M. Nathan (Cal Bar. No. 251316) (*pro hac vice* motion to be filed)
TENSEGRITY LAW GROUP, LLP
555 Twin Dolphin Drive, Suite 650
Redwood Shores, CA 94065
Telephone: (650) 802-6000
Facsimile: (650) 802-6001
matthew.powers@tensegritylawgroup.com
paul.ehrlich@tensegritylawgroup.com
william.nelson@tensegritylawgroup.com
aaron.nathan@tensegritylawgroup.com

Attorneys for Plaintiff Limelight Networks, Inc.

Exhibit A

US007715324B1

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

(10) **Patent No.: US 7,715,324 B1**
(45) **Date of Patent: May 11, 2010**

(54) **CONDITIONAL PROTOCOL CONTROL**

(75) Inventors: **Bradley B. Harvell**, Chandler, AZ (US); **Joseph D. DePalo**, Peoria, AZ (US); **Michael M. Gordon**, Paradise Valley, AZ (US); **Jason L. Wolfe**, Gilbert, AZ (US)

(73) Assignee: **Limelight Networks, Inc.**, Tempe, AZ (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **12/625,436**

(22) Filed: **Nov. 24, 2009**

Related U.S. Application Data

(63) Continuation of application No. 12/572,981, filed on Oct. 2, 2009, which is a continuation-in-part of application No. PCT/US2009/038361, filed on Mar. 26, 2009.

(30) **Foreign Application Priority Data**

Mar. 26, 2009 (AU) 2009201833

(51) **Int. Cl.**
H04J 1/16 (2006.01)
H04L 12/56 (2006.01)

(52) **U.S. Cl.** **370/252**; 370/389; 370/412; 370/466

(58) **Field of Classification Search** 370/252, 370/412, 389, 466
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,029,200 A 2/2000 Beckerman et al.
6,038,603 A 3/2000 Joseph
6,397,246 B1 5/2002 Wolfe
6,591,304 B1 7/2003 Sitaraman et al.
7,367,051 B1 4/2008 Siegrist et al.
7,480,254 B2 1/2009 Mayer
2003/0074482 A1 4/2003 Christensen et al.
2005/0097212 A1 5/2005 Engel et al.
2005/0210121 A1 * 9/2005 Taylor et al. 709/218

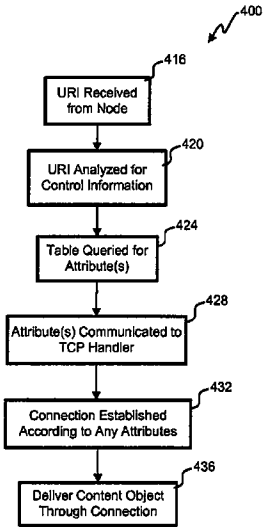
* cited by examiner

Primary Examiner—John Pezzlo
(74) *Attorney, Agent, or Firm*—Townsend and Townsend and Crew, LLP

(57) **ABSTRACT**

In one embodiment, a system for adapting the interoperation of nodes in an information processing system is disclosed. The system includes a protocol handler, a protocol attribute information store and a protocol attribute selector. The protocol handler manages a first connection and a second connection over the network using a protocol. The protocol attribute information store holds a plurality of attributes. The protocol attribute selector evaluates first information related to a first connection, utilizes the protocol attribute information store to determine first protocol attributes corresponding to the first connection, and communicates the first attributes for the first connection to the protocol handler. Additionally, the protocol attribute selector evaluates second information related to a second connection, utilizes the protocol attribute information store to determine second attributes corresponding to the second connection, and communicates the second attributes for the second connection the protocol handler. The protocol handler uses first attributes for the first connection and second attributes for the second connection.

11 Claims, 14 Drawing Sheets



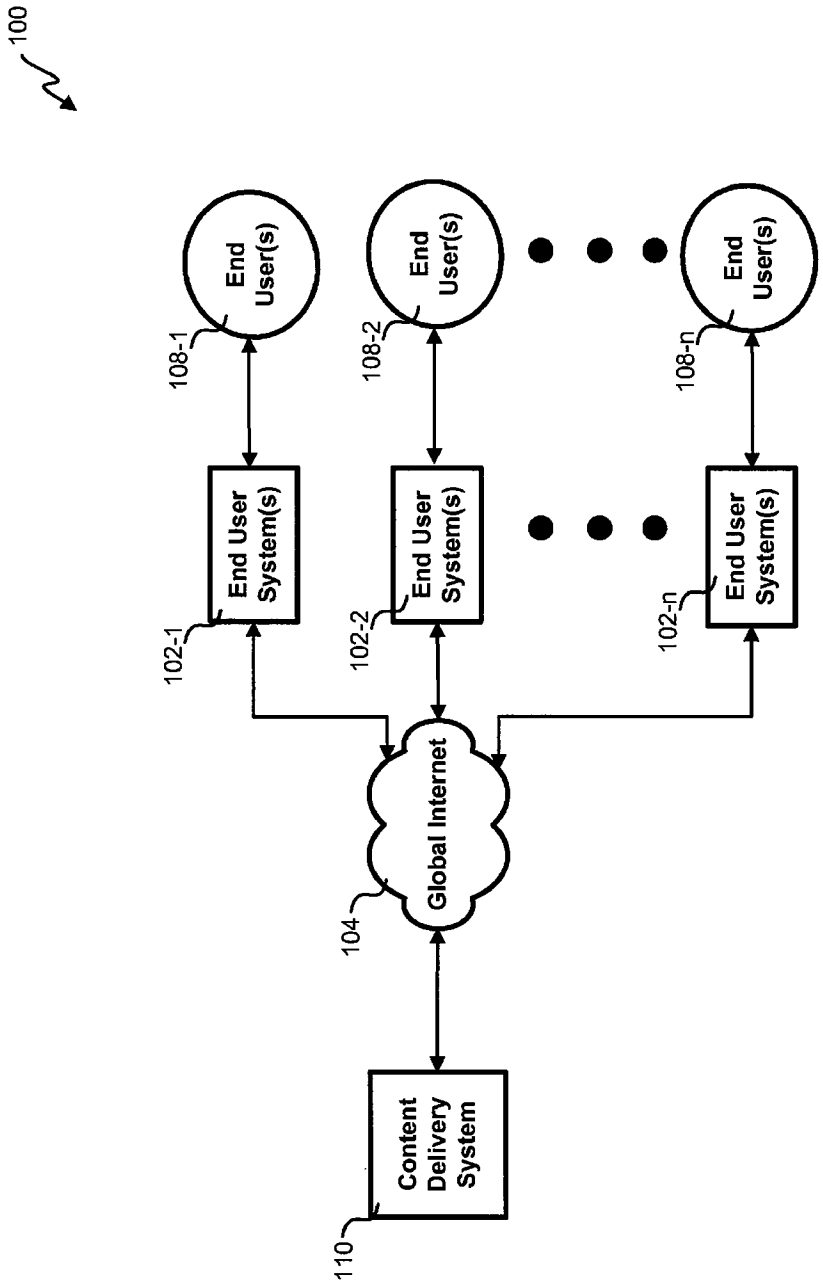

100
End User(s)
108-1
End User(s)
108-2
End User(s)
108-n
End User System(s)
102-1
End User System(s)
102-2
End User System(s)
102-n
Global Internet
104
Content Delivery System
110

FIG. 1

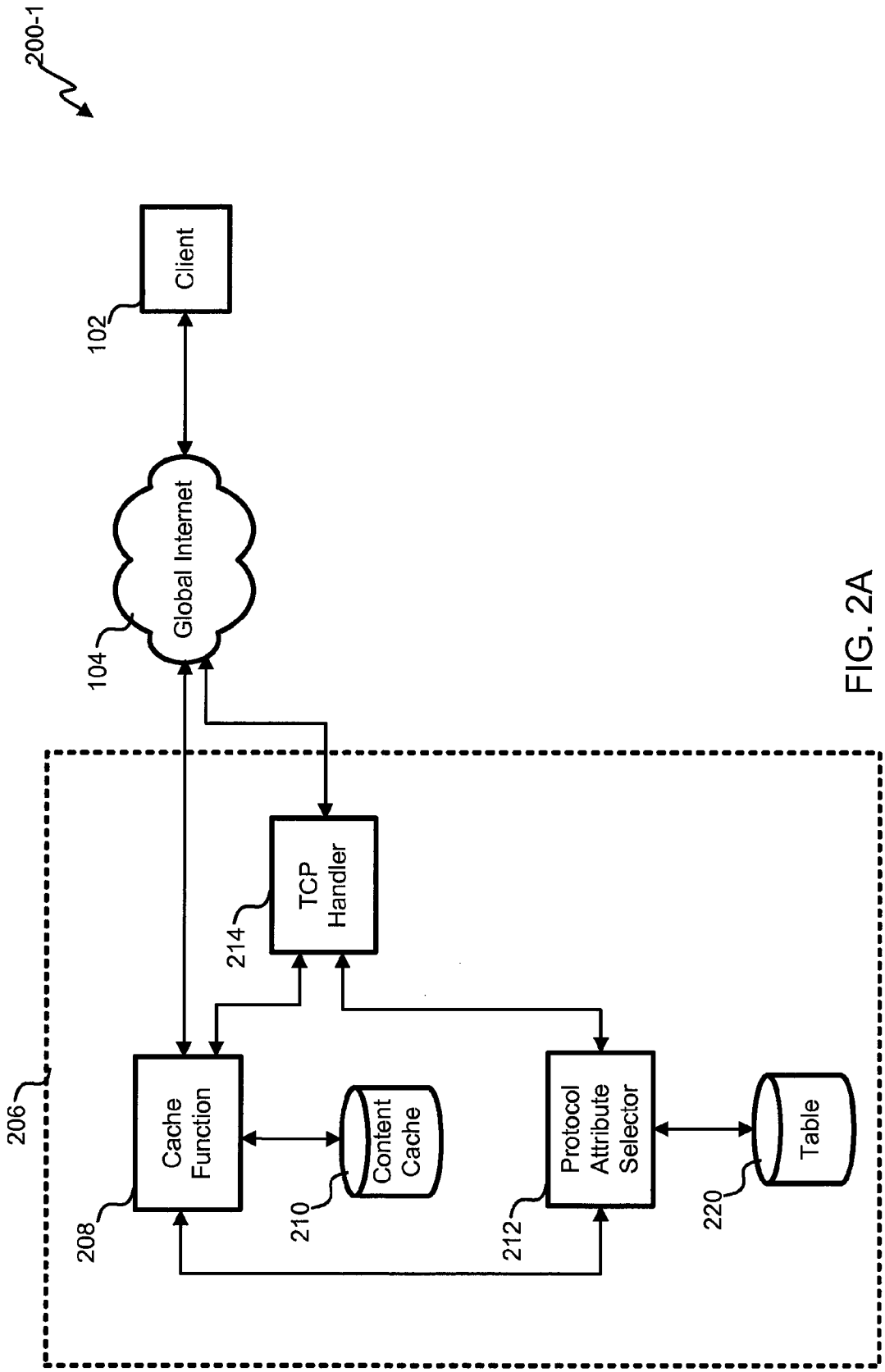

200-1
Client
102
Global Internet
104
TCP Handler
214
206
Cache Function
208
Content Cache
210
Protocol Attribute Selector
212
Table
220

FIG. 2A

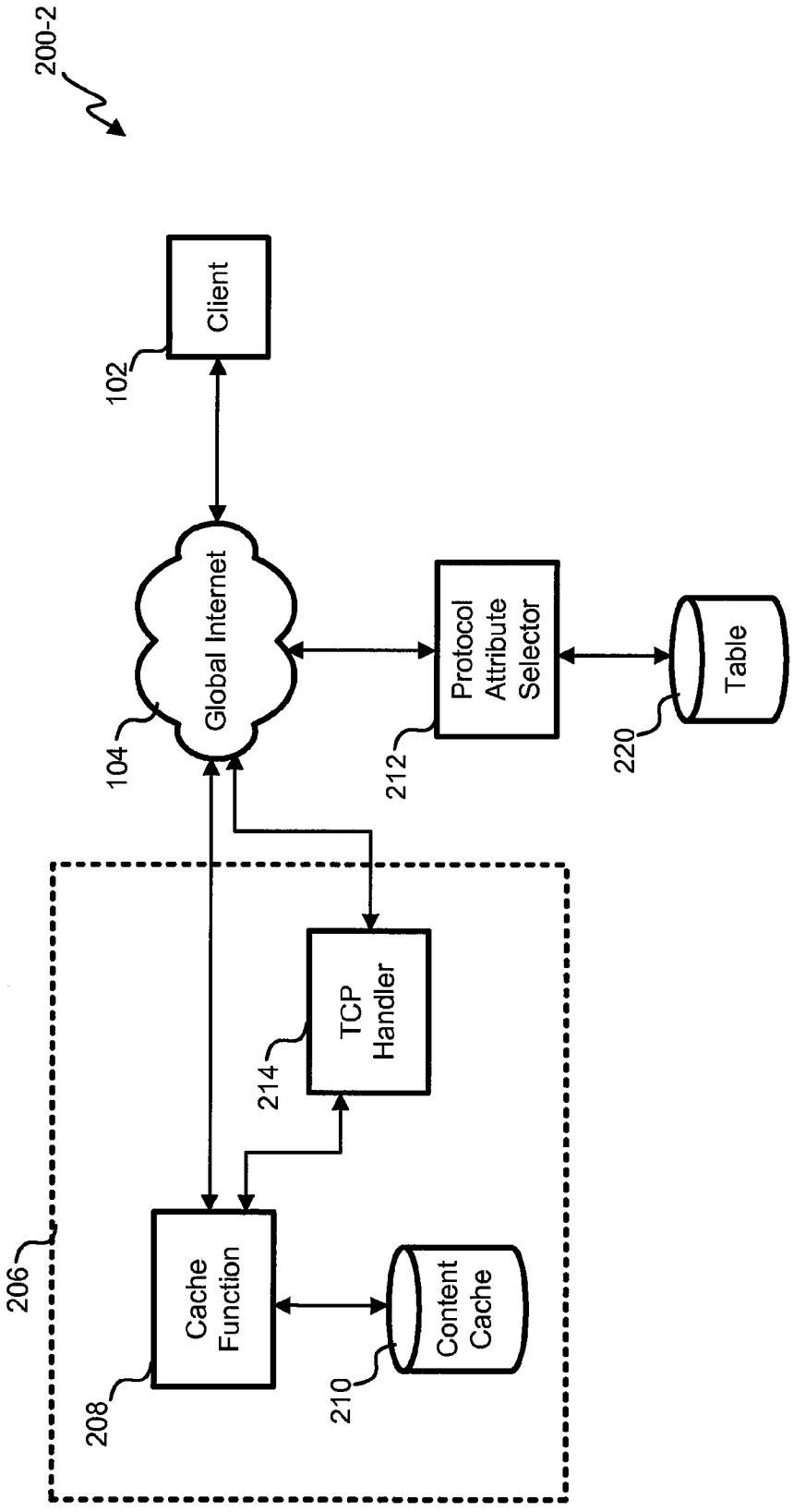

200-2
Client
102
Global Internet
104
Protocol Attribute Selector
212
Table
220
TCP Handler
214
206
Cache Function
208
Content Cache
210

FIG. 2B

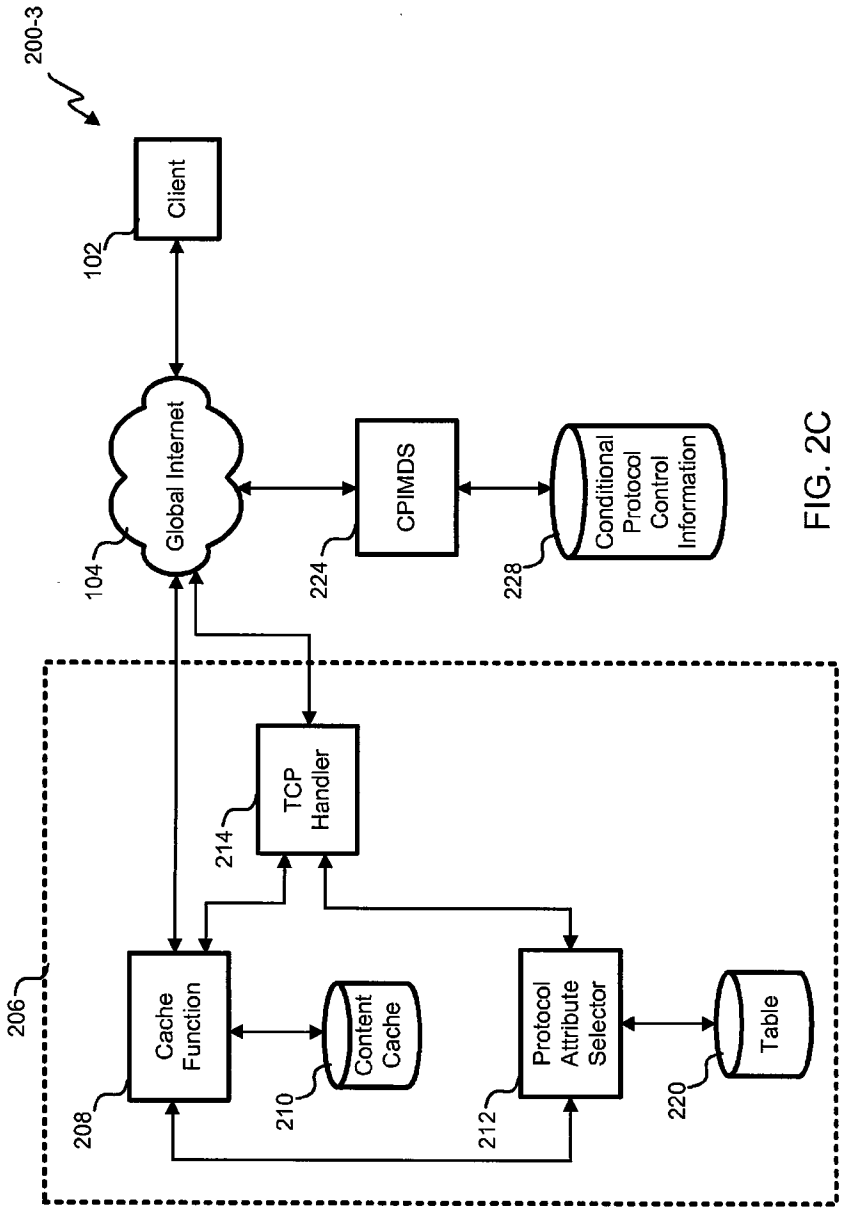

200-3
Client
102
Global Internet
104
CPIMDS
224
Conditional Protocol Control Information
228
TCP Handler
214
206
Cache Function
208
Content Cache
210
Protocol Attribute Selector
212
Table
220

FIG. 2C

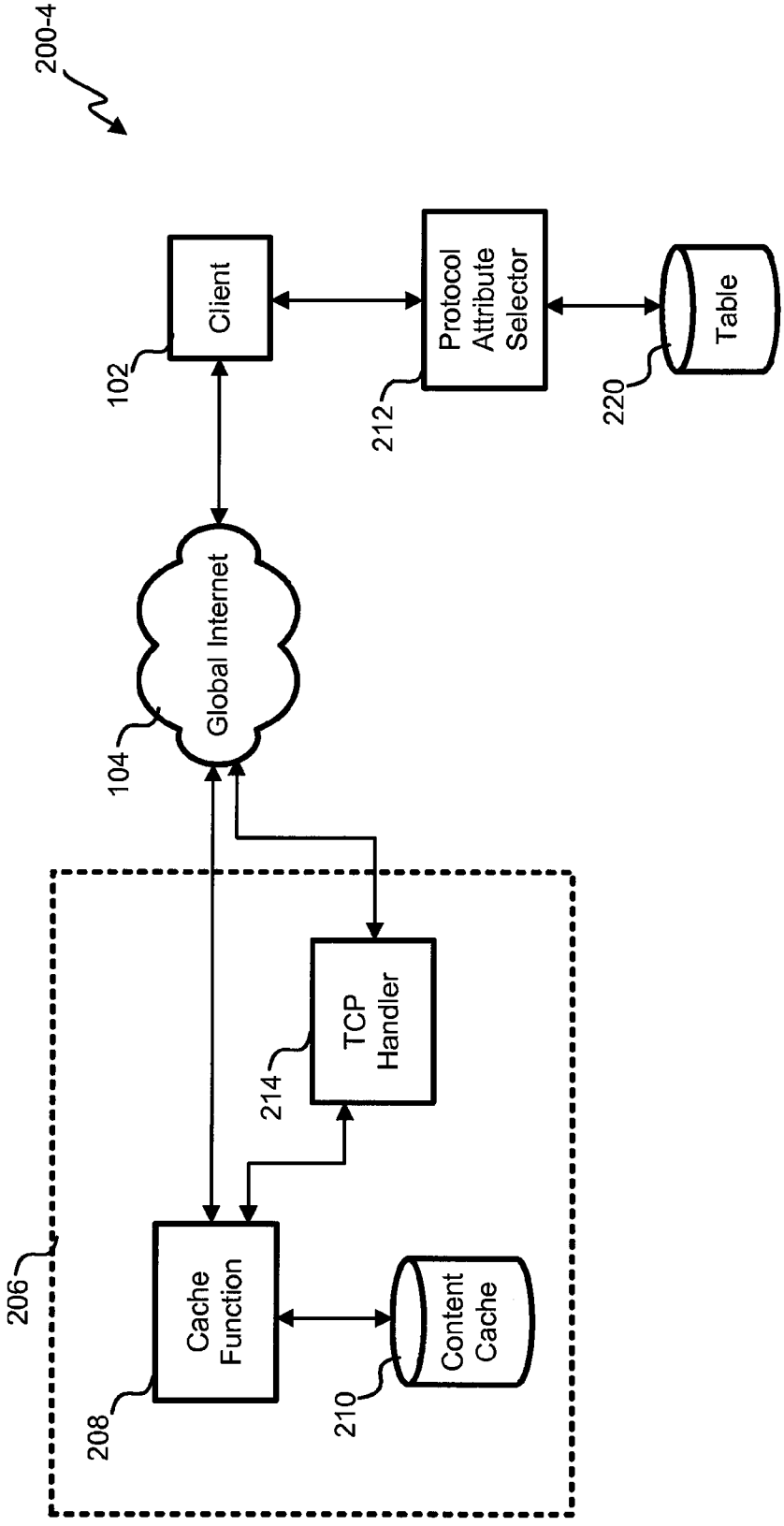

200-4
Client
102
Protocol Attribute Selector
212
Table
220
Global Internet
104
TCP Handler
214
206
Cache Function
208
Content Cache
210

FIG. 2D

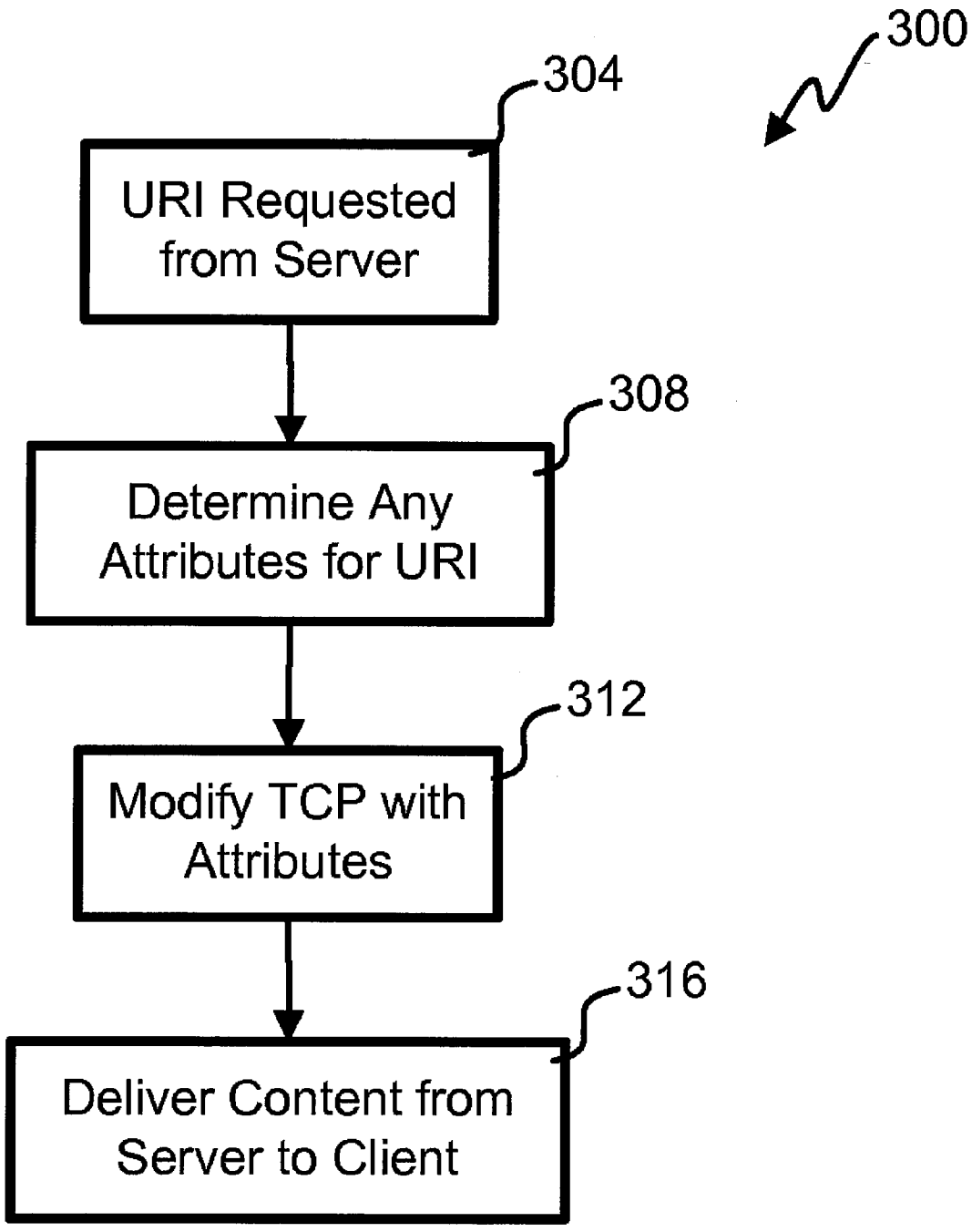

300
304
URI Requested from Server
308
Determine Any Attributes for URI
312
Modify TCP with Attributes
316
Deliver Content from Server to Client

Fig. 3

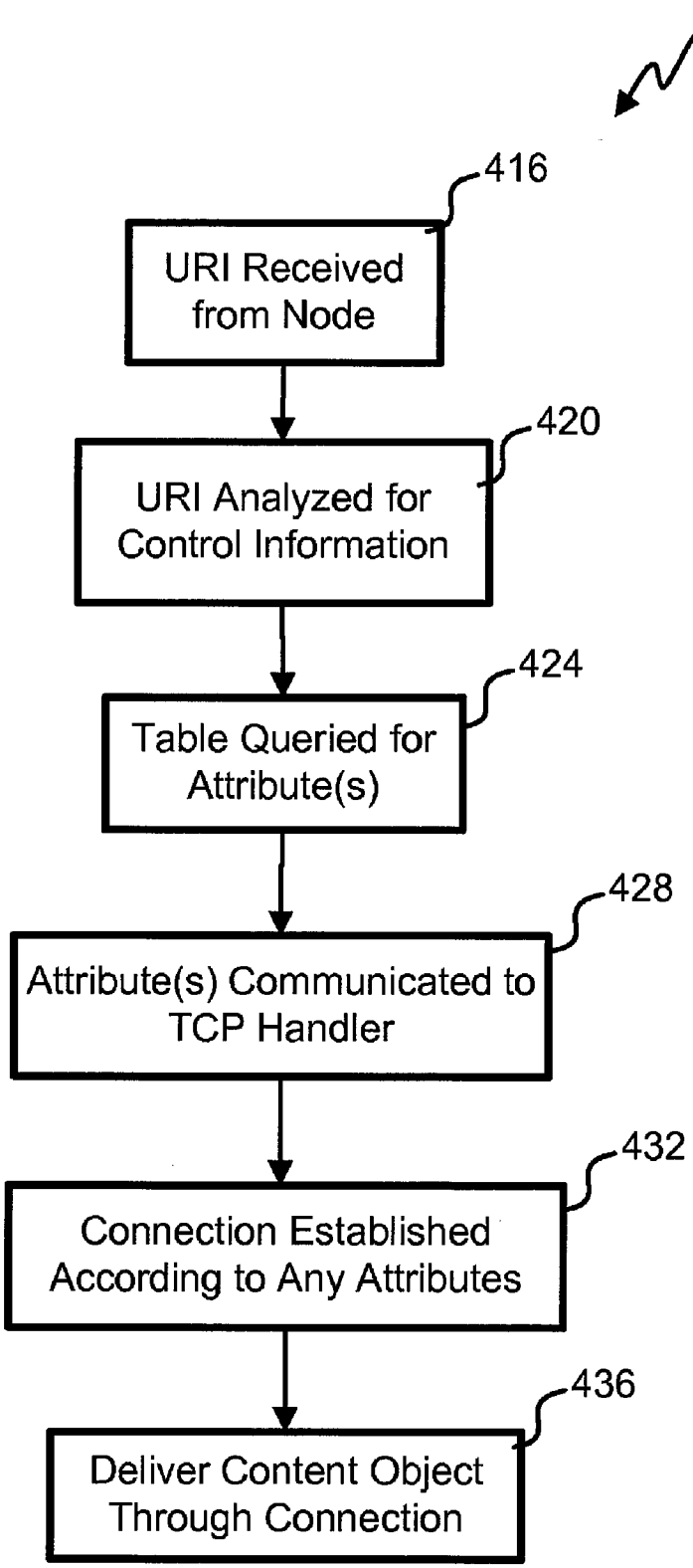

400
416
URI Received from Node
420
URI Analyzed for Control Information
424
Table Queried for Attribute(s)
428
Attribute(s) Communicated to TCP Handler
432
Connection Established According to Any Attributes
436
Deliver Content Object Through Connection

Fig. 4

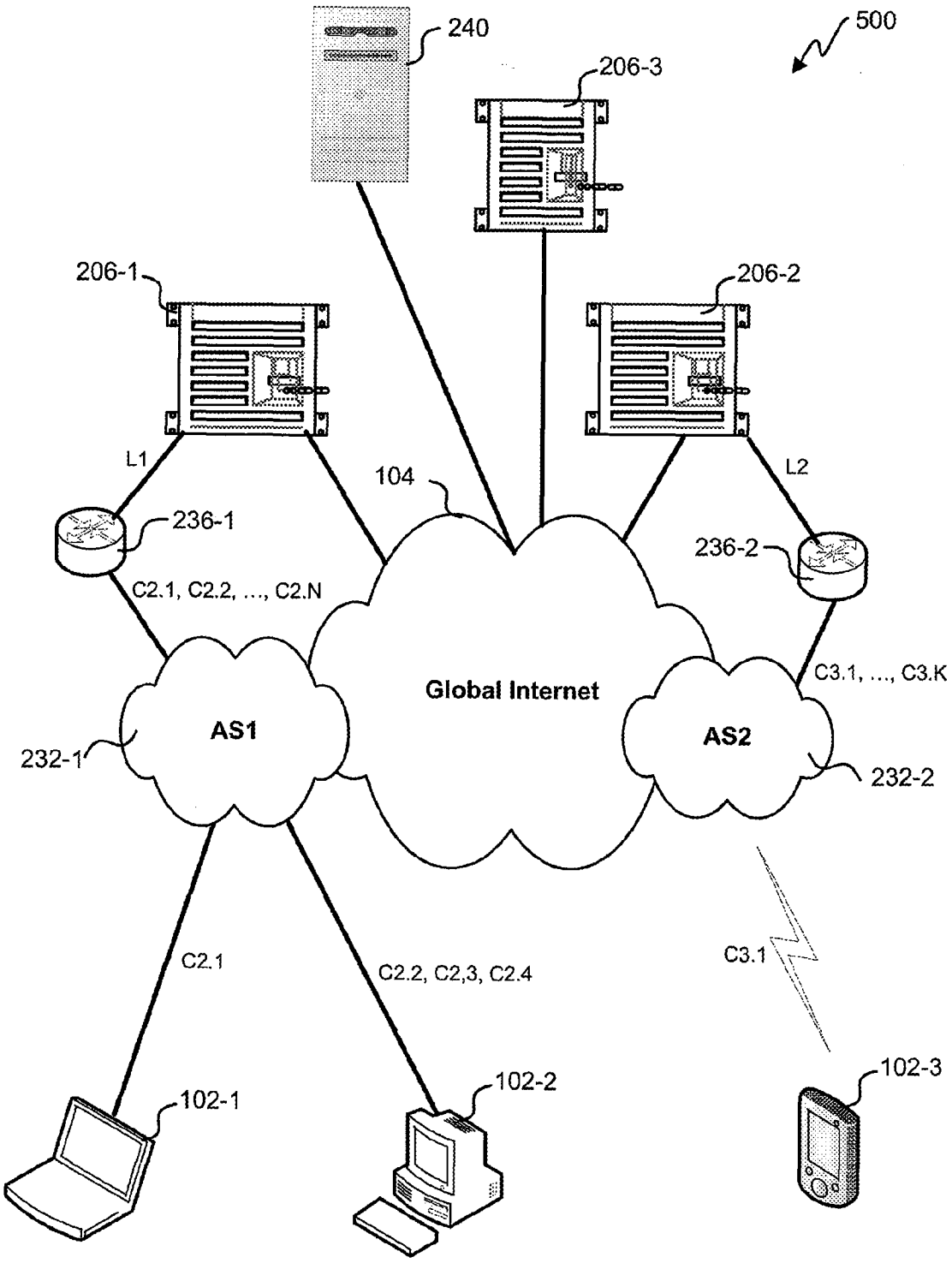

240
500
206-3
206-1
206-2
L1
104
L2
236-1
236-2
C2.1, C2.2, ..., C2.N
C3.1, ..., C3.K
Global Internet
AS1
AS2
232-1
232-2
C2.1
C2.2, C2,3, C2.4
C3.1
102-1
102-2
102-3

FIG. 5

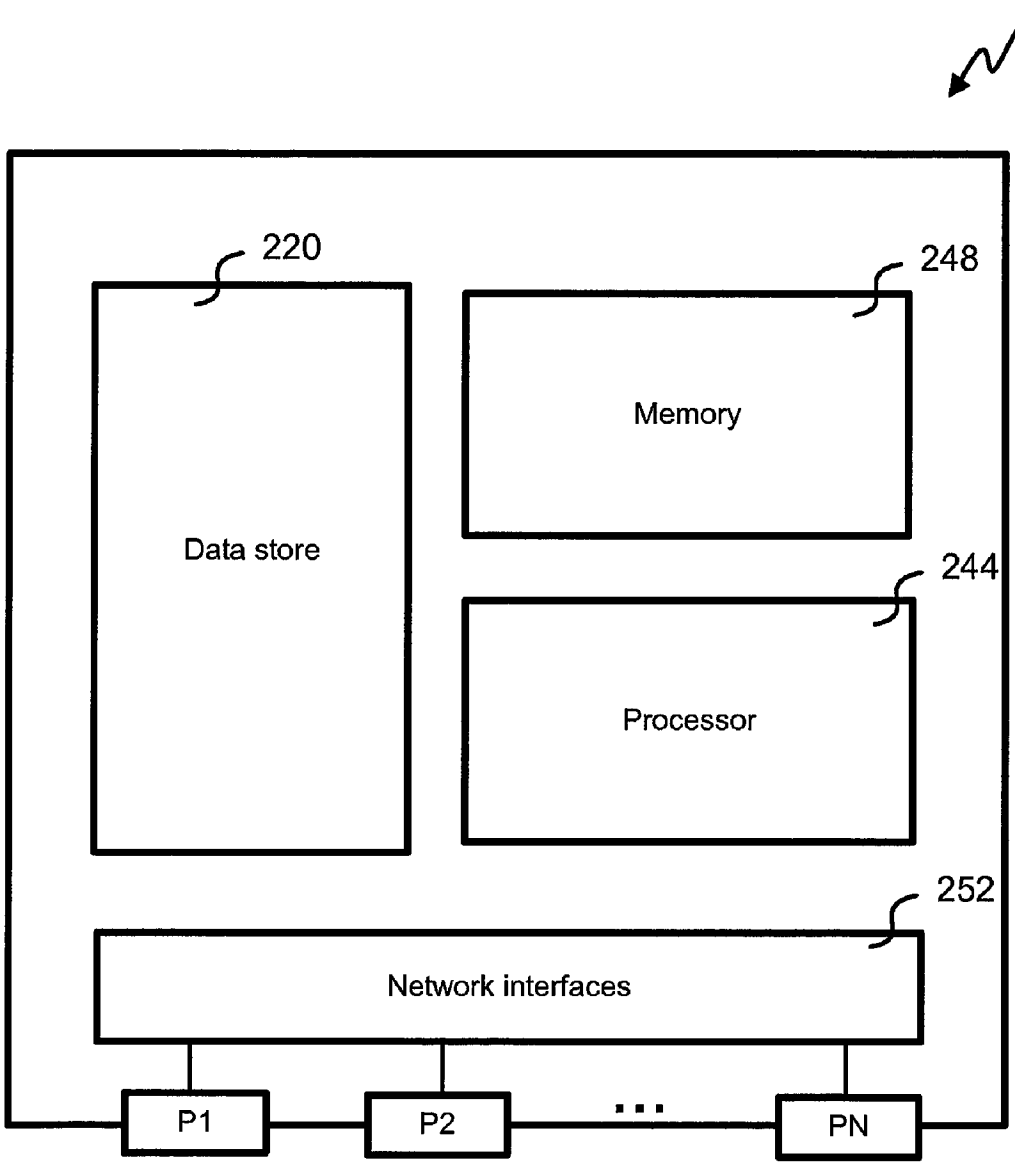

206
220
248
Memory
Data store
244
Processor
252
Network interfaces
P1
P2
...
PN

FIG. 6

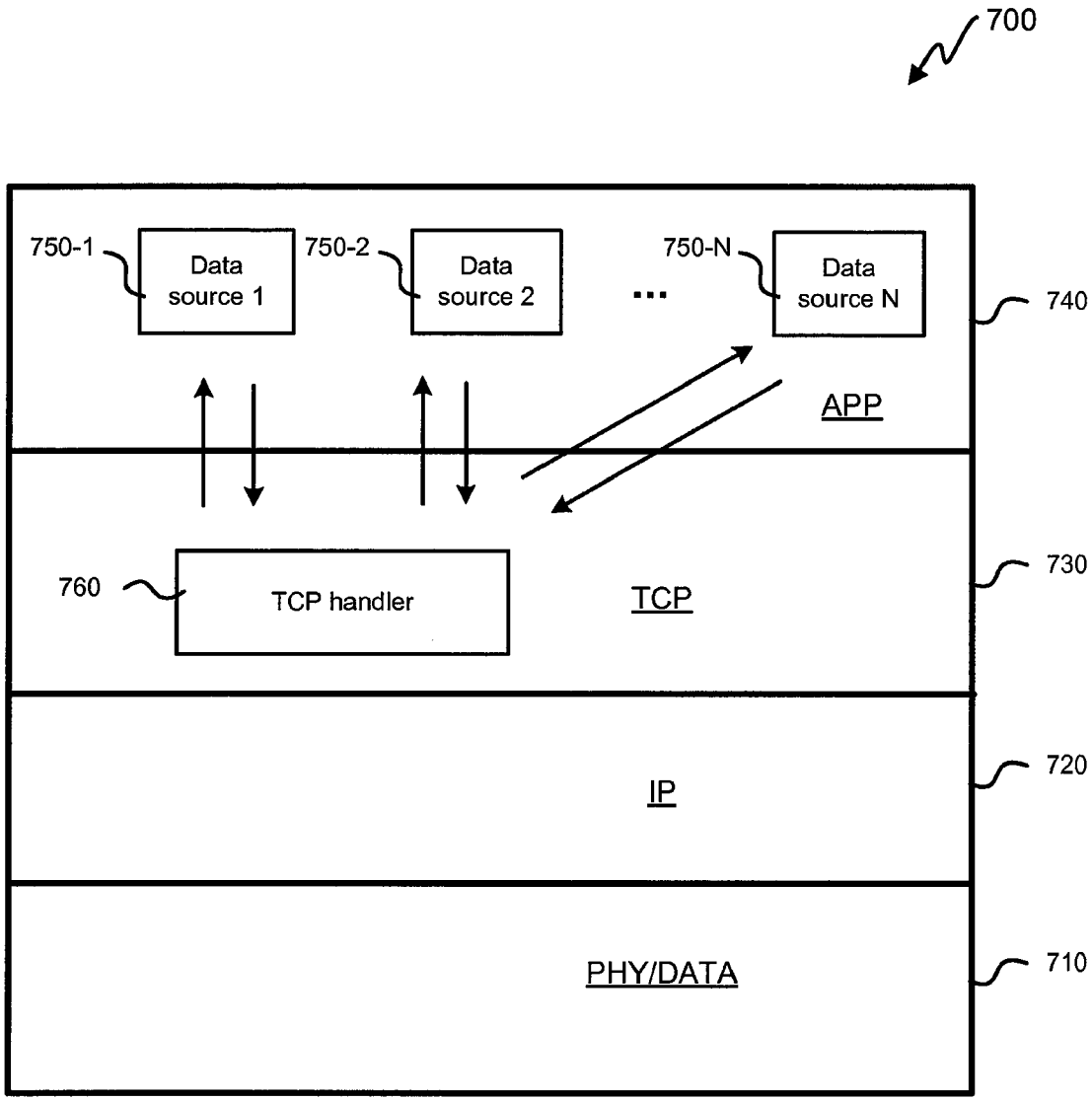

700
750-1
Data source 1
750-2
Data source 2
750-N
Data source N
740
APP
760
TCP handler
TCP
730
IP
720
PHY/DATA
710

FIG. 7

810

File name	Provider	File Size	Type	...	Attribs
logo.gif	ABC	50k	image	...	attr1=no
Index.html	ABC	100k	text	...	attr1=no
movie.mpg	ABC	1GB	video	...	attr1=yes

820

Provider	Service Level	...	Attribs
ABC	Premium	...	attr4=100000, attr6=fast
DEF	Standard	...	attr4=300000, attr6=slow

FIG. 8A

830

IP Address / Prefix	AS #	Country	...	Attribs
123.012.034.0 /24	12345	US	...	attr3=25
234.079.091.0 /24	34567	US	...	attr3=40
169.234.056.078	UNK	ASIA	...	attr3=35

840

AS #	Location	Service Type	Link utilization	...	Attribs
12345	Tempe, AZ	DSL	43%	...	attr6=fast, attr3=25
34567	Chicago, IL	Cable	92%	...	attr6=slow
34567	Atlanta, GA	SAT	–	...	attr3=40

FIG. 8B

850

ServerID	BW	CPS	...	Attribs
E1	725	900	...	attr4=100000, attr6=fast
E56	937	1877	...	attr4=300000, attr6=slow

FIG. 8C

900

Performance Profile	Timing	Pacing	Send Window	Comment
P1	25	1	300000	large file, near user
P2	40	1	175000	large file, latent user
P3	25	0	100000	small file

FIG. 9

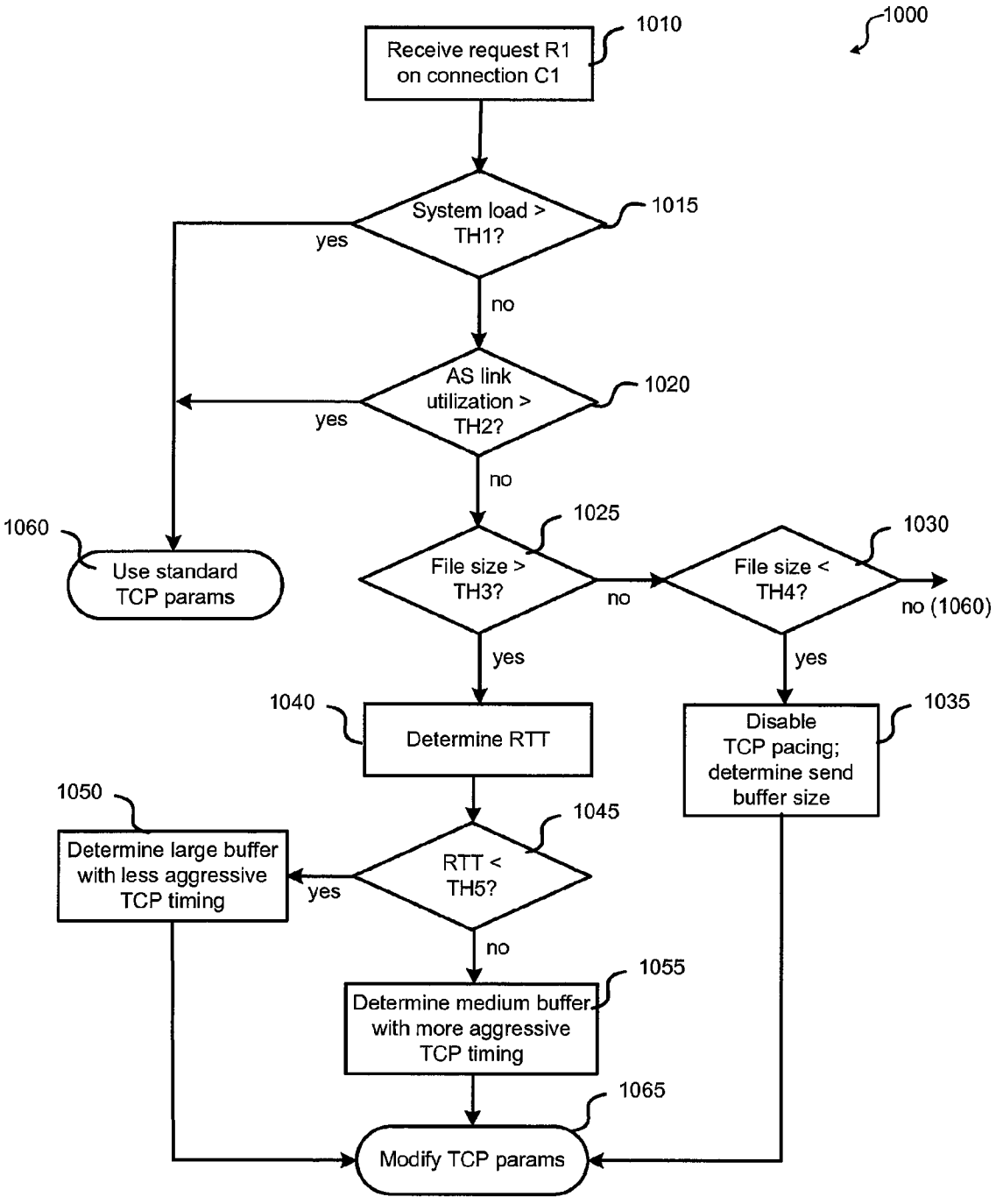

1000
1010
Receive request R1 on connection C1
1015
System load > TH1?
yes
no
1020
AS link utilization > TH2?
yes
no
1060
Use standard TCP params
1025
File size > TH3?
no
1030
File size < TH4?
no (1060)
yes
yes
1040
Determine RTT
1035
Disable TCP pacing; determine send buffer size
1050
Determine large buffer with less aggressive TCP timing
1045
RTT < TH5?
yes
no
1055
Determine medium buffer with more aggressive TCP timing
1065
Modify TCP params

FIG. 10

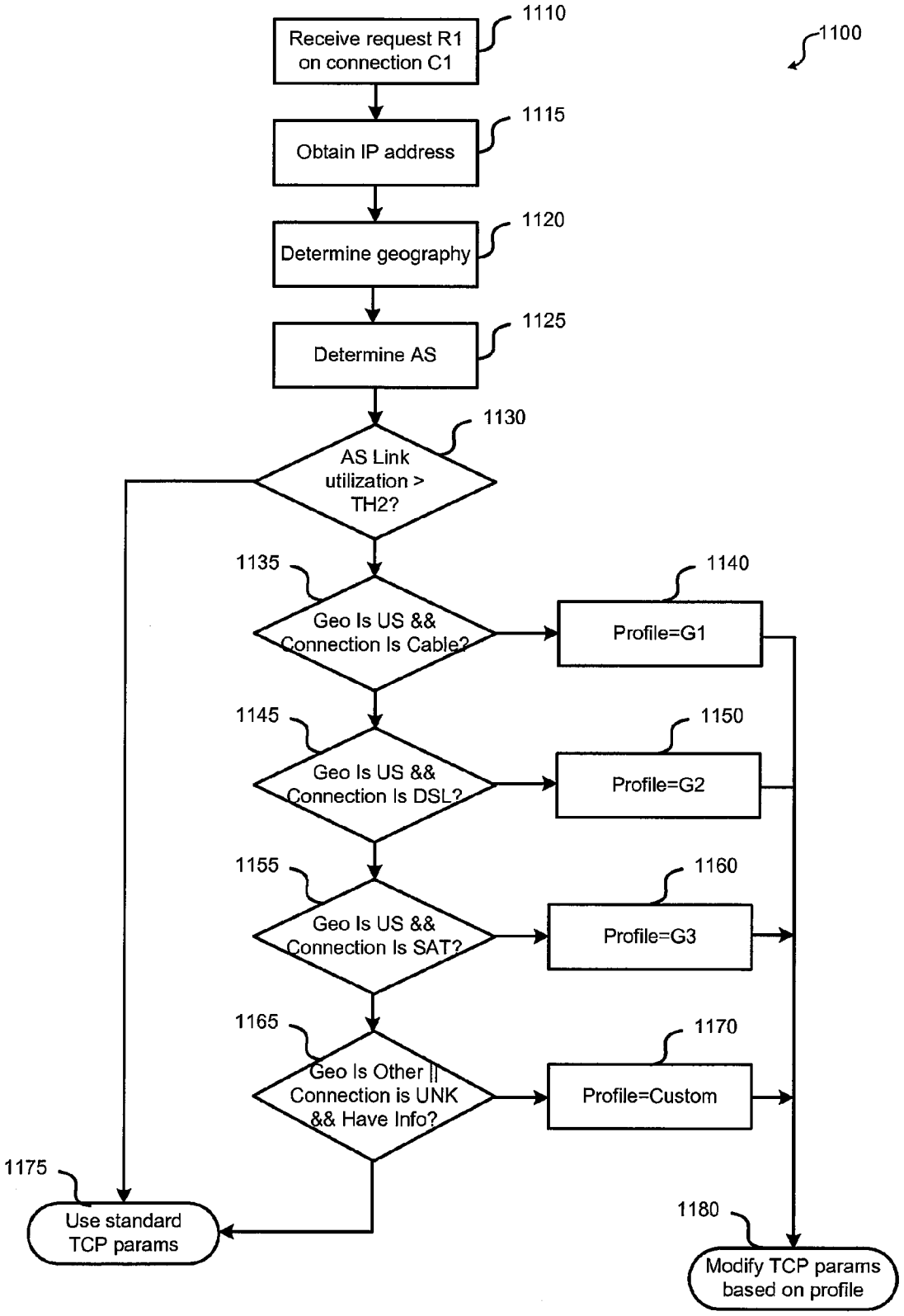

1100
1110
Receive request R1 on connection C1
1115
Obtain IP address
1120
Determine geography
1125
Determine AS
1130
AS Link utilization > TH2?
1135
Geo Is US && Connection Is Cable?
1140
Profile=G1
1145
Geo Is US && Connection Is DSL?
1150
Profile=G2
1155
Geo Is US && Connection Is SAT?
1160
Profile=G3
1165
Geo Is Other || Connection is UNK && Have Info?
1170
Profile=Custom
1175
Use standard TCP params
1180
Modify TCP params based on profile

FIG. 11

CONDITIONAL PROTOCOL CONTROL

CONDITIONAL PROTOCOL CONTROL

This application claims priority to Australian Patent Application Serial No. 2009201833, filed Mar. 26, 2009, which claims priority to International Patent Application Serial No. PCT/US2009/038361, filed Mar. 26, 2009; which are both incorporated herein by reference in their entirety.

This application also claims priority to U.S. patent application Ser. No. 12/572,981, filed Oct. 2, 2009, which is a continuation-in-part of International Patent Application Serial No. PCT/US2009/038361, filed Mar. 26, 2009; which are both incorporated herein by reference in their entirety.

BACKGROUND

This disclosure relates in general to interoperating nodes in an information processing system, such as an Internet content delivery system or an Internet transaction acceleration system, and, but not by way of limitation, to control of connection protocols.

In an information processing system, including communications networks such as the Internet, two or more nodes can work together, for example exchanging information or sharing resources, using one or more protocols that enable the participating nodes to interoperate. Nodes need not be physically distinct from one another, though they may be; normally, however, nodes are at least logically distinct from one another in at least some respect. Interoperating nodes may be operated or managed by a single common authority or by independent, unrelated authorities. Two or more interoperating nodes are often independently operated or managed; the Internet includes many well known examples of the interoperation of two or more independently managed nodes.

A protocol can be standardized such that a node using the standard protocol should be able to interoperate, at least at the level of the protocol, with any other node using the standard protocol. Standard protocols that become widely adopted can permit a node to interoperate with many other nodes. One such widely adopted standard protocol on the Internet is the Transmission Control Protocol (TCP), which today enables almost every device on the Internet to interoperate with almost every other device. TCP operates at the connection layer and enables nodes to interoperate with other nodes by establishing communications connections.

Standard protocols often employ the use of attributes, such as configurable parameters and selectable algorithms, to permit the protocol to operate effectively in various situations. For example, TCP controls message size, the rate at which messages are exchanged, and factors related to network congestion through the use of attributes, including both by the use of parameters, such as the receive window field used in sliding window flow control and the retransmission timer, and by the use of algorithms, such as slow-start, congestion avoidance, fast retransmit, and fast recovery algorithms. It is often the case, in many standard protocols, that at each node the initial protocol attribute settings to be used for all the communication connections at the node can be independently specified by the operator of the node.

A protocol can also be customized, which in general requires that each node have installed customized components to enable the custom protocol. Without the customized components, the node would not be able to fully interoperate with other nodes using the customized protocol. Although it therefore may limit the total number of interoperable nodes, or in the alternative require widespread action to install the protocol customized components, or possibly both, protocol customization is used in order to add function, improve performance, increase flexibility, or modify other characteristics of a standard protocol, or to make available an entirely new customized protocol. Many customized protocols have been proposed for use on the Internet.

SUMMARY

In one embodiment, a system utilizing a standard protocol to enable two or more nodes to interoperate is disclosed. The protocol attributes specified in the standard protocol are conditionally adapted to the circumstances, use, and/or operating conditions of the interoperation of the nodes. In another embodiment a method is disclosed for utilizing a standard protocol to enable two or more nodes to interoperate, wherein the protocol attributes specified in the standard protocol are conditionally adapted to the circumstances, use, and/or operating conditions of the interoperation of the nodes.

In one embodiment, a system for supplying content objects over a network is disclosed. The system includes a protocol handler, a protocol attribute information store, and a protocol attribute selector. The protocol handler manages a first connection and a second connection over the network using a standard protocol. The protocol attribute information store holds a plurality of attributes defined for a plurality of connections. The protocol attribute selector receives first information based on a first request for content, identifies first attributes corresponding to the connection that will service the first request for content, provides the protocol handler with the first attributes for the connection servicing the first request for content, receives second information based on a second request for content, identifies second attributes corresponding to the connection that will service the second request for content, and provides the protocol handler with the second attributes for the connection servicing the second request for content.

In another embodiment, a network connection method for delivering content is disclosed. A first request for content is received over a network at a server. The first request for content is evaluated to select first protocol attributes. A first connection that sends the content from the server to a first node is configured according to the first protocol attributes. A second request for content is received over the network at the server. The second request for content is evaluated to select second protocol attributes. A second connection that sends the content from the server to a second node is configured according to the second protocol attributes, where the first attributes affect the operation of the protocol differently than the second attributes affect the operation of the protocol.

In another embodiment, a system for conducting transactions over a network is disclosed. The system includes a protocol handler, a protocol attribute information store, and a protocol attribute selector. The protocol handler manages a first connection and a second connection over the network using a standard protocol. The protocol attribute information store holds a plurality of attributes defined for a plurality of connections. The protocol attribute selector receives first information based on a first transaction, identifies first attributes corresponding to the connection that will service the first transaction, provides the protocol handler with the first attributes for the connection servicing the first transaction, receives second information based on a second transaction, identifies second attributes corresponding to the connection that will service the second transaction, and provides the protocol handler with the second attributes for the connection servicing the second transaction.

In another embodiment, a network connection method for conducting transactions over a network is disclosed. A first transaction is initiated over a network at a server. The first transaction is evaluated to select first protocol attributes. A first connection, servicing the first transaction, between the server and a first node is configured according to the first protocol attributes. A second transaction is initiated over the network at the server. The second transaction is evaluated to select second protocol attributes. A second connection, servicing the second transaction, between the server and a second node is configured according to the second protocol attributes, where the first attributes affect the operation of the protocol differently than the second attributes affect the operation of the protocol.

Techniques for modifying the performance of a transport layer protocol in response to a request for content are disclosed. A connection can be established between a content distribution server and an end user computer according to preconfigured parameters. When a request for content is received over the connection, the content distribution server can determine one or more parameters relating to the performance of the connection using information from the request. The content distribution server can modify the connection at the transport layer according to the one or more parameters. Thereafter, the transport layer can manage delivery of the requested content to the end user computer in accordance with the modified parameters. In various embodiments, the content distribution server includes a modified TCP protocol stack which adjusts timing, pacing, and buffer allocation associated with a connection in response to requests from an application-layer data source.

In one embodiment, a method for managing delivery of content in a system comprising a server and an end user computer is disclosed. The method includes establishing a first connection at the server for communicating with the end user computer and receiving a request for content from the end user computer over the first connection. The method also includes determining one or more parameters relating to the performance of the first connection using information from the request and modifying the first connection at the transport layer based on the one or more parameters. Modifying the first connection can be done without notifying the end user computer. The method also includes sending the requested content from the server to the end user computer such that the transport layer manages delivery of the content in accordance with the modified parameters.

Optionally, the method includes retrieving metadata associated with a requested file and modifying the first connection based on the metadata. Alternatively or additionally, the method can include selecting a predetermined performance profile for the first connection using the information from the request and modifying the first connection based on the predetermined performance profile. The method can include determining a connection type of the end user computer and a latency characteristic associated with the connection type and modifying the first connection at the transport layer based on the latency characteristic. The method can also include determining a data size of the requested content, measuring a round trip time between the server and the end user computer when the data size exceeds a predetermined value, and modifying the first connection at the transport layer based on the size of the requested content and the round trip time.

In another embodiment, a content distribution server is disclosed. The server includes a network interface, a processor, a protocol handler, and a data source. The network interface includes a plurality of ports for sending and receiving data over a connecting network. The processor is coupled to the network interface and manages a plurality of connections to end user computers. The protocol handler establishes the connections with the end user computers according to preconfigured transport layer parameters of the content distribution server and manages the manner in which data is transmitted over the connections. The data source supplies the requested content. The data source monitors a first connection for a request, determines one or more modified transport layer parameters based on the request, and directs the protocol handler to modify the first connection independently of the other connections based on the one or more transport layer parameters.

In still another embodiment, a content distribution server is disclosed. The server includes means for sending and receiving data over a connecting network, means for managing a plurality of connections to end user computers, and means for establishing a connection with each end user computer according to preconfigured transport layer parameters. The server includes means for managing data transmission over the plurality of connections, means for modifying a connection based on one or more transport layer performance parameters, and means for supplying requested content to the end user computers over the plurality of connections. The server also includes means for monitoring a first connection for a content request, means for determining the one or more transport layer performance parameters for the first connection based on the request, and means for sending the requested content over the first connection modified by the one or more transport layer performance parameters.

In yet another embodiment, a computer program product comprising a computer-readable medium is disclosed. The computer-readable medium is encoded with one or more sequences of one or more instructions which, when executed by a processor, perform steps of establishing a first connection at the server for communicating with an end user computer and receiving a request for content from the end user computer over the first connection. The instructions operate to determine one or more parameters relating to the performance of the first connection based on information from the request and to modify the first connection at the transport layer using the one or more parameters without notifying the end user computer. Additionally, the instructions operate to send the requested content from the server to the end user computer such that the transport layer manages delivery of the content in accordance with the modified parameters.

Further areas of applicability of the present disclosure will become apparent from the detailed description provided hereinafter. It should be understood that the detailed description and specific examples, while indicating various embodiments, are intended for purposes of illustration only and are not intended to necessarily limit the scope of the disclosure.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** depicts a block diagram of an embodiment of a content delivery system.

FIGS. **2**A, **2**B, **2**C, and **2**D depict block diagrams of embodiments of a content download pair that sends content from a server to a client.

FIG. **3** illustrates a flowchart of an embodiment of a process for modification of the TCP protocol for various connections to a server.

FIG. **4** illustrates a flowchart of an embodiment of a process for modifying protocol attributes potentially on a connection-by-connection basis.

FIG. **5** shows aspects of a content delivery system.

FIG. **6** is a block diagram of an embodiment of a content distribution server.

FIG. **7** shows an exemplary content distribution server protocol stack.

FIGS. **8**A, **8**B, and **8**C show exemplary data elements such as can be used with a content distribution server.

FIG. **9** shows exemplary performance profiles such as can be used with a content distribution server.

FIG. **10** is a flowchart of a process for modifying transport layer protocol attributes.

FIG. **11** is a flowchart of process for modifying transport layer protocol attributes.

In the figures, similar components and/or features may have the same reference label. Further, various components of the same type may be distinguished by following the reference label by a dash and a second label that distinguishes among the similar components. If only the first reference label is used in the specification, the description is applicable to any one of the similar components having the same first reference label irrespective of the second reference label.

DETAILED DESCRIPTION OF EMBODIMENTS

The ensuing description provides preferred exemplary embodiment(s) only, and is not intended to limit the scope, applicability or configuration of the disclosure. Rather, the ensuing description of the preferred exemplary embodiment(s) will provide those skilled in the art with an enabling description for implementing a preferred exemplary embodiment. It being understood that various changes may be made in the function and arrangement of elements without departing from the spirit and scope as set forth in the appended claims.

Referring first to FIG. **1**, a block diagram of an embodiment of an Internet content delivery system **100** is shown. Generally, one or more nodes request content from one or more other nodes. In FIG. **1**, a number of end users **108** respectively use their end user system or client **102** to download and view content objects from the global Internet **104**. The content delivery system **110** has one or more servers that provide content object downloads. The content delivery system **110** can include any number of cache servers, application servers, content servers, service servers, and/or database servers to provide content to the clients **102**. Although this embodiment shows particular communication pairs, other embodiments could communicate between any pair of nodes on a network, including between pairs of clients or between pairs of servers, and yet other embodiments could communicate among more than two nodes, such as in a broadcast or multicast implementation.

With reference to FIGS. **2**A, **2**B, **2**C and **2**D, embodiments of a content download pair **200** that sends content from a server **206** to a client **102** are shown. A primary embodiment described here is the interoperation of two nodes **102**, **206** on the Internet communicating using TCP, one node being a client **102** that requests information, such as web page content, multimedia, or software downloads, and the second node being a server **206** that provides information in response to a request. TCP operates in the transport layer of the seven-layer Open Systems Interconnection (OSI) model. In other embodiments, nodes **102**, **206** interoperate in ways other than communication in a network, such as sharing data within a computer or group of computers across an available system or intersystem interface; interoperate using communications networks other than the Internet **104**, such as a private communications network; interoperate using the Internet **104** or a private network using protocols other than TCP, such as UDP, RTP, multicast protocols, and other standard protocols in the transport layer; interoperate using the Internet **104** or a private network using standard protocols operating in a layer that underlies the transport layer; interoperate, using standard protocols and the Internet or a private network, more than two at a time, such as in clusters or multicast groups; or interoperate, using standard protocols and the Internet or a private network, other than as a client and server, including interoperating as peers, as collaborative nodes, or as a group of nodes under the common control of one or more other nodes or under the common control of a controller.

In the primary embodiment, the server **206** conditionally adapts the attributes of the TCP protocol for each TCP connection established by a client **102**. Conditionally adapting the attributes of the TCP protocol does not require changes to standard TCP protocol implementations at every node, does not require special components be installed in the TCP protocol implementation at every node, and therefore does not comprise implementing a customized protocol as previously described; rather, the primary embodiment utilizes the standard TCP protocol and the attributes implemented in it. In other embodiments, the server conditionally adapts the attributes of other transport-layer protocols for each session established by a client **102**; the server conditionally adapts the attributes of other protocols that underlie the transport layer for each session established by a client **102**; the server **206** conditionally adapts the attributes of the TCP protocol or other protocol for groups of connections or sessions established by clients **102**; the server **206** conditionally adapts the attributes of the TCP protocol or other protocol for connections or sessions established by groups or subsets of groups of clients **102**; the client **102** conditionally adapts the attributes of the TCP protocol for each TCP connection established; the client **102** conditionally adapts the attributes of other protocols for each session established; the client **102** conditionally adapts the attributes of the TCP protocol or other protocol for groups of connections or sessions; a node conditionally adapts the attributes of the TCP protocol or other protocol for each connection or session; a node conditionally adapts the attributes of the TCP protocol or other protocol for groups of connections or sessions; a node conditionally adapts the attributes of the TCP protocol or other protocol for connections or sessions established by groups or subsets of groups of nodes; a controller conditionally adapts the attributes of the TCP protocol or other protocol for each connection or session of at least one node of an interoperating group of nodes; a controller conditionally adapts the attributes of the TCP protocol or other protocol for groups of connections or sessions of at least some nodes of an interoperating group of nodes; or, a controller conditionally adapts the attributes of the TCP protocol or other protocol for connections or sessions established by groups or subsets of groups of nodes.

Software, software modifications, or equivalent function, may optionally be implemented at a server, client, or node that sets the conditionally adapted protocol attributes of a connection or session, but need not be implemented at servers, clients, or nodes that passively participate in a conditionally adapted protocol connection or session. Such software, software modifications, or equivalent function will only be needed if existing protocol software or other software on the server, client, or node does not provide a facility for programmatically or similarly changing attributes of the protocol that is used; in this event, software, a software modification, or equivalent facilities to provide such a programmatic or similar interface may be implemented.

Conditionally adapting the protocol for each connection or session, or collection of connections or sessions, results in at

least one node that, concurrently or over time, uses a protocol for multiple unrelated connections or sessions wherein the protocol attributes vary, at least initially and sometimes persistently, from one connection or session to another, most often varying differently from any ordinary protocol attribute variations that naturally occur from one connection or session to another through use of the standard protocol implementation among heterogeneous nodes.

In the primary embodiment, TCP connections are established in order to use HyperText Transfer Protocol (HTTP) to communicate information requests from clients **102** to servers **206** and responses from servers **206** to clients **102**. HTTP is a scheme that operates above, and depends on the presence of a functioning and reliable protocol at, the transport layer of the seven-layer model developed in the Open Systems Interconnection (OSI) initiative. Other embodiments use application-layer protocols other than HTTP in conjunction with TCP; use TCP alone, i.e., without HTTP; use other protocols; or, use other application-layer protocols in conjunction with other protocols. HTTP utilizes Uniform Resource Locators (URLs), Uniform Resource Names (URNs), and Uniform Resource Identifiers (URIs) to identify information. URLs are used in the primary embodiment. Other embodiments use URIs, URNs, other identifiers, or other information. A URL begins with the scheme identifier, which identifies the namespace, purpose, and syntax of the remainder of the URL. In the primary embodiment utilizing HTTP, the typical scheme is "http". The scheme is followed by a host field, which contains the IP address or name of the host where the requested information can be found, optionally followed by a port number, optionally followed by a path, which is an HTTP selector, optionally followed by a search portion, which is a query string. The full URL, then, is an alphanumeric string containing the scheme, host field, any optional following strings, and special characters such as ":", "/", and "?" that are reserved for special functions such as designating a hierarchical structure in the URL. Other embodiments could use different application-layer protocols such as Telnet, File Transfer Protocol (FTP), secure HTTP (HTTPS), and Simple Mail Transfer Protocol (SMTP).

In the primary embodiment, the server **206** bases the conditional adaptation of the attributes of the TCP protocol on the alphanumeric URL string provided by the client **102** in its information request. In another embodiment, a server, client or other node bases the conditional adaptation of the attributes of the TCP protocol or other protocol on the application-layer protocol specified or on identifying information, equivalent to a URL, or other information provided in, or characteristic of, an information request, connection, or session. In other embodiments, a server, client or other node bases the conditional adaptation of the attributes of the TCP protocol or other protocol on the IP address of one or more servers, clients, or nodes; on network information associated with the IP address of one or more servers, clients, or nodes, including the Autonomous System (AS) number, identity of network operator, geographic location, logical or physical network location, logical or physical network segment, or network interconnection characteristics associated with the IP address(es) of one or more servers, clients, or nodes; the geographic location of the server, client or node; and/or, the logical or physical network location of the server, client or node; the logical or physical address of the server, client or node; the logical or physical name of the server, client or node; and/or, the network or other path from or to a server, client or node. In other embodiments a server, client or node bases the conditional adaptation of the attributes of the TCP protocol or other protocol on recent network performance measurements, including latency, jitter, packet loss, round trip time, and/or the measured variance in a network performance measurement across multiple samples; on recent measures of utilization of a network, network segment, network interface, or network port; and/or, on recent measurements of performance or utilization of a server, group of servers, or server component(s) such as memory, processor, disk, bus, intersystem interface, and/or network interface. In still other embodiments, a server, client or node bases the conditional adaptation of the attributes of the TCP protocol or other protocol on temporal factors, including time of day; day of week, month, or year; specific date; occurrence of a holiday or religious observance; occurrence of a temporal event such as a news event or sports event; seasonal occurrence; and/or a scheduled event or time period.

In the primary embodiment, the protocol attribute selector **212** of the server **206** compares the alphanumeric URL string provided by the client **102** in its information request to a table **220** containing partial or whole URLs and identifies the most specific match from left to right that it can find in the table **220**. In another embodiment, the server **206** compares a subset of the alphanumeric string, for example some or all of the characters in the query string, or the characters following the host field up to the first subsequent slash (i.e., "/"), to a table **220**. In another embodiment, the client or node **102** makes a conditional adaptation of protocol attributes, using the alphanumeric URL string or a subset of it. In other embodiments, the alphanumeric URL string or a subset of it is processed to obtain a value or indicator that is used to determine a conditional adaptation of protocol attributes. In other embodiments, the information used to determine the conditional adaptation of protocol attributes is identifying information equivalent to a URL, one or more IP addresses, network information associated with one or more IP addresses, network interconnection characteristics associated with one or more IP addresses, or the geographic location, or logical or physical network location, of a server, client or node. In other embodiments, the information used to determine the conditional adaptation of protocol attributes comprises one or more recent performance measurements or thresholds related to one or more servers, clients, or nodes, or groups of servers, clients, or nodes, or related to one or more networks, network segments, network components, or network interfaces, or groups of network segments, network components, or network interfaces; rates or amounts of variation in one or more performance measurements or thresholds related to one or more servers, clients, or nodes, or groups of servers, clients, or nodes, or related to one or more networks, network segments, network components, or network interfaces, or groups of network segments, network components or network interfaces; rates or amounts of resource utilization, including utilization related to one or more servers, clients, or nodes, or groups of servers, clients, or nodes, or components of one or more servers, clients, or nodes, groups of components of servers, clients, or nodes, or related to one or more networks, network segments, network components, or network interfaces, or groups of network segments, network components, or network interfaces; rates or amounts of variation in resource utilization, including variation in utilization related to one or more servers, clients, or nodes, or groups of servers, clients, or nodes, or components of one or more servers, clients, or nodes, groups of components of servers, clients, or nodes, or related to one or more networks, network segments, network components, or network interfaces, or groups of network segments, network components, or network interfaces; and/or, thresholds of resource utilization, including utilization related to one or more servers, clients, or nodes, or

groups of servers, clients, or nodes, or components of one or more servers, clients, or nodes, groups of components of servers, clients, or nodes, or related to one or more networks, network segments, network components, or network interfaces, or groups of network segments, network components, or network interfaces.

In the primary embodiment, a table **220** containing partial or whole URLs for comparison by the protocol attribute selector **212** is stored on the server **206**. In other embodiments, a table **220** containing partial or whole URLs for comparison, or subsets of the alphanumeric URL string used for comparison, is stored remotely from the server **206** such as the embodiment shown in FIG. **2**B and the comparison of the alphanumeric URL string or subset of the alphanumeric URL string is made at the server **206**, or is made at the remote table storage location or at another location and the result of the comparison, or an indicator of the result of the comparison, or the protocol attributes to be used, are returned to the server **206**. In other embodiments, a table **220** or database of information used for comparison is stored at the server **206**, or is stored remotely from the server **206** and one or more comparisons are made at the server **206**, or are made at the remote storage location or at another location, and the results of the comparisons, or one or more indicators of the results of the comparisons, or the protocol attributes to be used, are returned to the server **206**. In yet other embodiments, a table **220** containing partial or whole URLs for comparison, or subsets of the alphanumeric URL string used for comparison, or a database of information used for comparison is stored at a client or node **102** as is shown in FIG. **2**D, or is stored remotely from a client or node and one or more comparisons are made at the client or node **102**, or are made at the remote storage location or at another location and the results of the comparisons, or one or more indicators of the results of the comparisons, or the protocol attributes to be used, are returned to the client or node **102**. In other embodiments, a protocol attribute to be used, or an indicator of a protocol attribute to be used, is extracted from, or derived from, a URL or equivalent identifying information used by a server, client, or node; is extracted from, or derived from, address information or a whole or partial name of a server, client, or node; is extracted from, or derived from a user name or identifier, class or type of user, group of users, or selection of users, optionally as associated with a service, server, client, or node; is extracted from, derived from, or associated with, an application or class or group of applications, a service or class or group of services, or a database or equivalent source of data or a class or group of databases or sources of data; is extracted from, or derived from, the identifier of a standard protocol, a standard protocol message (for example, a TCP SYN), the protocol-level content of a message, or protocol message headers or equivalent information; is derived from all or a part of the time, all or a part of the date, or all or a part of the duration since a time or date; is determined according to a sequence or progression; and/or, is determined randomly or pseudo-randomly.

After one or more comparisons are made, or after the extraction or derivation of a value, or after an association or determination is made, the server, client, or node will have one or more values it can use as protocol attributes or as indicators of which protocol attributes to use; or, if there has not been a determination of a condition resulting in one or more protocol attribute values to use, then in the primary embodiment the server **206** will use the standard values for the protocol attributes as configured for the server **206**, and in other embodiments, the server, client, or node will use the standard values as configured for it for some or all of the protocol attributes; the server, client, or node will use another default value for some or all of the attributes; the server, client, or node will use the last-used values for some or all of the attributes; or, the server, client, or node will use randomly, pseudo-randomly, or arbitrarily determined values for some or all of the attributes.

In the primary embodiment, the server **206** is a cache server, typically operating in a group of cache servers, and groups of cache servers are distributed at numerous points on the global Internet. The server **206** includes a cache function **208** coupled to a content cache **210** to provide caching of content for the server **206**. In another embodiment, the server **206** is a cache server operating individually, or independently from other cache servers. In other embodiments, the server **206** is a content server that provides content; a content server operating in a group of content servers; a content server operating in a group of content servers distributed at numerous points on the global Internet; an application server that supports one or more applications; an application server operating in a group of application servers; an application server operating in a group of application servers distributed at numerous points on the global Internet; a service server that provides one or more services; a service server operating in a group of service servers; a service server operating in a group of service servers distributed at numerous points on the global Internet; a database server that provides data; a database server operating in a group of database servers; a database server operating in a group of database servers distributed at numerous points on the global Internet; a server operating in a heterogeneous group of servers; or, a server operating in a heterogeneous group of servers distributed at numerous points on the global Internet.

In the primary embodiment, each server **206** has an identical copy of a table **220** containing partial or whole URLs to which the alphanumeric URL string provided by the client **102** in its information request will be matched if possible, along with an indicator of the protocol attribute values to be used when that match occurs or the actual protocol attribute values themselves to be used when that match occurs. The table **220** can be modified from time to time, including adding new entries, changing the contents of existing entries, and deleting entries. Any time that a new version of the table **220** is created, copies of that version are distributed to servers **206** in the global group of cache servers. Optionally, the table **220** can have an expiration date and time, after which the server **206** will stop using the table **220** if it is out-of-date and has not been replaced with an updated table. In the primary embodiment, the conditional protocol control information is combined with other information distributed to servers **206** so as to minimize to the extent practicable the number of tables **220** that are distributed to, and synchronized among, the global group of servers; in another embodiment, this combination and minimization is not implemented; and this combination and minimization is not done in most or all other embodiments.

In other embodiments, the conditional protocol information may be in the form of a table **220** as in the primary embodiment, or may be in the form of multiple tables, or may be in the form of one or more data files, one or more databases, one or more calculated or derived elements, one or more calculated or derived groups of elements, one or more individual equations or formulas, one or more groups of equations or formulas, one or more individual expressions, one or more groups of expressions, one or more individual programs or routines, one or more groups of programs or routines, one or more individual listings, one or more groups of listings, and/or other forms equivalent to any of these.

In other embodiments, the conditional protocol control information contains partial or whole URLs to which the alphanumeric URL string provided by the client **102** in its information request will be matched if possible, along with an indicator of the protocol attribute values to be used when that match occurs or the actual protocol attribute values themselves to be used when that match occurs, and is the same for each server **206** at a location, but varies from one location to another; is the same for each server in a region, but varies from one region to another; or, is different at each server. In other embodiments the conditional protocol control information contains partial or whole URLs to which the alphanumeric URL string used by a client or node **102** will be matched if possible, along with an indicator of the protocol attribute values to be used when that match occurs or the actual protocol attribute values themselves to be used when that match occurs, and the conditional protocol control information is identical at each client or node **102**; is the same for each client or node **102** in a group, but varies from one group of clients or nodes **102** to another; is the same for each client or node **102** in a region, but varies from one region to another; or, varies from one client or node **102** to another client or node **102**.

In other embodiments, the conditional protocol control information contains other comparison information that will be matched, if possible, along with a corresponding indicator of the protocol attribute values to be used when that match occurs or the corresponding actual protocol attribute values themselves to be used when that match occurs, and each server, client, or node has an identical copy of the conditional protocol control information; the conditional protocol control information is the same for each server, client, or node at a location, but varies from one location to another; the conditional protocol control information is the same for each server, client, or node in a region, but varies from one region to another; the conditional protocol control information is the same for each server, client, or node in a group, but varies from one group to another; or, the conditional protocol control information varies from one server, client, or node to another. In these embodiments, the comparison information can include subsets of the alphanumeric URL string; identifying information equivalent to a URL; one or more IP addresses; network information associated with one or more IP addresses; network interconnection characteristics associated with one or more IP addresses; the geographic location, or logical or physical network location, of a server, client, or node; a value of a performance measurement or threshold; a rate or amount of variation in a performance measurement or threshold; a rate or amount of resource utilization; a rate or amount of variation in resource utilization; a threshold of resource utilization; addresses, names, classes, types, or group identifiers of servers, clients, nodes, users, applications, services, databases, or other data sources; protocol identifiers, protocol message identifiers or attributes, or protocol message headers or attributes; times, dates, or intervals; and/or, sequences, ratios, progressions, equations, randomly generated data, or arbitrarily determined data.

In embodiments where the conditional protocol control information is not identical at all servers, clients, or nodes, optionally a system can create conditional protocol control information for each location, region, server, client, or node, and optionally can manage the distribution of the conditional protocol control information to each location, region, server, client, or node. For partial or whole URLs, subsets of a URL alphanumeric string, or other comparison information that will be compared, a conditional protocol information management and distribution system (CPIMDS) optionally generates, or stores, or generates and stores, protocol attributes for any or all of a server, all servers, a subset of servers, a location, all locations, a subset of locations, a region, all regions, a subset of regions, a group, all groups, a subset of groups, an application, all applications, a subset of applications, a service, all services, a subset of services, a database or data source, all databases or data sources, a subset of databases or data sources, a protocol, all protocols, a subset of protocols, a client, all clients, a subset of clients, a node, all nodes, a subset of nodes, or a combination of some, a subset of, a subset of some of, or all of, servers, locations, regions, groups, applications, services, databases or data sources, protocols, clients, or nodes; and optionally generates, or stores, or generates and stores, default protocol attributes to be used in the absence of a matching comparison for any or all of a server, all servers, a subset of servers, a location, all locations, a subset of locations, a region, all regions, a subset of regions, a group, all groups, a subset of groups, an application, all applications, a subset of applications, a service, all services, a subset of services, a database or data source, all databases or data sources, a subset of databases or data sources, a protocol, all protocols, a subset of protocols, a client, all clients, a subset of clients, a node, all nodes, a subset of nodes, or a combination of some, a subset of, a subset of some of, or all of, servers, locations, regions, groups, applications, services, databases or data sources, protocols, clients, or nodes. The CPIMDS also optionally generates, or stores, or generates and stores, one or more rules for determining which of the server, location, region, group, application, service, database or data source, protocol, client, node, or combined values to use when determining a comparison to be included in the conditional protocol control information for a server, all servers, a subset of servers, a location, all locations, a subset of locations, a region, all regions, a subset of regions, a group, all groups, a subset of groups, an application, all applications, a subset of applications, a service, all services, a subset of services, a database or data source, all databases or data sources, a subset of databases or data sources, a protocol, all protocols, a subset of protocols, a client, all clients, a subset of clients, a node, all nodes, a subset of nodes, or a combination of some, a subset of, a subset of some of, or all of, servers, locations, regions, groups, applications, services, databases or data sources, protocols, clients, or nodes; optionally generates, or stores, or generates and stores, one or more rules for determining when to create, or distribute, or create and distribute, conditional protocol control information for or to a server, all servers, a subset of servers, a location, all locations, a subset of locations, a region, all regions, a subset of regions, a group, all groups, a subset of groups, a client, all clients, a subset of clients, a node, all nodes, a subset of nodes, or a combination of some or all of servers, locations, regions, groups, clients, nodes, or a subset of servers, locations, regions, groups, clients, nodes, or for or to all servers, clients, or nodes; optionally generates, or stores, or generates and stores, one or more rules for determining whether, and if so when, any conditional protocol control information shall expire; optionally includes an expiration value with some or all of the conditional protocol control information; and, optionally distributes conditional protocol control information to a server, all servers, a subset of servers, a location, all locations, a subset of locations, a region, all regions, a subset of regions, a group, all groups, a subset of groups, a client, all clients, a subset of clients, a node, all nodes, a subset of nodes, or a combination of some or all of servers, locations, regions, groups, clients, nodes, or a subset of servers, locations, regions, groups, clients, nodes, or for or to all servers, clients, or nodes.

In embodiments where the conditional protocol control information is maintained separately from each server, client, or node, the conditional protocol control information can optionally include, or can optionally correlate to, an indication of for which servers, locations, regions, groups, applications, services, databases or data sources, protocols, clients, nodes, or supersets, subsets, or combinations thereof, the conditional protocol control information can be used in adapting protocol attributes. The conditional protocol control information can be maintained at a single location or multiple locations; if maintained at multiple locations, each location can maintain a complete copy of all conditional protocol control information, or each location can maintain a copy of a subset of the conditional protocol control information. Optionally, a conditional protocol information management and distribution system can create conditional protocol control information for, and optionally distribute conditional protocol control information to, each conditional protocol control information location, groups of conditional protocol control information locations, or all conditional protocol control information locations. If conditional protocol control information locations store a subset of the conditional protocol control information, participating servers, clients, or nodes can optionally use a table, ordered table, shuffled table, directory, logical name translation system, or information obtained from, or received from, another server, client, node, or controller, to determine which conditional protocol control information location to receive conditional protocol control information from.

Referring next to FIG. **3**, an embodiment of a process **300** for modification of the TCP protocol for various connections to a server **206**. In summary, in the primary embodiment two nodes on the Internet communicate using TCP, one node being a client **102** that requests information in block **304**, such as web page content, and the second node being a cache server **206** that provides information in response to a request. The cache server **206** operates as one of a group of cache servers, and groups of cache servers are distributed at numerous points on the global Internet. TCP connections are established in order to use HTTP to communicate information requests from clients **102** to servers **206** in block **304** and responses from servers **206** to clients **102**. HTTP utilizes URLs; each URL begins with the scheme "http", which is followed by a host field containing the IP address or name of the host where the requested information can be found, followed by a path, which will be used at the server **206** to locate the requested object or information, optionally followed by a query string. Thus, the full URL typically is an alphanumeric string containing the scheme, host field, path, and any optional query string, each part of which is separated from the other parts by special characters such as ":", "/", and "?". Thus, an example URL referencing content that can be served by the cache server **206** in the primary embodiment might look like:

http://customer1.webserving.com/folderB/directory/logo.gif

optionally followed by "?" and a query string. A client **102** requesting this object would send an HTTP message using an HTTP method called "GET" to the server **206** identified by the hostname "customer1.webserving.com" in block **304**. The server identification is accomplished through a DNS translation of the hostname into one or more IP addresses. The Domain Name System (DNS) is the name translation system used in the Internet. The HTTP GET message sent to the server **206** typically includes part or all of the URL that the client **102** is requesting. In block **308**, some or all of the URL is compared with entries in the table **220** to determine the attributes to use for the connection as described further below.

In the primary embodiment, in block **312** the server **206** conditionally adapts the attributes of the TCP protocol for each TCP connection established by a client **102**. Conditionally adapting the attributes of the TCP protocol does not require changes to standard TCP protocol implementations at every node, i.e., any client **102** that supports standard TCP (which virtually all Internet and all World Wide Web clients do) can communicate with the server **206** to get the object via a conditionally adapted TCP connection, but may require a modification to the server's TCP protocol handler to allow changes to be made to the TCP protocol attributes by another program, or on a per-connection basis, or both. Where the TCP protocol handler provides a facility for programmatically making these changes, the content serving application may require an interface enabling it to communicate conditionally adapted TCP protocol attributes to the TCP protocol handler. In the primary embodiment, a set sockets statement can be used to communicate conditionally adapted TCP protocol attributes from the content serving application to the TCP handler **214**, which can be a modified TCP software stack that accepts and implements changes to the TCP protocol attributes on a per-connection or per-request basis. In various embodiments, the TCP handler could be implemented in software or hardware or both, and can be implemented as part of a node or separately from the node, for example in a switch or other device that provides protocol services.

Conditionally adapting the protocol for each connection results in the server **206** concurrently using the TCP protocol for multiple unrelated connections, wherein the TCP protocol attributes vary, at least initially and sometimes persistently, from one connection to another. At block **316**, the requested object or information is sent from the server **206** to the client **102**. When a requested object or information is delivered without any change to the TCP protocol attributes, the standard TCP protocol attributes pre-configured for the TCP protocol handler are used; but when the requested object or information is delivered with conditionally adapted TCP protocol attributes, the conditionally determined TCP protocol attributes are used instead of the standard pre-configured TCP attributes.

A typical caching content server **206** in actual operating use in a web hosting service provider environment may typically service 2,000 to 3,000 requests per second, across 100 to 200 (or more) customers. If half of these requests will use changed TCP protocol attributes and half will not, then on average, half of the connections managed by the server **206** at any one point in time would use the changed TCP protocol attributes and half would use the standard TCP protocol attributes as pre-configured in the TCP handler **214**. Additionally, if there are five separate (different) changed TCP protocol attribute sets across which those requests using changed TCP protocol attributes are evenly distributed, then on average at any one point in time a server **206** would concurrently be operating TCP connections with six different sets of TCP protocol attributes: 50% of the connections would utilize standard pre-configured TCP protocol attributes, and 10% would utilize each of the five possible changed TCP protocol attribute sets.

In the primary embodiment, the server **206** bases the conditional adaptation of the attributes of the TCP protocol on the alphanumeric URL string provided by the client **102** in the HTTP GET message. In the primary embodiment, the conditional protocol control information is in the form of a table **220** containing partial and/or whole URLs for comparison,

the table **220** is stored on the server **206**, and the table **220** is generally kept reconciled on all servers **206**. The server **206** compares the alphanumeric URL string provided by the client **102** in the GET request to the table **220** and identifies the most specific match from left to right that it can find in the table **220** in block **308**. Taking our earlier example and assuming a simplified table, the sole Table shows mappings from whole and/or partial URLs into TCP attribute sets comprising specific protocols attributes (identified as "attr**1**", "attr**2**", "attr**3**", etc.) to be used and the appropriate value or setting for that use of that attribute.

Table Mapping to TCP attribute sets

Partial URL	TCP Attribute Set
http://customer1.webserving.com/folderA/	attr1 = yes, attr3 = 25, attr4 = low
http://customer1.webserving.com/folderB/	attn = no, attr2 = 1, attr4 = high
fastnet.com	attr3 = 50, attr6 = fast
http://customer2.webserving.com/	
http://customer3.webserving.com/	attr1 = yes, attr3 = 25, attr4 = low

The URL in the example above, "http://customer1.webserving.com/folderB/directory/logo.gif," would be matched against the second line-entry in the table. The TCP protocol attribute set (group of TCP protocol attributes) to be used for the TCP connection that services, or responds to, this HTTP GET message from this client **102** would be "attr**1**=no, attr**2**=**1**, attr**4**=high" and the TCP protocol attributes for this TCP connection would be set accordingly. This simplified table example also illustrates that the scheme and host name may be sufficient for a matching entry, such as in the case of the entry "http://customer3.webserving.com," or the host name alone may be sufficient, such as in the case of the entry "fastnet.com." This simplified table example also illustrates that there may be a null entry in the table, as in the case of "http://customer2.webserving.com"; in this simplified table example, the null entry signifies use of the standard preconfigured TCP protocol attributes. Note that an actual table in the primary embodiment would contain more information than the simplified example table shown here, as discussed.

The table **220** can be modified from time to time, including adding new entries, changing the contents of existing entries, and deleting entries. When a new version of the table **220** is created, copies of the new version of the table **220** are distributed to the servers **206** in the global group of cache servers. Optionally, the table **220** can have an expiration date and time, after which the server **206** will stop using the table **220** if it is out-of-date and has not been replaced with an updated table **220**; under this condition, the server **206** would then use the standard TCP protocol attributes for connections until the server **206** received an updated or unexpired table **220**.

In the primary embodiment, the conditional protocol control information is combined with other information, such as customer billing codes and other customer-specific information, that is distributed to all servers **206**, in order to minimize the number of tables that are distributed to, and reconciled throughout, the global group of servers. This embodiment has the advantage of simplifying administration, change management, and rollback in the event that distribution of a new table **220** causes a problem or error condition, but the disadvantage that the conditional protocol adaptation is the same for any given entry in the table **220** at every server **206**, regardless of the server's location or other factors. In other embodiments, as discussed, the conditional protocol control information can vary from one server **206** to another.

Referring to FIG. **4**, an embodiment of a process for potentially modifying protocol attributes on a connection-by-connection basis is shown. The depicted portion of the process begins in block **416** where a uniform resource indicator (URI) is requested by the client **102**. The cache function **208** of the server **206** receives the request for the content object. The URI is evaluated by the protocol attribute selector **212** to find a match to something in the table **220**. The table **220** is queried in block **424** for any attributes. Retrieved attributes are communicated to the TCP handler **214** in block **428**. The connection is established in block **432** according to the selected attributes to connect the end user system **102** with the server **206**. The content object is delivered in block **436**. This process is performed on each URI such that each connection or socket can be independently controlled, if desired. Indeed, two different end user systems **102** could request the same content object and it could be delivered in a very different manner with different selected protocol attributes for each user.

Referring to FIG. **5**, a block diagram of an embodiment of a content delivery system **500** is shown. Content delivery system **500** operates in a manner that is similar to the content delivery systems of FIGS. **1-2** and therefore the description of those systems is also applicable to the present embodiment. While continuing reference will be made to the preceding embodiments, in the interest of clarity, the discussion of common elements and functionality will not be repeated.

As illustrated, end user computers **102** access the global internet **104** through autonomous systems **232**. Autonomous systems **232** may include internet service providers which offer end users access to the global internet **104** over a private communication network. Different providers may offer different types of service and may serve different geographic areas. For example, autonomous system AS1 can represent a DSL communication network such as those operated by AT&T or Qwest Communications, or it could be a cable access network such as those operated by Cox Communications in the United States, or by Rogers Communications in Canada. Autonomous system AS2 could be a satellite communication network, a cellular network, a WiMAX (IEEE 802.16) network, Wi-Fi™ (IEEE 801.11) access, and the like. Depending upon the underlying communications technology, autonomous systems **232** can present different network characteristics that are relevant to the performance of a transport layer protocol such as TCP.

Content delivery servers **206** are also connected to the global internet **104** and can be connected to corresponding autonomous systems **232**. As shown, content server **206-1** is connected to autonomous system **232-1** by router **236-1** and content server **206-2** is connected to autonomous system **232-2** by router **236-2**. Routers **236** thus provide direct links L1, L2 between servers **206** and their corresponding autonomous systems **232**. In some embodiments, servers **206-1**, **206-2** can be edge servers that are collocated with the autonomous system network infrastructure and provide large bandwidth and fast response times for content distribution to end users in a particular location. In an exemplary embodiment, each server **206** is configured to handle approximately 2000 connections per second and can support a 10 Gbps link to its corresponding autonomous system. Of course, the number of servers, number of connections, and data rates may vary based on the location served, traffic patterns, hardware capabilities, and other factors.

Servers **206** also communicate with origin server **240**. Origin server **240** can act as a source of the content distributed to end users. For example, servers **206** may cache content received from origin server **240** and may use the cached content to fulfill end user requests. If requested content is not found in their respective caches, servers **206** can send a request for the missing content to origin server **240**. When requesting content, servers **206** can report information about the content request as well as the conditionally adapted protocol parameters to the origin server **240**. The origin server **240** can collect, compile, and distribute information to servers **206** for use in adapting connection parameters. For example, the origin server **240** can function as part of a CPIMDS and can distribute URL tables **220** and other protocol performance information to servers **206** as previously discussed.

FIG. **6** is a functional block diagram of content distribution server **206** according one embodiment of the present invention. As shown, server **206** includes a processor **244**, memory **248**, one or more network interfaces **252**, and a data store **220**. Although not shown, server **206** can also include a cache function **208** and a content cache **210** as previously described.

Network interfaces **252** can include a plurality of ports (P1, . . . , PN) for sending and receiving data over a connecting network. In some embodiments, at least one network interface is dedicated to providing a high-bandwidth link to an autonomous system and can be matched to its particular network characteristics. Additional ports and/or network interfaces can provide access to the global internet **104**, origin server **204**, or other parts of a content delivery system.

Processor **244** executes programmable instructions for managing the delivery of content to end user computers **102** and can include one or more general purpose microprocessors or application-specific hardware elements. As shown, processor **244** is coupled to memory **248** which can include any combination of volatile or non-volatile computer-readable storage such as random access memory (RAM), read only memory (ROM), magnetic disk storage, and the like. Memory **248** can provide a data store **220** which, as previously described, can be a table or other data structure including information for modifying transport layer performance parameters. Data store **220** is discussed further in connection with FIG. **8**.

Server **206** can include a number of data sources which respond to content requests from end users. In one embodiment, processor **244** supports a protocol stack that enables changes affecting the performance of the transport layer to be made from higher layers in the stack on a per-connection or per-request basis. This enables server **206** to receive a content request from an end user computer over an existing connection, gather information about the request, and intelligently modify the performance of the connection based on information from the request. In this way, for example, server **206** can modify TCP performance parameters based on known characteristics of the connecting network, the geographic location of the end user, metadata associated with the requested content, a service level of the content provider, link utilization, or any combination of these and other factors.

FIG. **7** illustrates a modified TCP protocol stack **700** such as can be included as part of content server **206** according to one embodiment of the present invention. As shown, layers **710-740** correspond roughly to layers of the standard OSI network model. At the lowest layers **710** in the protocol stack (physical and data link layers), data bits are received at the network interface hardware and assembled into data units for delivery to the next higher layer. Here, a network layer **720** includes an IP module that receives IP packets from the lower layers and determines an appropriate transport layer protocol based on their header information. With transport control protocol, network layer **720** sends TCP messages to the transport layer **730** which, in turn, passes application messages to data sources **750** in the application layer **740**.

In operation, an end user computer **102** can establish a TCP connection with content server **206**. The connection can be created using a collection of TCP parameters which are preconfigured at the server and do not necessarily reflect information about the end user or the way in which the end user computer is connected to the server. For example, referring again to FIG. **5**, computer **102-1** can initiate a connection C2.1 with server **206-1** to start a TCP session. Once the connection is established, the end user can send a content request over the connection. As illustrated, a single end user computer **102** can establish multiple connections to a given server **206** and each connection can carry multiple content requests. Protocol stack **700** is configured such that TCP settings can be adjusted on a per-connection or even a per-request basis.

When a content request is received at server **206**, it is conveyed through protocol stack **700** to an appropriate data source **750** in the application layer **740**. For example, a web server **750-1** can respond to requests for web pages, a caching application **750-2** can respond to file requests, and an application server **750-N** can respond to requests for application services. Server **206** can include any number or arrangement of data sources **750** and each data source can respond to multiple content requests.

Each data source **750** can interact with a TCP handler **760** at the transport layer **730** to modify its connections. In one embodiment, TCP handler **760** enables the data sources to modify the timing at which packets are sent to the end user computer to be more or less aggressive based on information gathered from the content request. TCP handler **760** can also modify the pace at which packets are sent. Pacing can indicate that a burst of packets should be sent as soon as possible or that data transmission should be spread out over time. The maximum TCP send window ("send buffer size") for a connection can also be adjusted. For example, in some embodiments, TCP handler **760** can adjust the maximum send buffer to be a multiple of a standard size and can permit buffer utilization to increase until it reaches the maximum size. Alternatively, TCP handler **760** can vary the number of bytes allocated for a particular connection directly.

Generally speaking, each content request has at least two pieces of information. These include a source address of the end user computer and an identifier corresponding to the requested content. For example, a data source **750** that responds to the request for sample URL, http://customer1.webserving.com/folderB/directory/logo.gif, would know the IP address of the requesting computer (e.g., abc.def.ghi) as well as the file name of the requested content (logo.gif). From this starting point, server **206** can obtain additional information from data store **220** with which to modify the transport layer parameters of the TCP connection.

FIGS. **8A-8C** illustrate partial exemplary data elements **810-850** such as can be maintained in data store **220** and used for determining modified parameters for a TCP connection.

FIG. **8A** illustrates exemplary data corresponding to a requested content object. In particular, table **810** can represent a collection of metadata **810**. Metadata for each requested object can include a file name, file size, file type, and content provider as well as TCP attributes associated with the content object. For example, attr**1** can represent pacing on the TCP connection. In some embodiments, pacing is disabled for small files and enabled for large files. This can permit content requests involving a large number of small

files to be fulfilled with quick bursts and can facilitate a more even delivery of large content. TCP attributes can also correspond to the type of data such as whether the requested content is text or video information.

Table **820** includes information about content providers. Each content object can be associated with a content provider. The associated content provider can be identified in the file metadata or it can be determined from the URL of the content request. In some cases, content providers can select a service level for the distribution of their content on the content delivery system. Among other possibilities, requests for content from providers that choose a premium service level can be biased in favor of increased performance. For example, requests for provider ABC's content can be preferentially modified (e.g., attr**6**=fast) and, when appropriate, can be allocated a relatively larger send buffer (e.g., attr**4**=300,000). By contrast, requests for provider DEF's content at the standard service level can be assigned a smaller send buffer (attr**4**=**100**, **000**) and modified only on a best efforts basis (attr**6**=slow). Service level can also set on a per-request basis. For example, a customer can elect a high level of service by adding information to the request query string.

FIG. **8**B illustrates exemplary data corresponding to the network address of an end user computer. Table **830** provides an association between IP address, geographic location, and autonomous system number. A source IP address can be obtained from TCP header information and used to identify a geographic region of the end user computer. The geographic region can be a city, state, country, or continent and can provide a rough estimate of the distance or round trip time from the server **206** to the end user computer. In addition, geographic location can also be a rough indicator of service. For example, network communications in Asia may be characterized by a higher latency than network connections in Europe or some other location. These differences can be factored into the TCP attributes so that, for example, more aggressive timing parameters can be used with Asia-based IP addresses.

Table **830** can also store information about primary routes to particular locations. For example, a considerable amount of network traffic destined for South America passes through servers in Florida and other primary gateway locations. The attributes in table **830** can be biased to optimize TCP performance based on conditions at these gateway servers. For example, TCP timing parameters may be adjusted based on traffic statistics and load along a primary route such that transmit timing for South American connections is made more or less aggressive. Many other location-specific adjustments are possible within the scope of the present invention. In some embodiments, cost and path information can also be included. As an example, transit charges and other direct costs of providing service can be tracked as well as indirect or resource costs.

Table **840** provides information about the autonomous systems. Server **206** can determine an autonomous system (AS) number for an end user computer based on the source address of a content request. The AS number, in turn, can be used to obtain additional information for modifying TCP parameters. For example, if it is known that a particular AS is associated with a type of network, the characteristics of the network technology can be used to determine appropriate TCP parameters for a connection. Cable networks can have a relatively high bandwidth and may be less prone to saturation than DSL networks. Satellite connections, on the other hand, are typically associated with high latency. Server **206** can take advantage of these characteristics by matching timing and pacing parameters to the particular type of network.

When server **206** is collocated at an AS data center, link statistics can be maintained and used to determine TCP parameters. As a link nears full capacity, for example, it may be inappropriate to increase the timing or send buffer size of connections. In some embodiments, the preconfigured TCP parameters are used when link utilization exceeds a predetermined threshold. Thus, among other possibilities, the attributes associated with AS information can indicate whether or not TCP parameters should be modified and, if modification is appropriate, which parameters are best suited for known characteristics of the AS network.

FIG. **8**C illustrates exemplary data corresponding to server utilization. In some embodiments, each content distribution server **206** has a limited amount of bandwidth and is intended to support a certain network load. As the server approaches its limits, it may be appropriate to scale back on resource allocation to TCP connections. Conversely, when the server experiences a light load, it may be appropriate to allocate more system resources to improving the performance of TCP connections. Table **850** provides information for judging load at a content distribution server, including a bandwidth allocation (BW) measures and a connection rate (CPS).

By way of illustration, assume that a particular server **206** can support up to 2,000 connections per second and has available bandwidth of 1 Gbps. When connection rates and bandwidth usage are low, more resources are available for modifying connections. In that case, it may be appropriate to use more aggressive timing, larger buffers and other performance enhancements. Thus, for example, connections to server E1 may be modified by increasing the maximum send buffer size to 300,000 bytes and biasing towards aggressive timing and/or pacing utilization. On the other hand, server **E56** is nearing full capacity and may therefore bias new connections to the preconfigured TCP parameters.

As will be readily appreciated, many different combinations of factors can affect when and how a TCP connection is modified. Different weights and precedence can be assigned to the different types of information available from the data store **220**. For example, system resources may have the highest precedence, followed by service level, and then by metadata and AS factors, and finally by geographic considerations. Across categories, different weights may be assigned to the attributes so that a data source **750** can determine modified TCP parameters based on the net effect of some or all of the available information.

FIG. **9** shows exemplary performance profiles **900** such as can be utilized to modify TCP parameters according to embodiments of the present invention. Rather than determining parameters by combining individual factors, server **106** can include predetermined profiles for content requests. In the example, profile P1 provides TCP settings for sending large files to nearby (low-latency) users. As illustrated, a performance increase can be realized by pacing such connections and allowing the TCP send buffer to grow very large. Relatively less aggressive timing adjustments are needed due to the low latency factor. Profile P2, on the other hand, represents large file transfers to a latent user. In that case, pacing is still used with the transfer, but more aggressive transmit timing may help to compensate for latency and an intermediate send buffer may be appropriate. Profile P3 can be used to transfer small files. With small files, it may be desirable to disable pacing and transmit files in bursts. As a result, a large send buffer may not be needed.

FIG. **10** shows an embodiment of a process **1000** for modifying protocol attributes on a connection-by-connection or request-by-request basis. Process **1000** can be performed by a data source **750** or by the caching function **208** of a content

distribution server **206**. At block **1010**, a content request R1 is received over an existing connection C1 at the server. The request can include the URI of a content object. The content object may be available in content cache **210**, or from origin server **240**, or from some other server accessible through the content delivery system.

The content request R1 can be conveyed through the lower layers of protocol stack **700** to the appropriate data source **750** in the application layer **740**. Since server **206** is capable of modifying transport layer parameters on a connection-by-connection and even a request-by-request basis, the present process can be repeated for each new request (e.g., R2/C1) and/or each new connection (e.g., R1/C2) as determined by the data source **750** or caching function. Note also that that server **206** need not be dedicated to serving a particular type of content but can deliver files, images, video, or any other content available through the content delivery system.

After the request for content is received, the responding data source **750** determines whether the transport layer parameters used with the connection and/or request should be modified. Modifying the transport layer parameters is completely transparent to the end user; the end user is not required to install software or monitoring applications to receive a performance benefit.

At block **1015**, the server **206** makes an initial determination as to whether system load exceeds a predetermined threshold TH1. For example, the responding data source **750** can query the information in table **850** to determine current bandwidth usage and system load. If the system is experiencing a heavy load, pre-configured TCP parameters may be used. In that case, the process ends at block **1060**. On the other hand, if system load is below threshold TH1, a further determination is made as to link utilization. This can involve, for example, accessing the information in table **840**. If link utilization exceeds a predetermined threshold TH2, then the process can terminate at block **1060** and preconfigured TCP parameters can be used with the connection/request.

When there is sufficient system resources and link capacity, a determination can be made regarding the TCP parameters based on file size. The size of a requested file can be determined by accessing the metadata of table **810**. At block **1025**, the file size is compared to a threshold value TH3 to determine if it is a "large" file. If the file is not a large file, then at block **1030** it is compared to another threshold TH4 to determine if it is a "small" file. If the requested file does not fit in either category, then the preconfigured TCP settings may be used. Otherwise, for small files, pacing can be disabled and an appropriate send buffer size can be determined at block **1035**. Thereafter, at block **1065**, the responding data source or cache application directs the TCP handler to modify the connection for the small-file transfer.

With large files, it can be useful to make a further determination as to latency. At block **1045**, a round trip time (RTT) from the server to the end user computer is determined. This can be done by sending ICMP messages to the end user's address and measuring the response time. If RTT is less than a predetermined threshold TH5, then the connection may be characterized as low-latency. In that case, a relatively large send buffer size and less aggressive TCP timing may be appropriate. At block **1050**, these settings are determined by the data source or cache application either based on information from individual items in data store **220** or by selecting a performance profile. On the other hand, if RTT exceeds the threshold, the connection may be characterized as high-latency. At block **1055**, parameters for the large-file, high-latency transfer are determined. At block **1065**, the data source or cache function modifies the connection through the TCP handler for the large file transfer.

FIG. **11** shows an additional embodiment of a process **1100** for modifying protocol attributes. Process **1100** can be performed by a data source **750** or by the caching function of content distribution server **206**. The process begins at block **1110** when a request R1 is received from an end user computer over connection C1. As previously noted, the process can be repeated for each new request (e.g., R2/C1) and/or each new connection (e.g., R1/C2) as determined by the data source **750** or caching function.

At block **1115**, the IP address of the client is determined and the data source or cache function begins to gather information for modifying the connection. Initially, a geographic location and autonomous system of the end user computer are determined based on the IP address (blocks **1120-1125**). If the server has a dedicated link to the AS, link utilization is determined at block **1130** and compared to a predetermined threshold TH2. When the link capacity is below the threshold, the process terminates and the standard or pre-configured TCP parameters are used for the connection/request. If link utilization does not apply to the connection, or if link utilization is below threshold TH2, the process continues.

In this embodiment, a predetermined profile is selected based on the geographic location of the client and the type of connection. For locations in the United States served by cable access networks, blocks **1135-1140**, a first geographic performance profile G1 can be used. Relatively low latency may be assumed for US locations and this profile can adjust TCP timing to take advantage of the relatively high burst capability of cable networks. For locations in the United States served by digital subscriber line (DSL) networks, blocks **1145-1150**, a second geographic performance profile G2 can be used. This profile may use slightly less aggressive timing with a relatively large send buffer. Finally, for US locations served by satellite networks, blocks **1155-1160**, a third geographic profile G3 can be used. This profile may assume high latency but reliable delivery and therefore use relatively more aggressive TCP timing and an intermediate send buffer size.

Customized profiles can be used for non-US locations or when autonomous system information is not available as shown by blocks **1165-1170**. For example, a China-specific profile or an Asia-specific profile can be developed based on historical network performance measures. Similarly, where a primary route to a particular destination is known, profiles may be developed that are customized for the appropriate connecting network elements. When the appropriate geographic performance profile has been selected, the data store or cache function modifies the connection accordingly.

Throughout this document, the terms content delivery and content download are used and can mean either file download or streaming delivery. Additionally, a content object can be either a file or a stream. For example, the content object could be a voice call stream, a video stream, an image file, a music file, a live concert, an animation, an advertisement, a web page, a slide show, data file, hosted software, transactions, executable software or the like. Content can be static or dynamic, can pre-exist on the server, can be created on the server, or can be created or obtained elsewhere in response to a request from a client.

A number of variations and modifications of the disclosed embodiments can also be used. For example, some of the above embodiments discuss use of the TCP protocol or a transport-layer protocol. Other protocols could be modified on a connection-by-connection or request-by-request basis in other embodiments. Also, connection parameters can be modified based on additional information gathered from or

associated with content requests such as HTTP request headers (e.g., content-length, cookies, content-type, user agent, etc.), transport layer security (e.g., HTTPS), layer 2 addressing (e.g., the MAC address of the router from which the request was received), port number, IP properties (e.g., TOS—terms of service), hostname, and whether or not a request successfully passed through a rewrite process.

While the principles of the disclosure have been described above in connection with specific apparatuses and methods, it is to be clearly understood that this description is made only by way of example and not as limitation on the scope of the disclosure.

What is claimed is:

1. A network connection method for delivering content, the network connection method comprising:
- receiving a first request for content from a network at a server;
- analyzing the first request for content to determine first attributes, wherein analyzing the first request comprises comparing a first uniform resource indicator (URI) with an alphanumeric string to correlate the first URI with the first attributes;
- configuring a first connection for serving the content between the server and a first node;
- configuring a protocol of the first connection according to the first attributes, wherein the protocol that is configured is a transport layer protocol;
- receiving a second request for content from the network at the server;
- analyzing the second request for content to determine second attributes;
- configuring a second connection for serving the content between the server and a second node; and
- configuring the protocol of the second connection according to the second attributes, wherein the first attributes affect operation of the protocol differently than the second attributes affect operation of the protocol.

2. The network connection method for delivering content as recited in claim 1, wherein the protocol is transmission control protocol (TCP).

3. The network connection method for delivering content as recited in claim 1, wherein the processes of analyzing the first request and analyzing the second request are both performed away from the server.

4. The network connection method for delivering content as recited in claim 1, wherein the first connection and second connection serve the content at least partially coextensive in time.

5. The network connection method for delivering content as recited in claim 1, wherein the content in the first-listed analyzing limitation is different from the content in the second-listed analyzing limitation.

6. A system for managing delivery of content over a network, the system comprising:
- protocol handler managing a first connection and a second connection over the network using a protocol, wherein:
  - the protocol operates at an transport layer,
  - the protocol handler is configured to use first attributes for the first connection, and
  - the protocol handler is configured to use second attributes for the second connection;
- a store holding a plurality of attributes; and
- a protocol attribute selector, configured to:
  - receive first information relating to a first request for content wherein the first information is derived from a first uniform resource indicator (URI) associated with the first request for content,
  - query the store for first attributes corresponding to the first information,
  - program the protocol handler with the first attributes for the first connection,
  - receive second information relating to a second request for content,
  - query the store for second attributes corresponding to the second information, and
- program the protocol handler with the second attributes for the second connection.

7. The system for managing delivery of content over the network as recited in claim 6, wherein the protocol is transmission control protocol (TCP).

8. The system for managing delivery of content over the network as recited in claim 6, further comprising a protocol stack that sends the content to a first node using the first connection and a second node using the second connection.

9. The system for managing delivery of content over the network as recited in claim 6, wherein:
- the protocol handler is part of a server, and
  - at least one of the store and the protocol attribute selector is located away from the server.

10. The system for managing delivery of content over the network as recited in claim 6, wherein the first connection and second connection serve the content at least partially coextensive in time.

11. The system for managing delivery of content over the network as recited in claim 6, wherein the content of the first connection is different from the content in the second connection.

* * * * *

Exhibit B

US008750155B2

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

(10) **Patent No.: US 8,750,155 B2**
(45) **Date of Patent: *Jun. 10, 2014**

(54) **CONDITIONAL PROTOCOL CONTROL**

(75) Inventors: **Bradley B. Harvell**, Chandler, AZ (US); **Joseph D. DePalo**, Peoria, AZ (US); **Michael M. Gordon**, Paradise Valley, AZ (US); **Jason L. Wolfe**, Gilbert, AZ (US)

(73) Assignee: **Limelight Networks, Inc.**, Tempe, AZ (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 169 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **13/595,904**

(22) Filed: **Aug. 27, 2012**

(65) **Prior Publication Data**

US 2013/0060893 A1 Mar. 7, 2013

Related U.S. Application Data

(63) Continuation of application No. 12/572,981, filed on Oct. 2, 2009, now Pat. No. 8,274,909, which is a continuation-in-part of application No. PCT/US2009/038361, filed on Mar. 26, 2009.

(51) **Int. Cl.**

G01R 31/08	(2006.01)
G06F 11/00	(2006.01)
G08C 15/00	(2006.01)
H04J 1/16	(2006.01)
H04J 3/14	(2006.01)
H04L 1/00	(2006.01)
H04L 12/26	(2006.01)
H04L 12/56	(2006.01)

(52) **U.S. Cl.**

CPC ***H04L 47/193*** (2013.01)
USPC **370/252**; 370/389; 370/412; 370/466

(58) **Field of Classification Search**

CPC H04L 47/193; H04L 69/326; G06F 15/16
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

2002/0099844	A1*	7/2002	Baumann et al.	709/232
2007/0067424	A1*	3/2007	Raciborski et al.	709/223
2008/0031149	A1*	2/2008	Hughes et al.	370/252
2008/0225721	A1*	9/2008	Plamondon	370/235
2010/0131671	A1*	5/2010	Kohli et al.	709/233

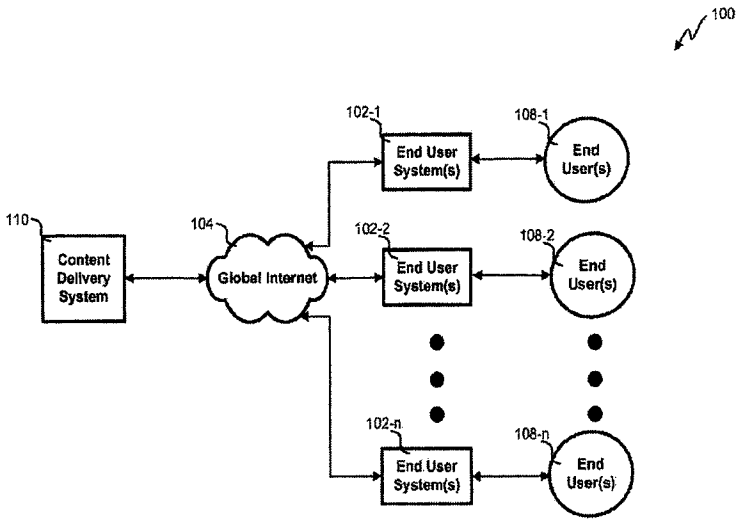
* cited by examiner

Primary Examiner — Jay Y Lee
(74) *Attorney, Agent, or Firm* — Kilpatrick Townsend & Stockton LLP

(57) **ABSTRACT**

Techniques for modifying the performance of a transport layer protocol in response to a request for content are disclosed. A connection can be established between a content distribution server and an end user computer according to preconfigured parameters. When a request for content is received over the connection, the content distribution server can determine one or more parameters relating to the performance of the connection using information from the request. The content distribution server can modify the connection at the transport layer according to the one or more parameters. Thereafter, the transport layer can manage delivery of the requested content to the end user computer in accordance with the modified parameters. In various embodiments, the content distribution server includes a modified TCP protocol stack which adjusts timing, pacing, and buffer allocation associated with a connection in response to requests from an application-layer data source.

20 Claims, 14 Drawing Sheets



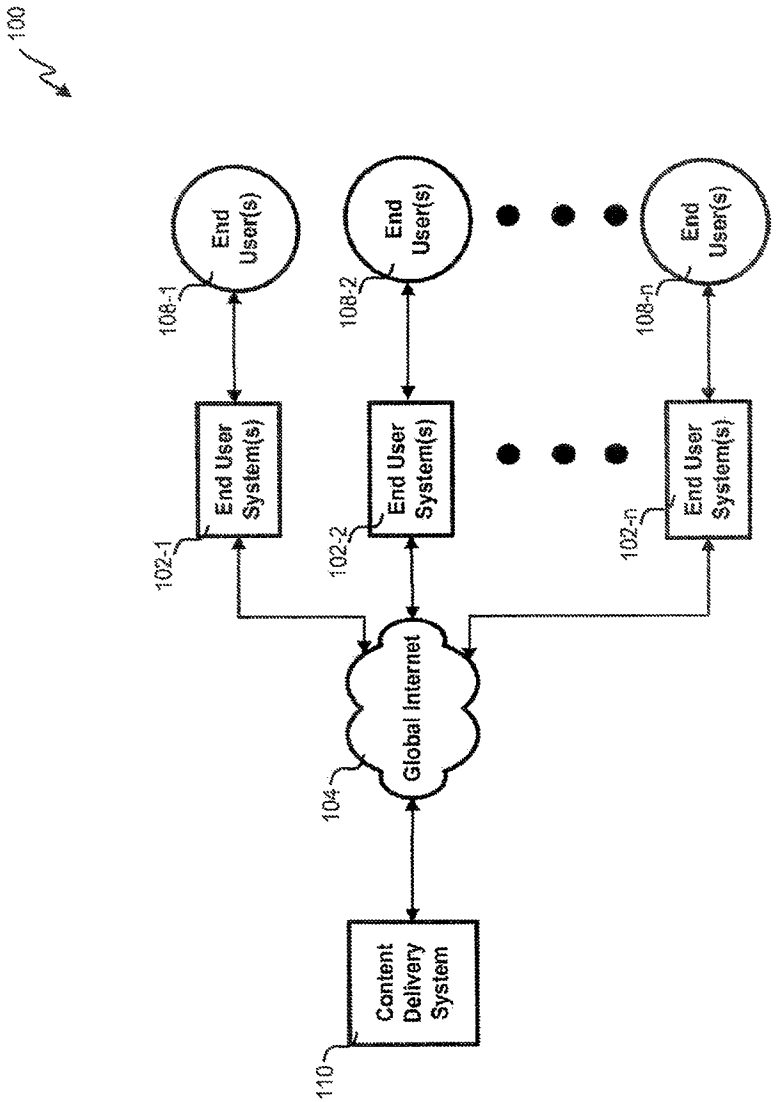

100
End User(s)
108-1
End User(s)
108-2
End User(s)
108-n
End User System(s)
102-1
End User System(s)
102-2
End User System(s)
102-n
Global Internet
104
Content Delivery System
110

FIG. 1

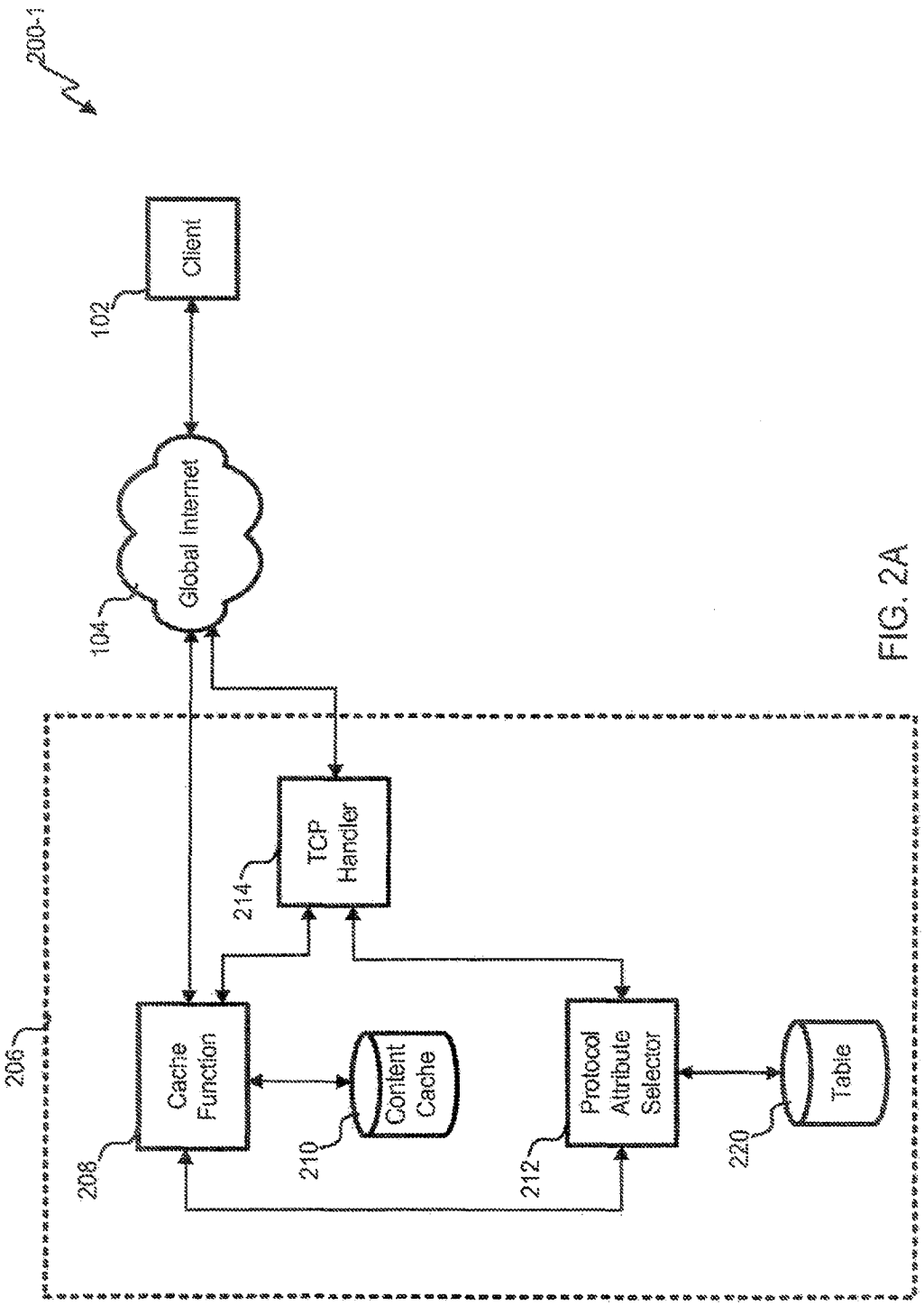

200-1
Client
102
Global Internet
104
TCP Handler
214
206
Cache Function
208
Content Cache
210
Protocol Attribute Selector
212
Table
220

FIG. 2A

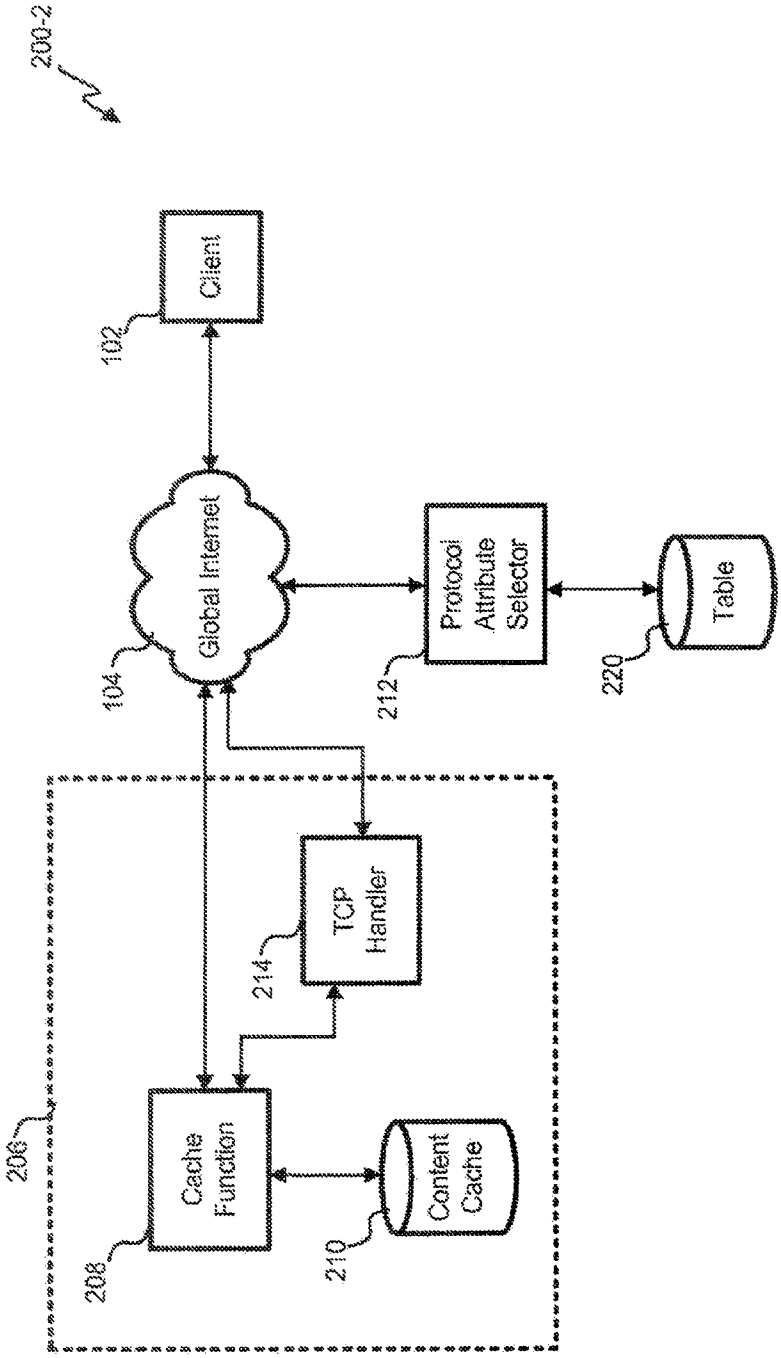

200-2
Client
102
Global Internet
104
Protocol Attribute Selector
212
Table
220
TCP Handler
214
206
Cache Function
208
Content Cache
210

FIG. 2B

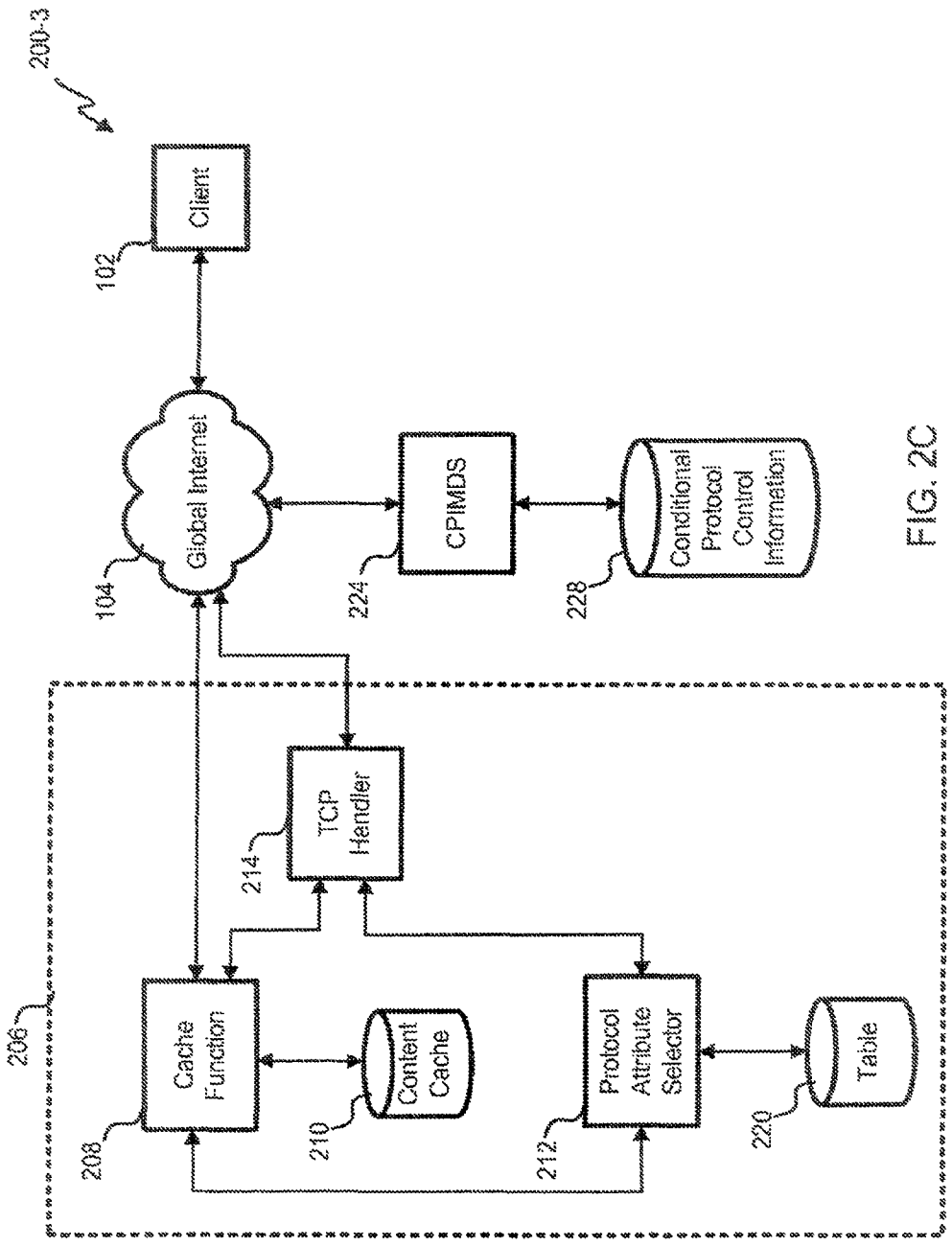

200-3
Client
102
Global Internet
104
CPIMDS
224
Conditional Protocol Control Information
228
TCP Handler
214
206
Cache Function
208
Content Cache
210
Protocol Attribute Selector
212
Table
220

FIG. 2C

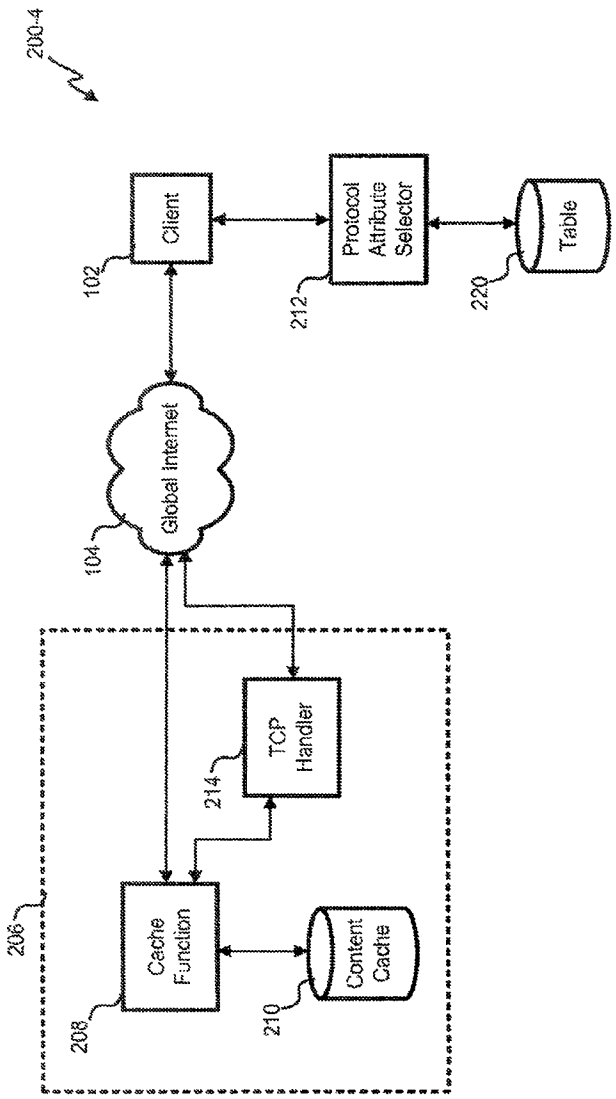

200-4
Client
102
Protocol Attribute Selector
212
Table
220
Global Internet
104
TCP Handler
214
206
Cache Function
208
Content Cache
210

FIG. 2D

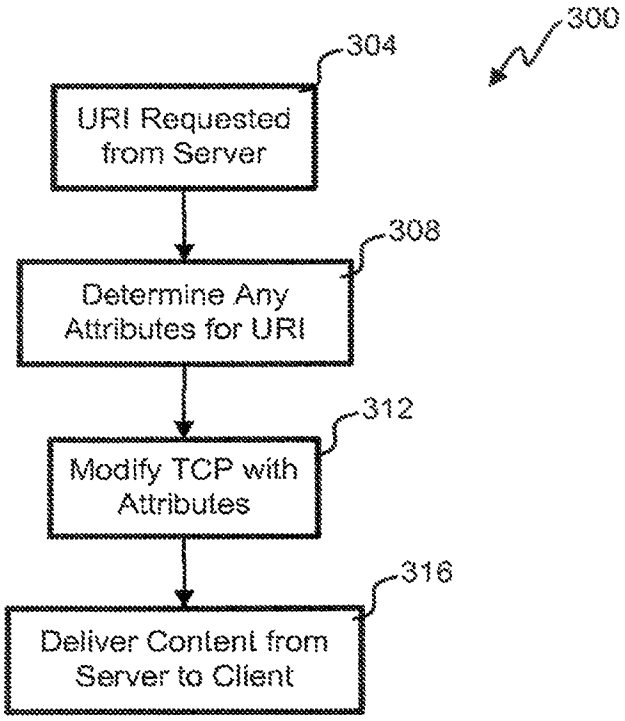

300
304
URI Requested from Server
308
Determine Any Attributes for URI
312
Modify TCP with Attributes
316
Deliver Content from Server to Client

Fig. 3

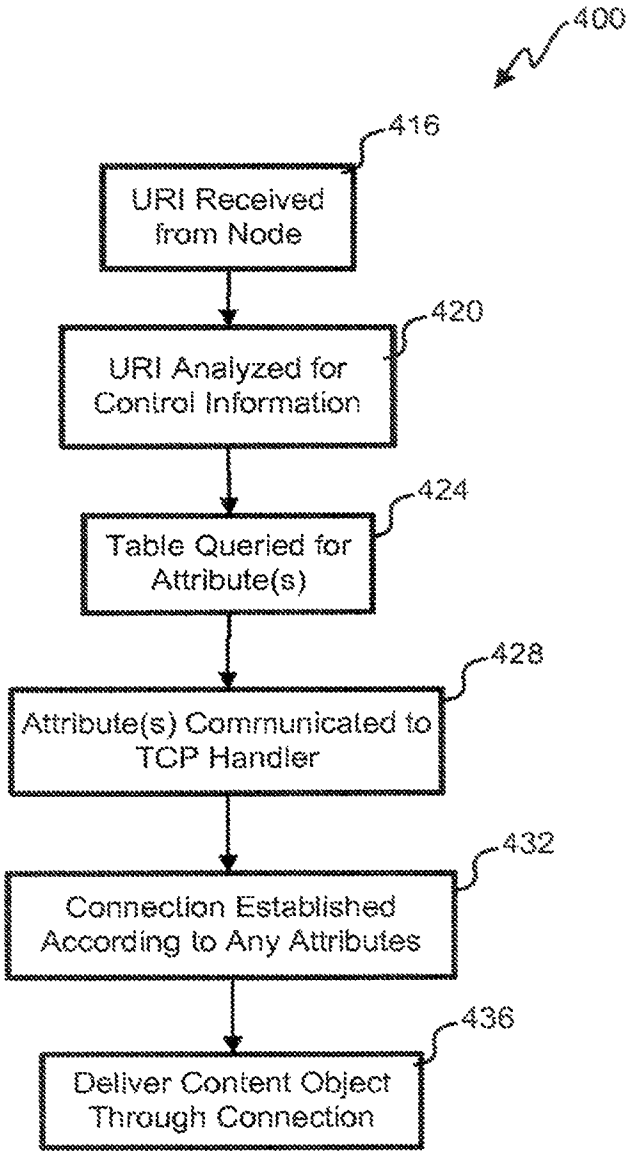

400
416
URI Received from Node
420
URI Analyzed for Control Information
424
Table Queried for Attribute(s)
428
Attribute(s) Communicated to TCP Handler
432
Connection Established According to Any Attributes
436
Deliver Content Object Through Connection

Fig. 4

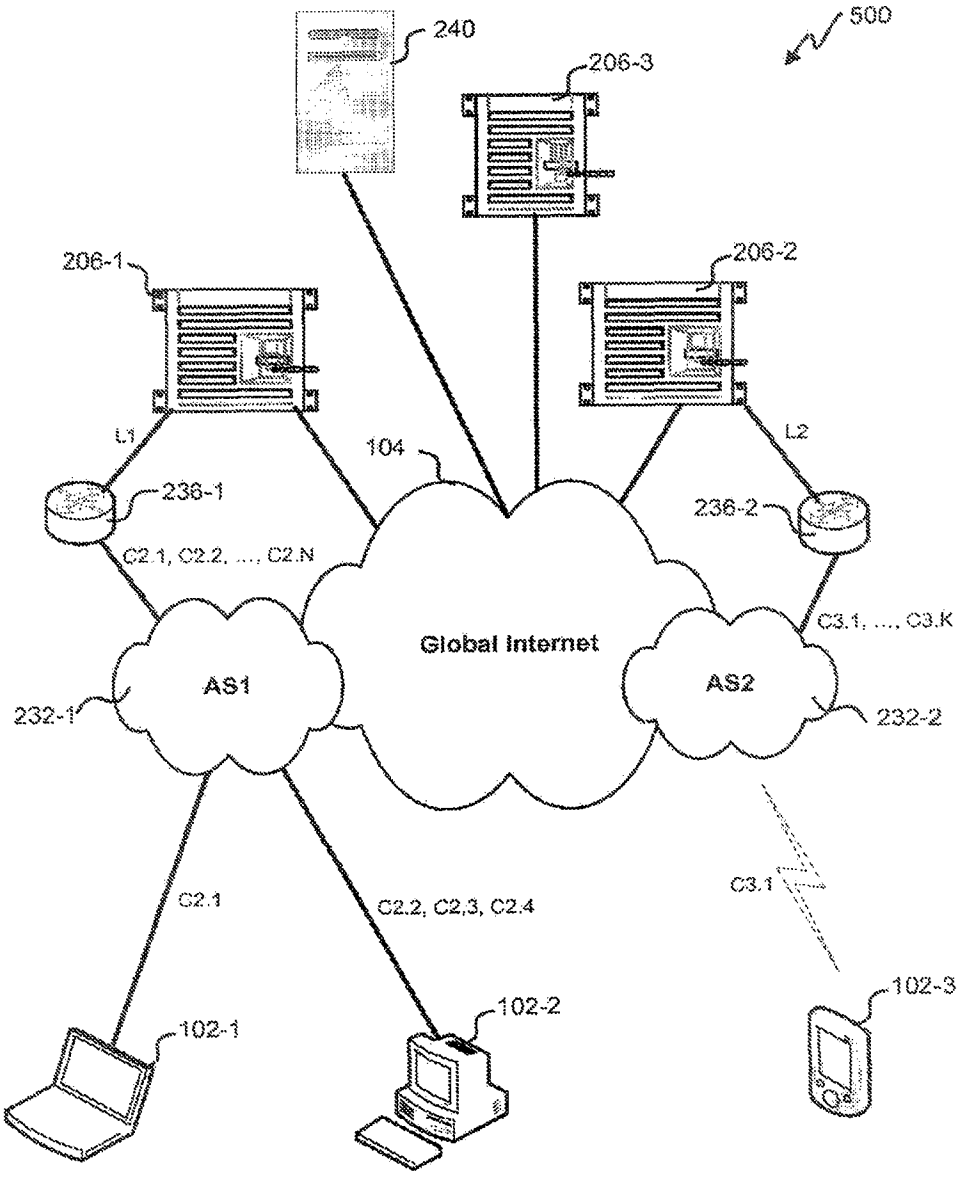

500
240
206-3
206-1
206-2
L1
L2
104
236-1
236-2
C2.1, C2.2, ..., C2.N
C3.1, ..., C3.K
Global Internet
AS1
AS2
232-1
232-2
C2.1
C2.2, C2.3, C2.4
C3.1
102-1
102-2
102-3

FIG. 5

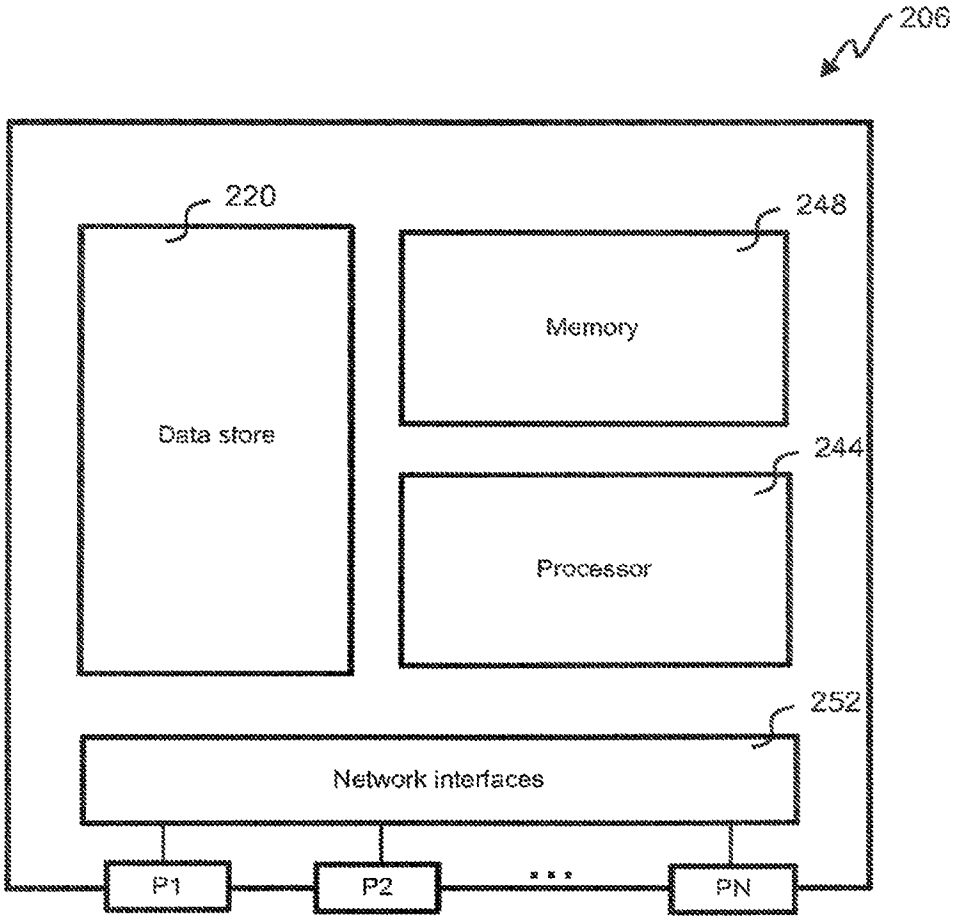

206
220
248
Memory
Data store
244
Processor
252
Network interfaces
P1
P2
...
PN

FIG. 6

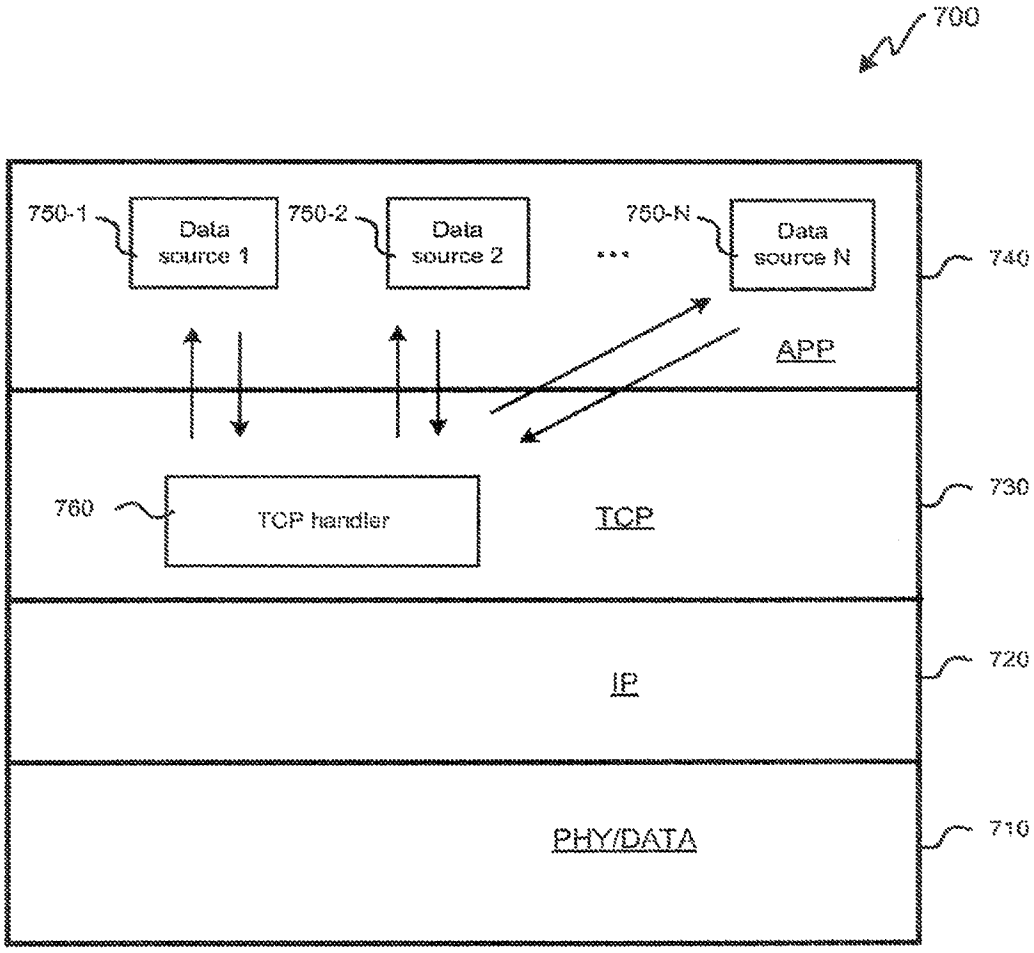

700
750-1
Data source 1
750-2
Data source 2
750-N
Data source N
740
APP
760
TCP handler
TCP
730
IP
720
PHY/DATA
710

FIG. 7

810

File name	Provider	File Size	Type	***	Attribs
logo.gif	ABC	50k	image	***	attr1=no
index.html	ABC	100k	text	***	attr1=no
movie.mpg	ABC	1GB	video	***	attr1=yes

820

Provider	Service Level	***	Attribs
ABC	Premium	***	attr4=100000, attr6=fast
DEF	Standard	***	attr4=300000, attr6=slow

FIG. 8A

830

IP Address / Prefix	AS #	Country	***	Attribs
123.012.034.0 /24	12345	US	***	attr3=25
234.079.091.0 /24	34567	US	***	attr3=40
169.234.056.078	UNK	ASIA	***	attr3=35

840

AS #	Location	Service Type	Link utilization	***	Attribs
12345	Tempe, AZ	DSL	43%	***	attr6=fast, attr3=25
34567	Chicago, IL	Cable	92%	***	attr6=slow
34567	Atlanta, GA	SAT	--	***	attr3=40

FIG. 8B

850

ServerID	BW	CPS	...	Attribs
E1	725	900	...	attr4=100000, attr6=fast
E56	937	1877	...	attr4=300000, attr6=slow

FIG. 8C

900

Performance Profile	Timing	Pacing	Send Window	Comment
P1	25	1	300000	large file, near user
P2	40	1	175000	large file, latent user
P3	25	0	100000	small file

FIG. 9

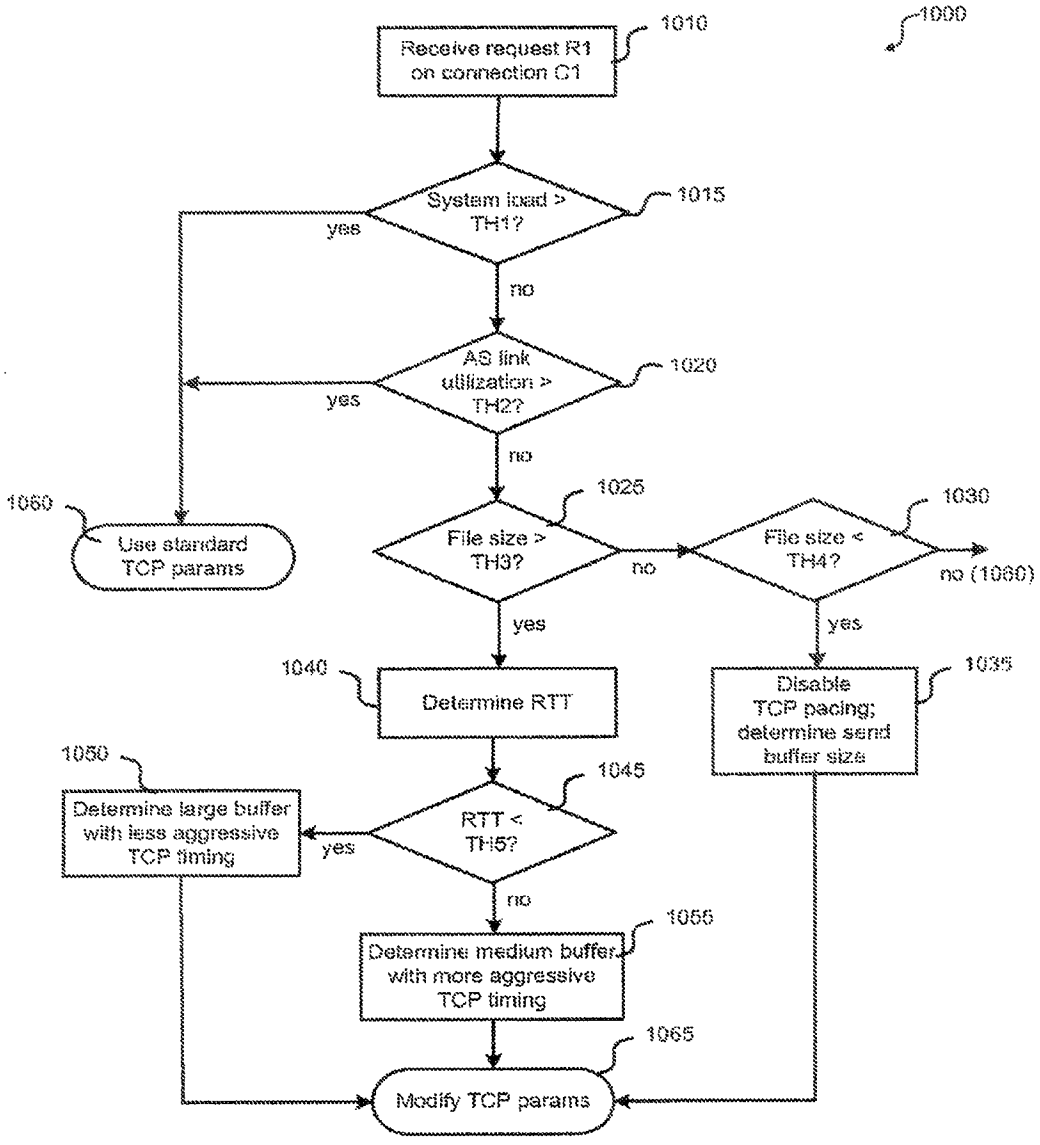

1000
1010
Receive request R1 on connection C1
1015
System load > TH1?
yes
no
1020
AS link utilization > TH2?
yes
no
1025
File size > TH3?
no
1030
File size < TH4?
no (1060)
yes
yes
1060
Use standard TCP params
1040
Determine RTT
1035
Disable TCP pacing; determine send buffer size
1045
RTT < TH5?
1050
Determine large buffer with less aggressive TCP timing
yes
no
1055
Determine medium buffer with more aggressive TCP timing
1065
Modify TCP params

FIG. 10

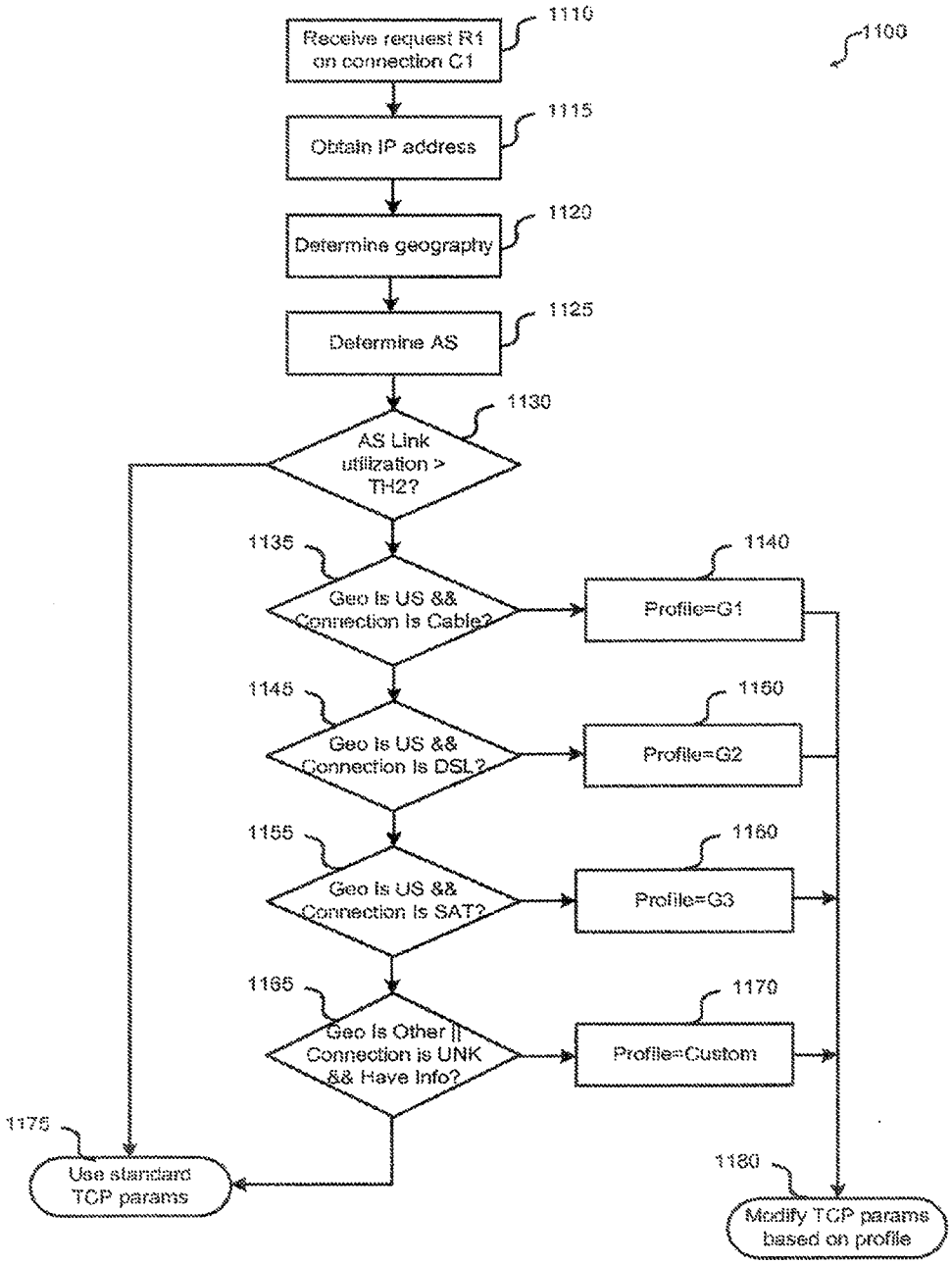

1100
1110
Receive request R1 on connection C1
1115
Obtain IP address
1120
Determine geography
1125
Determine AS
1130
AS Link utilization > TH2?
1135
Geo is US && Connection is Cable?
1140
Profile=G1
1145
Geo is US && Connection is DSL?
1150
Profile=G2
1155
Geo is US && Connection is SAT?
1160
Profile=G3
1165
Geo is Other || Connection is UNK && Have Info?
1170
Profile=Custom
1175
Use standard TCP params
1180
Modify TCP params based on profile

FIG. 11

CONDITIONAL PROTOCOL CONTROL

CROSS-REFERENCES TO RELATED APPLICATIONS

This application is a continuation of U.S. patent application Ser. No. 12/572,981, filed Oct. 2, 2009 which is a continuation-in-part claiming priority benefit under 35 U.S.C. §365(c) of International Patent Application Serial No. PCT/US2009/038361, filed Mar. 26, 2009. Each of the above-listed applications is hereby incorporated by reference in its entirety for all purposes.

BACKGROUND

This disclosure relates in general to interoperating nodes in an information processing system, such as an Internet content delivery system or an Internet transaction acceleration system, and, but not by way of limitation, to control of connection protocols.

In an information processing system, including communications networks such as the Internet, two or more nodes can work together, for example exchanging information or sharing resources, using one or more protocols that enable the participating nodes to interoperate. Nodes need not be physically distinct from one another, though they may be; normally, however, nodes are at least logically distinct from one another in at least some respect. Interoperating nodes may be operated or managed by a single common authority or by independent, unrelated authorities. Two or more interoperating nodes are often independently operated or managed; the Internet includes many well known examples of the interoperation of two or more independently managed nodes.

A protocol can be standardized such that a node using the standard protocol should be able to interoperate, at least at the level of the protocol, with any other node using the standard protocol. Standard protocols that become widely adopted can permit a node to interoperate with many other nodes. One such widely adopted standard protocol on the Internet is the Transmission Control Protocol (TCP), which today enables almost every device on the Internet to interoperate with almost every other device. TCP operates at the connection layer and enables nodes to interoperate with other nodes by establishing communications connections.

Standard protocols often employ the use of attributes, such as configurable parameters and selectable algorithms, to permit the protocol to operate effectively in various situations. For example, TCP controls message size, the rate at which messages are exchanged, and factors related to network congestion through the use of attributes, including both by the use of parameters, such as the receive window field used in sliding window flow control and the retransmission timer, and by the use of algorithms, such as slow-start, congestion avoidance, fast retransmit, and fast recovery algorithms. It is often the case, in many standard protocols, that at each node the initial protocol attribute settings to be used for all the communication connections at the node can be independently specified by the operator of the node.

A protocol can also be customized, which in general requires that each node have installed customized components to enable the custom protocol. Without the customized components, the node would not be able to fully interoperate with other nodes using the customized protocol. Although it therefore may limit the total number of interoperable nodes, or in the alternative require widespread action to install the protocol customized components, or possibly both, protocol customization is used in order to add function, improve performance, increase flexibility, or modify other characteristics of a standard protocol, or to make available an entirely new customized protocol. Many customized protocols have been proposed for use on the Internet.

SUMMARY

Techniques for modifying the performance of a transport layer protocol in response to a request for content are disclosed. A connection can be established between a content distribution server and an end user computer according to preconfigured parameters. When a request for content is received over the connection, the content distribution server can determine one or more parameters relating to the performance of the connection using information from the request. The content distribution server can modify the connection at the transport layer according to the one or more parameters. Thereafter, the transport layer can manage delivery of the requested content to the end user computer in accordance with the modified parameters. In various embodiments, the content distribution server includes a modified TCP protocol stack which adjusts timing, pacing, and buffer allocation associated with a connection in response to requests from an application-layer data source.

In one embodiment, a method for managing delivery of content in a system comprising a server and an end user computer is disclosed. The method includes establishing a first connection at the server for communicating with the end user computer and receiving a request for content from the end user computer over the first connection. The method also includes determining one or more parameters relating to the performance of the first connection using information from the request and modifying the first connection at the transport layer based on the one or more parameters. Modifying the first connection can be done without notifying the end user computer. The method also includes sending the requested content from the server to the end user computer such that the transport layer manages delivery of the content in accordance with the modified parameters.

Optionally, the method includes retrieving metadata associated with a requested file and modifying the first connection based on the metadata. Alternatively or additionally, the method can include selecting a predetermined performance profile for the first connection using the information from the request and modifying the first connection based on the predetermined performance profile. The method can include determining a connection type of the end user computer and a latency characteristic associated with the connection type and modifying the first connection at the transport layer based on the latency characteristic. The method can also include determining a data size of the requested content, measuring a round trip time between the server and the end user computer when the data size exceeds a predetermined value, and modifying the first connection at the transport layer based on the size of the requested content and the round trip time.

In another embodiment, a content distribution server is disclosed. The server includes a network interface, a processor, a protocol handler, and a data source. The network interface includes a plurality of ports for sending and receiving data over a connecting network. The processor is coupled to the network interface and manages a plurality of connections to end user computers. The protocol handler establishes the connections with the end user computers according to preconfigured transport layer parameters of the content distribution server and manages the manner in which data is transmitted over the connections. The data source supplies the requested content. The data source monitors a first connection

for a request, determines one or more modified transport layer parameters based on the request, and directs the protocol handler to modify the first connection independently of the other connections based on the one or more transport layer parameters.

In still another embodiment, a content distribution server is disclosed. The server includes means for sending and receiving data over a connecting network, means for managing a plurality of connections to end user computers, and means for establishing a connection with each end user computer according to preconfigured transport layer parameters. The server includes means for managing data transmission over the plurality of connections, means for modifying a connection based on one or more transport layer performance parameters, and means for supplying requested content to the end user computers over the plurality of connections. The server also includes means for monitoring a first connection for a content request, means for determining the one or more transport layer performance parameters for the first connection based on the request, and means for sending the requested content over the first connection modified by the one or more transport layer performance parameters.

In yet another embodiment, a computer program product comprising a computer-readable medium is disclosed. The computer-readable medium is encoded with one or more sequences of one or more instructions which, when executed by a processor, perform steps of establishing a first connection at the server for communicating with an end user computer and receiving a request for content from the end user computer over the first connection. The instructions operate to determine one or more parameters relating to the performance of the first connection based on information from the request and to modify the first connection at the transport layer using the one or more parameters without notifying the end user computer. Additionally, the instructions operate to send the requested content from the server to the end user computer such that the transport layer manages delivery of the content in accordance with the modified parameters.

Further areas of applicability of the present disclosure will become apparent from the detailed description provided hereinafter. It should be understood that the detailed description and specific examples, while indicating various embodiments, are intended for purposes of illustration only and are not intended to necessarily limit the scope of the disclosure.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** depicts a block diagram of an embodiment of a content delivery system.

FIGS. **2**A, **2**B, **2**C, and **2**D depict block diagrams of embodiments of a content download pair that sends content from a server to a client.

FIG. **3** illustrates a flowchart of an embodiment of a process for modification of the TCP protocol for various connections to a server.

FIG. **4** illustrates a flowchart of an embodiment of a process for modifying protocol attributes potentially on a connection-by-connection basis.

FIG. **5** shows aspects of a content delivery system.

FIG. **6** is a block diagram of an embodiment of a content distribution server.

FIG. **7** shows an exemplary content distribution server protocol stack.

FIGS. **8**A, **8**B, and **8**C show exemplary data elements such as can be used with a content distribution server.

FIG. **9** shows exemplary performance profiles such as can be used with a content distribution server.

FIG. **10** is a flowchart of a process for modifying transport layer protocol attributes.

FIG. **11** is a flowchart of process for modifying transport layer protocol attributes.

In the figures, similar components and/or features may have the same reference label. Further, various components of the same type may be distinguished by following the reference label by a dash and a second label that distinguishes among the similar components. If only the first reference label is used in the specification, the description is applicable to any one of the similar components having the same first reference label irrespective of the second reference label.

DETAILED DESCRIPTION OF EMBODIMENTS

The ensuing description provides preferred exemplary embodiment(s) only, and is not intended to limit the scope, applicability or configuration of the disclosure. Rather, the ensuing description of the preferred exemplary embodiment(s) will provide those skilled in the art with an enabling description for implementing a preferred exemplary embodiment. It being understood that various changes may be made in the function and arrangement of elements without departing from the spirit and scope as set forth in the appended claims.

Referring first to FIG. **1**, a block diagram of an embodiment of an Internet content delivery system **100** is shown. Generally, one or more nodes request content from one or more other nodes. In FIG. **1**, a number of end users **108** respectively use their end user system or client **102** to download and view content objects from the global Internet **104**. The content delivery system **110** has one or more servers that provide content object downloads. The content delivery system **110** can include any number of cache servers, application servers, content servers, service servers, and/or database servers to provide content to the clients **102**. Although this embodiment shows particular communication pairs, other embodiments could communicate between any pair of nodes on a network, including between pairs of clients or between pairs of servers, and yet other embodiments could communicate among more than two nodes, such as in a broadcast or multicast implementation.

With reference to FIGS. **2**A, **2**B, **2**C and **2**D, embodiments of a content download pair **200** that sends content from a server **206** to a client **102** are shown. A primary embodiment described here is the interoperation of two nodes **102**, **206** on the Internet communicating using TCP, one node being a client **102** that requests information, such as web page content, multimedia, or software downloads, and the second node being a server **206** that provides information in response to a request. TCP operates in the transport layer of the seven-layer Open Systems Interconnection (OSI) model. In other embodiments, nodes **102**, **206** interoperate in ways other than communication in a network, such as sharing data within a computer or group of computers across an available system or intersystem interface; interoperate using communications networks other than the Internet **104**, such as a private communications network; interoperate using the Internet **104** or a private network using protocols other than TCP, such as UDP, RTP, multicast protocols, and other standard protocols in the transport layer; interoperate using the Internet **104** or a private network using standard protocols operating in a layer that underlies the transport layer; interoperate, using standard protocols and the Internet or a private network, more than two at a time, such as in clusters or multicast groups; or interoperate, using standard protocols and the Internet or a private network, other than as a client and server, including interop-

erating as peers, as collaborative nodes, or as a group of nodes under the common control of one or more other nodes or under the common control of a controller.

In the primary embodiment, the server **206** conditionally adapts the attributes of the TCP protocol for each TCP connection established by a client **102**. Conditionally adapting the attributes of the TCP protocol does not require changes to standard TCP protocol implementations at every node, does not require special components be installed in the TCP protocol implementation at every node, and therefore does not comprise implementing a customized protocol as previously described; rather, the primary embodiment utilizes the standard TCP protocol and the attributes implemented in it. In other embodiments, the server conditionally adapts the attributes of other transport-layer protocols for each session established by a client **102**; the server conditionally adapts the attributes of other protocols that underlie the transport layer for each session established by a client **102**; the server **206** conditionally adapts the attributes of the TCP protocol or other protocol for groups of connections or sessions established by clients **102**; the server **206** conditionally adapts the attributes of the TCP protocol or other protocol for connections or sessions established by groups or subsets of groups of clients **102**; the client **102** conditionally adapts the attributes of the TCP protocol for each TCP connection established; the client **102** conditionally adapts the attributes of other protocols for each session established; the client **102** conditionally adapts the attributes of the TCP protocol or other protocol for groups of connections or sessions; a node conditionally adapts the attributes of the TCP protocol or other protocol for each connection or session; a node conditionally adapts the attributes of the TCP protocol or other protocol for groups of connections or sessions; a node conditionally adapts the attributes of the TCP protocol or other protocol for connections or sessions established by groups or subsets of groups of nodes; a controller conditionally adapts the attributes of the TCP protocol or other protocol for each connection or session of at least one node of an interoperating group of nodes; a controller conditionally adapts the attributes of the TCP protocol or other protocol for groups of connections or sessions of at least some nodes of an interoperating group of nodes; or, a controller conditionally adapts the attributes of the TCP protocol or other protocol for connections or sessions established by groups or subsets of groups of nodes.

Software, software modifications, or equivalent function, may optionally be implemented at a server, client, or node that sets the conditionally adapted protocol attributes of a connection or session, but need not be implemented at servers, clients, or nodes that passively participate in a conditionally adapted protocol connection or session. Such software, software modifications, or equivalent function will only be needed if existing protocol software or other software on the server, client, or node does not provide a facility for programmatically or similarly changing attributes of the protocol that is used; in this event, software, a software modification, or equivalent facilities to provide such a programmatic or similar interface may be implemented.

Conditionally adapting the protocol for each connection or session, or collection of connections or sessions, results in at least one node that, concurrently or over time, uses a protocol for multiple unrelated connections or sessions wherein the protocol attributes vary, at least initially and sometimes persistently, from one connection or session to another, most often varying differently from any ordinary protocol attribute variations that naturally occur from one connection or session to another through use of the standard protocol implementation among heterogeneous nodes.

In the primary embodiment, TCP connections are established in order to use HyperText Transfer Protocol (HTTP) to communicate information requests from clients **102** to servers **206** and responses from servers **206** to clients **102**. HTTP is a scheme that operates above, and depends on the presence of a functioning and reliable protocol at, the transport layer of the seven-layer model developed in the Open Systems Interconnection (OSI) initiative. Other embodiments use application-layer protocols other than HTTP in conjunction with TCP; use TCP alone, i.e., without HTTP; use other protocols; or, use other application-layer protocols in conjunction with other protocols. HTTP utilizes Uniform Resource Locators (URLs), Uniform Resource Names (URNs), and Uniform Resource Identifiers (URIs) to identify information. URLs are used in the primary embodiment. Other embodiments use URIs, URNs, other identifiers, or other information. A URL begins with the scheme identifier, which identifies the namespace, purpose, and syntax of the remainder of the URL. In the primary embodiment utilizing HTTP, the typical scheme is "http". The scheme is followed by a host field, which contains the IP address or name of the host where the requested information can be found, optionally followed by a port number, optionally followed by a path, which is an HTTP selector, optionally followed by a search portion, which is a query string. The full URL, then, is an alphanumeric string containing the scheme, host field, any optional following strings, and special characters such as ":", "/", and "?" that are reserved for special functions such as designating a hierarchical structure in the URL. Other embodiments could use different application-layer protocols such as Telnet, File Transfer Protocol (FTP), secure HTTP (HTTPS), and Simple Mail Transfer Protocol (SMTP).

In the primary embodiment, the server **206** bases the conditional adaptation of the attributes of the TCP protocol on the alphanumeric URL string provided by the client **102** in its information request. In another embodiment, a server, client or other node bases the conditional adaptation of the attributes of the TCP protocol or other protocol on the application-layer protocol specified or on identifying information, equivalent to a URL, or other information provided in, or characteristic of, an information request, connection, or session. In other embodiments, a server, client or other node bases the conditional adaptation of the attributes of the TCP protocol or other protocol on the IP address of one or more servers, clients, or nodes; on network information associated with the IP address of one or more servers, clients, or nodes, including the Autonomous System (AS) number, identity of network operator, geographic location, logical or physical network location, logical or physical network segment, or network interconnection characteristics associated with the IP address(es) of one or more servers, clients, or nodes; the geographic location of the server, client or node; and/or, the logical or physical network location of the server, client or node; the logical or physical address of the server, client or node; the logical or physical name of the server, client or node; and/or, the network or other path from or to a server, client or node. In other embodiments a server, client or node bases the conditional adaptation of the attributes of the TCP protocol or other protocol on recent network performance measurements, including latency, jitter, packet loss, round trip time, and/or the measured variance in a network performance measurement across multiple samples; on recent measures of utilization of a network, network segment, network interface, or network port; and/or, on recent measurements of performance or utilization of a server, group of servers, or server component(s) such as memory, processor, disk, bus, intersystem interface, and/or network interface. In still other

embodiments, a server, client or node bases the conditional adaptation of the attributes of the TCP protocol or other protocol on temporal factors, including time of day; day of week, month, or year; specific date; occurrence of a holiday or religious observance; occurrence of a temporal event such as a news event or sports event; seasonal occurrence; and/or a scheduled event or time period.

In the primary embodiment, the protocol attribute selector **212** of the server **206** compares the alphanumeric URL string provided by the client **102** in its information request to a table **220** containing partial or whole URLs and identifies the most specific match from left to right that it can find in the table **220**. In another embodiment, the server **206** compares a subset of the alphanumeric string, for example some or all of the characters in the query string, or the characters following the host field up to the first subsequent slash (i.e., "/"), to a table **220**. In another embodiment, the client or node **102** makes a conditional adaptation of protocol attributes, using the alphanumeric URL string or a subset of it. In other embodiments, the alphanumeric URL string or a subset of it is processed to obtain a value or indicator that is used to determine a conditional adaptation of protocol attributes. In other embodiments, the information used to determine the conditional adaptation of protocol attributes is identifying information equivalent to a URL, one or more IP addresses, network information associated with one or more IP addresses, network interconnection characteristics associated with one or more IP addresses, or the geographic location, or logical or physical network location, of a server, client or node. In other embodiments, the information used to determine the conditional adaptation of protocol attributes comprises one or more recent performance measurements or thresholds related to one or more servers, clients, or nodes, or groups of servers, clients, or nodes, or related to one or more networks, network segments, network components, or network interfaces, or groups of network segments, network components, or network interfaces; rates or amounts of variation in one or more performance measurements or thresholds related to one or more servers, clients, or nodes, or groups of servers, clients, or nodes, or related to one or more networks, network segments, network components, or network interfaces, or groups of network segments, network components or network interfaces; rates or amounts of resource utilization, including utilization related to one or more servers, clients, or nodes, or groups of servers, clients, or nodes, or components of one or more servers, clients, or nodes, groups of components of servers, clients, or nodes, or related to one or more networks, network segments, network components, or network interfaces, or groups of network segments, network components, or network interfaces; rates or amounts of variation in resource utilization, including variation in utilization related to one or more servers, clients, or nodes, or groups of servers, clients, or nodes, or components of one or more servers, clients, or nodes, groups of components of servers, clients, or nodes, or related to one or more networks, network segments, network components, or network interfaces, or groups of network segments, network components, or network interfaces; and/or, thresholds of resource utilization, including utilization related to one or more servers, clients, or nodes, or groups of servers, clients, or nodes, or components of one or more servers, clients, or nodes, groups of components of servers, clients, or nodes, or related to one or more networks, network segments, network components, or network interfaces, or groups of network segments, network components, or network interfaces.

In the primary embodiment, a table **220** containing partial or whole URLs for comparison by the protocol attribute selector **212** is stored on the server **206**. In other embodiments, a table **220** containing partial or whole URLs for comparison, or subsets of the alphanumeric URL string used for comparison, is stored remotely from the server **206** such as the embodiment shown in FIG. **2**B and the comparison of the alphanumeric URL string or subset of the alphanumeric URL string is made at the server **206**, or is made at the remote table storage location or at another location and the result of the comparison, or an indicator of the result of the comparison, or the protocol attributes to be used, are returned to the server **206**. In other embodiments, a table **220** or database of information used for comparison is stored at the server **206**, or is stored remotely from the server **206** and one or more comparisons are made at the server **206**, or are made at the remote storage location or at another location, and the results of the comparisons, or one or more indicators of the results of the comparisons, or the protocol attributes to be used, are returned to the server **206**. In yet other embodiments, a table **220** containing partial or whole URLs for comparison, or subsets of the alphanumeric URL string used for comparison, or a database of information used for comparison is stored at a client or node **102** as is shown in FIG. **2**D, or is stored remotely from a client or node and one or more comparisons are made at the client or node **102**, or are made at the remote storage location or at another location and the results of the comparisons, or one or more indicators of the results of the comparisons, or the protocol attributes to be used, are returned to the client or node **102**. In other embodiments, a protocol attribute to be used, or an indicator of a protocol attribute to be used, is extracted from, or derived from, a URL or equivalent identifying information used by a server, client, or node; is extracted from, or derived from, address information or a whole or partial name of a server, client, or node; is extracted from, or derived from a user name or identifier, class or type of user, group of users, or selection of users, optionally as associated with a service, server, client, or node; is extracted from, derived from, or associated with, an application or class or group of applications, a service or class or group of services, or a database or equivalent source of data or a class or group of databases or sources of data; is extracted from, or derived from, the identifier of a standard protocol, a standard protocol message (for example, a TCP SYN), the protocol-level content of a message, or protocol message headers or equivalent information; is derived from all or a part of the time, all or a part of the date, or all or a part of the duration since a time or date; is determined according to a sequence or progression; and/or, is determined randomly or pseudo-randomly.

After one or more comparisons are made, or after the extraction or derivation of a value, or after an association or determination is made, the server, client, or node will have one or more values it can use as protocol attributes or as indicators of which protocol attributes to use; or, if there has not been a determination of a condition resulting in one or more protocol attribute values to use, then in the primary embodiment the server **206** will use the standard values for the protocol attributes as configured for the server **206**, and in other embodiments, the server, client, or node will use the standard values as configured for it for some or all of the protocol attributes; the server, client, or node will use another default value for some or all of the attributes; the server, client, or node will use the last-used values for some or all of the attributes; or, the server, client, or node will use randomly, pseudo-randomly, or arbitrarily determined values for some or all of the attributes.

In the primary embodiment, the server **206** is a cache server, typically operating in a group of cache servers, and

groups of cache servers are distributed at numerous points on the global Internet. The server **206** includes a cache function **208** coupled to a content cache **210** to provide caching of content for the server **206**. In another embodiment, the server **206** is a cache server operating individually, or independently from other cache servers. In other embodiments, the server **206** is a content server that provides content; a content server operating in a group of content servers; a content server operating in a group of content servers distributed at numerous points on the global Internet; an application server that supports one or more applications; an application server operating in a group of application servers; an application server operating in a group of application servers distributed at numerous points on the global Internet; a service server that provides one or more services; a service server operating in a group of service servers; a service server operating in a group of service servers distributed at numerous points on the global Internet; a database server that provides data; a database server operating in a group of database servers; a database server operating in a group of database servers distributed at numerous points on the global Internet; a server operating in a heterogeneous group of servers; or, a server operating in a heterogeneous group of servers distributed at numerous points on the global Internet.

In the primary embodiment, each server **206** has an identical copy of a table **220** containing partial or whole URLs to which the alphanumeric URL string provided by the client **102** in its information request will be matched if possible, along with an indicator of the protocol attribute values to be used when that match occurs or the actual protocol attribute values themselves to be used when that match occurs. The table **220** can be modified from time to time, including adding new entries, changing the contents of existing entries, and deleting entries. Any time that a new version of the table **220** is created, copies of that version are distributed to servers **206** in the global group of cache servers. Optionally, the table **220** can have an expiration date and time, after which the server **206** will stop using the table **220** if it is out-of-date and has not been replaced with an updated table. In the primary embodiment, the conditional protocol control information is combined with other information distributed to servers **206** so as to minimize to the extent practicable the number of tables **220** that are distributed to, and synchronized among, the global group of servers; in another embodiment, this combination and minimization is not implemented; and this combination and minimization is not done in most or all other embodiments.

In other embodiments, the conditional protocol information may be in the form of a table **220** as in the primary embodiment, or may be in the form of multiple tables, or may be in the form of one or more data files, one or more databases, one or more calculated or derived elements, one or more calculated or derived groups of elements, one or more individual equations or formulas, one or more groups of equations or formulas, one or more individual expressions, one or more groups of expressions, one or more individual programs or routines, one or more groups of programs or routines, one or more individual listings, one or more groups of listings, and/or other forms equivalent to any of these.

In other embodiments, the conditional protocol control information contains partial or whole URLs to which the alphanumeric URL string provided by the client **102** in its information request will be matched if possible, along with an indicator of the protocol attribute values to be used when that match occurs or the actual protocol attribute values themselves to be used when that match occurs, and is the same for each server **206** at a location, but varies from one location to another; is the same for each server in a region, but varies from one region to another; or, is different at each server. In other embodiments the conditional protocol control information contains partial or whole URLs to which the alphanumeric URL string used by a client or node **102** will be matched if possible, along with an indicator of the protocol attribute values to be used when that match occurs or the actual protocol attribute values themselves to be used when that match occurs, and the conditional protocol control information is identical at each client or node **102**; is the same for each client or node **102** in a group, but varies from one group of clients or nodes **102** to another; is the same for each client or node **102** in a region, but varies from one region to another; or, varies from one client or node **102** to another client or node **102**.

In other embodiments, the conditional protocol control information contains other comparison information that will be matched, if possible, along with a corresponding indicator of the protocol attribute values to be used when that match occurs or the corresponding actual protocol attribute values themselves to be used when that match occurs, and each server, client, or node has an identical copy of the conditional protocol control information; the conditional protocol control information is the same for each server, client, or node at a location, but varies from one location to another; the conditional protocol control information is the same for each server, client, or node in a region, but varies from one region to another; the conditional protocol control information is the same for each server, client, or node in a group, but varies from one group to another; or, the conditional protocol control information varies from one server, client, or node to another. In these embodiments, the comparison information can include subsets of the alphanumeric URL string; identifying information equivalent to a URL; one or more IP addresses; network information associated with one or more IP addresses; network interconnection characteristics associated with one or more IP addresses; the geographic location, or logical or physical network location, of a server, client, or node; a value of a performance measurement or threshold; a rate or amount of variation in a performance measurement or threshold; a rate or amount of resource utilization; a rate or amount of variation in resource utilization; a threshold of resource utilization; addresses, names, classes, types, or group identifiers of servers, clients, nodes, users, applications, services, databases, or other data sources; protocol identifiers, protocol message identifiers or attributes, or protocol message headers or attributes; times, dates, or intervals; and/or, sequences, ratios, progressions, equations, randomly generated data, or arbitrarily determined data.

In embodiments where the conditional protocol control information is not identical at all servers, clients, or nodes, optionally a system can create conditional protocol control information for each location, region, server, client, or node, and optionally can manage the distribution of the conditional protocol control information to each location, region, server, client, or node. For partial or whole URLs, subsets of a URL alphanumeric string, or other comparison information that will be compared, a conditional protocol information management and distribution system (CPIMDS) optionally generates, or stores, or generates and stores, protocol attributes for any or all of a server, all servers, a subset of servers, a location, all locations, a subset of locations, a region, all regions, a subset of regions, a group, all groups, a subset of groups, an application, all applications, a subset of applications, a service, all services, a subset of services, a database or data source, all databases or data sources, a subset of databases or data sources, a protocol, all protocols, a subset of

protocols, a client, all clients, a subset of clients, a node, all nodes, a subset of nodes, or a combination of some, a subset of, a subset of some of, or all of, servers, locations, regions, groups, applications, services, databases or data sources, protocols, clients, or nodes; and optionally generates, or stores, or generates and stores, default protocol attributes to be used in the absence of a matching comparison for any or all of a server, all servers, a subset of servers, a location, all locations, a subset of locations, a region, all regions, a subset of regions, a group, all groups, a subset of groups, an application, all applications, a subset of applications, a service, all services, a subset of services, a database or data source, all databases or data sources, a subset of databases or data sources, a protocol, all protocols, a subset of protocols, a client, all clients, a subset of clients, a node, all nodes, a subset of nodes, or a combination of some, a subset of, a subset of some of, or all of, servers, locations, regions, groups, applications, services, databases or data sources, protocols, clients, or nodes. The CPIMDS also optionally generates, or stores, or generates and stores, one or more rules for determining which of the server, location, region, group, application, service, database or data source, protocol, client, node, or combined values to use when determining a comparison to be included in the conditional protocol control information for a server, all servers, a subset of servers, a location, all locations, a subset of locations, a region, all regions, a subset of regions, a group, all groups, a subset of groups, an application, all applications, a subset of applications, a service, all services, a subset of services, a database or data source, all databases or data sources, a subset of databases or data sources, a protocol, all protocols, a subset of protocols, a client, all clients, a subset of clients, a node, all nodes, a subset of nodes, or a combination of some, a subset of, a subset of some of, or all of, servers, locations, regions, groups, applications, services, databases or data sources, protocols, clients, or nodes; optionally generates, or stores, or generates and stores, one or more rules for determining when to create, or distribute, or create and distribute, conditional protocol control information for or to a server, all servers, a subset of servers, a location, all locations, a subset of locations, a region, all regions, a subset of regions, a group, all groups, a subset of groups, a client, all clients, a subset of clients, a node, all nodes, a subset of nodes, or a combination of some or all of servers, locations, regions, groups, clients, nodes, or a subset of servers, locations, regions, groups, clients, nodes, or for or to all servers, clients, or nodes; optionally generates, or stores, or generates and stores, one or more rules for determining whether, and if so when, any conditional protocol control information shall expire; optionally includes an expiration value with some or all of the conditional protocol control information; and, optionally distributes conditional protocol control information to a server, all servers, a subset of servers, a location, all locations, a subset of locations, a region, all regions, a subset of regions, a group, all groups, a subset of groups, a client, all clients, a subset of clients, a node, all nodes, a subset of nodes, or a combination of some or all of servers, locations, regions, groups, clients, nodes, or a subset of servers, locations, regions, groups, clients, nodes, or for or to all servers, clients, or nodes.

In embodiments where the conditional protocol control information is maintained separately from each server, client, or node, the conditional protocol control information can optionally include, or can optionally correlate to, an indication of for which servers, locations, regions, groups, applications, services, databases or data sources, protocols, clients, nodes, or supersets, subsets, or combinations thereof, the conditional protocol control information can be used in adapting protocol attributes. The conditional protocol control information can be maintained at a single location or multiple locations; if maintained at multiple locations, each location can maintain a complete copy of all conditional protocol control information, or each location can maintain a copy of a subset of the conditional protocol control information. Optionally, a conditional protocol information management and distribution system can create conditional protocol control information for, and optionally distribute conditional protocol control information to, each conditional protocol control information location, groups of conditional protocol control information locations, or all conditional protocol control information locations. If conditional protocol control information locations store a subset of the conditional protocol control information, participating servers, clients, or nodes can optionally use a table, ordered table, shuffled table, directory, logical name translation system, or information obtained from, or received from, another server, client, node, or controller, to determine which conditional protocol control information location to receive conditional protocol control information from.

Referring next to FIG. **3**, an embodiment of a process **300** for modification of the TCP protocol for various connections to a server **206**. In summary, in the primary embodiment two nodes on the Internet communicate using TCP, one node being a client **102** that requests information in block **304**, such as web page content, and the second node being a cache server **206** that provides information in response to a request. The cache server **206** operates as one of a group of cache servers, and groups of cache servers are distributed at numerous points on the global Internet. TCP connections are established in order to use HTTP to communicate information requests from clients **102** to servers **206** in block **304** and responses from servers **206** to clients **102**. HTTP utilizes URLs; each URL begins with the scheme "http", which is followed by a host field containing the IP address or name of the host where the requested information can be found, followed by a path, which will be used at the server **206** to locate the requested object or information, optionally followed by a query string. Thus, the full URL typically is an alphanumeric string containing the scheme, host field, path, and any optional query string, each part of which is separated from the other parts by special characters such as ":", "/", and "?". Thus, an example URL referencing content that can be served by the cache server **206** in the primary embodiment might look like:

http://customer1.webserving.com/folderB/directory/logo.gif

optionally followed by "?" and a query string. A client **102** requesting this object would send an HTTP message using an HTTP method called "GET" to the server **206** identified by the hostname "customer1.webserving.com" in block **304**. The server identification is accomplished through a DNS translation of the hostname into one or more IP addresses. The Domain Name System (DNS) is the name translation system used in the Internet. The HTTP GET message sent to the server **206** typically includes part or all of the URL that the client **102** is requesting. In block **308**, some or all of the URL is compared with entries in the table **220** to determine the attributes to use for the connection as described further below.

In the primary embodiment, in block **312** the server **206** conditionally adapts the attributes of the TCP protocol for each TCP connection established by a client **102**. Conditionally adapting the attributes of the TCP protocol does not require changes to standard TCP protocol implementations at every node, i.e., any client **102** that supports standard TCP (which virtually all Internet and all World Wide Web clients do) can communicate with the server **206** to get the object via

a conditionally adapted TCP connection, but may require a modification to the server's TCP protocol handler to allow changes to be made to the TCP protocol attributes by another program, or on a per-connection basis, or both. Where the TCP protocol handler provides a facility for programmatically making these changes, the content serving application may require an interface enabling it to communicate conditionally adapted TCP protocol attributes to the TCP protocol handler. In the primary embodiment, a set sockets statement can be used to communicate conditionally adapted TCP protocol attributes from the content serving application to the TCP handler **214**, which can be a modified TCP software stack that accepts and implements changes to the TCP protocol attributes on a per-connection or per-request basis. In various embodiments, the TCP handler could be implemented in software or hardware or both, and can be implemented as part of a node or separately from the node, for example in a switch or other device that provides protocol services.

Conditionally adapting the protocol for each connection results in the server **206** concurrently using the TCP protocol for multiple unrelated connections, wherein the TCP protocol attributes vary, at least initially and sometimes persistently, from one connection to another. At block **316**, the requested object or information is sent from the server **206** to the client **102**. When a requested object or information is delivered without any change to the TCP protocol attributes, the standard TCP protocol attributes pre-configured for the TCP protocol handler are used; but when the requested object or information is delivered with conditionally adapted TCP protocol attributes, the conditionally determined TCP protocol attributes are used instead of the standard pre-configured TCP attributes.

A typical caching content server **206** in actual operating use in a web hosting service provider environment may typically service 2,000 to 3,000 requests per second, across 100 to 200 (or more) customers. If half of these requests will use changed TCP protocol attributes and half will not, then on average, half of the connections managed by the server **206** at any one point in time would use the changed TCP protocol attributes and half would use the standard TCP protocol attributes as pre-configured in the TCP handler **214**. Additionally, if there are five separate (different) changed TCP protocol attribute sets across which those requests using changed TCP protocol attributes are evenly distributed, then on average at any one point in time a server **206** would concurrently be operating TCP connections with six different sets of TCP protocol attributes: 50% of the connections would utilize standard pre-configured TCP protocol attributes, and 10% would utilize each of the five possible changed TCP protocol attribute sets.

In the primary embodiment, the server **206** bases the conditional adaptation of the attributes of the TCP protocol on the alphanumeric URL string provided by the client **102** in the HTTP GET message. In the primary embodiment, the conditional protocol control information is in the form of a table **220** containing partial and/or whole URLs for comparison, the table **220** is stored on the server **206**, and the table **220** is generally kept reconciled on all servers **206**. The server **206** compares the alphanumeric URL string provided by the client **102** in the GET request to the table **220** and identifies the most specific match from left to right that it can find in the table **220** in block **308**. Taking our earlier example and assuming a simplified table, the sole Table shows mappings from whole and/or partial URLs into TCP attribute sets comprising specific protocols attributes (identified as "attr1", "attr2", "attr3", etc.) to be used and the appropriate value or setting for that use of that attribute.

Table Mapping to TCP attribute sets

Partial URL	TCP Attribute Set
http://customer1.webserving.com/folderA/	attr1 = yes, attr3 = 25, attr4 = low
http://customer1.webserving.com/folderB/fastnet.com	attr1 = no, attr2 = 1, attr4 = high
http://customer2.webserving.com/	attr3 = 50, attr6 = fast
http://customer3.webserving.com/	attr1 = yes, attr3 = 25, attr4 = low

The URL in the example above, "http://customer1.webserving.com/folderB/directory/logo.gif," would be matched against the second line-entry in the table. The TCP protocol attribute set (group of TCP protocol attributes) to be used for the TCP connection that services, or responds to, this HTTP GET message from this client **102** would be "attr1=no, attr2=1, attr4=high" and the TCP protocol attributes for this TCP connection would be set accordingly. This simplified table example also illustrates that the scheme and host name may be sufficient for a matching entry, such as in the case of the entry "http://customer3.webserving.com," or the host name alone may be sufficient, such as in the case of the entry "fastnet.com." This simplified table example also illustrates that there may be a null entry in the table, as in the case of "http://customer2.webserving.com"; in this simplified table example, the null entry signifies use of the standard pre-configured TCP protocol attributes. Note that an actual table in the primary embodiment would contain more information than the simplified example table shown here, as discussed.

The table **220** can be modified from time to time, including adding new entries, changing the contents of existing entries, and deleting entries. When a new version of the table **220** is created, copies of the new version of the table **220** are distributed to the servers **206** in the global group of cache servers. Optionally, the table **220** can have an expiration date and time, after which the server **206** will stop using the table **220** if it is out-of-date and has not been replaced with an updated table **220**; under this condition, the server **206** would then use the standard TCP protocol attributes for connections until the server **206** received an updated or unexpired table **220**.

In the primary embodiment, the conditional protocol control information is combined with other information, such as customer billing codes and other customer-specific information, that is distributed to all servers **206**, in order to minimize the number of tables that are distributed to, and reconciled throughout, the global group of servers. This embodiment has the advantage of simplifying administration, change management, and rollback in the event that distribution of a new table **220** causes a problem or error condition, but the disadvantage that the conditional protocol adaptation is the same for any given entry in the table **220** at every server **206**, regardless of the server's location or other factors. In other embodiments, as discussed, the conditional protocol control information can vary from one server **206** to another.

Referring to FIG. **4**, an embodiment of a process for potentially modifying protocol attributes on a connection-by-connection basis is shown. The depicted portion of the process begins in block **416** where a uniform resource indicator (URI) is requested by the client **102**. The cache function **208** of the server **206** receives the request for the content object. The

URI is evaluated by the protocol attribute selector **212** to find a match to something in the table **220**. The table **220** is queried in block **424** for any attributes. Retrieved attributes are communicated to the TCP handler **214** in block **428**. The connection is established in block **432** according to the selected attributes to connect the end user system **102** with the server **206**. The content object is delivered in block **436**. This process is performed on each URI such that each connection or socket can be independently controlled, if desired. Indeed, two different end user systems **102** could request the same content object and it could be delivered in a very different manner with different selected protocol attributes for each user.

Referring to FIG. **5**, a block diagram of an embodiment of a content delivery system **500** is shown. Content delivery system **500** operates in a manner that is similar to the content delivery systems of FIGS. **1-2** and therefore the description of those systems is also applicable to the present embodiment. While continuing reference will be made to the preceding embodiments, in the interest of clarity, the discussion of common elements and functionality will not be repeated.

As illustrated, end user computers **102** access the global internet **104** through autonomous systems **232**. Autonomous systems **232** may include internet service providers which offer end users access to the global internet **104** over a private communication network. Different providers may offer different types of service and may serve different geographic areas. For example, autonomous system AS1 can represent a DSL communication network such as those operated by AT&T or Qwest Communications, or it could be a cable access network such as those operated by Cox Communications in the United States, or by Rogers Communications in Canada. Autonomous system AS2 could be a satellite communication network, a cellular network, a WiMAX (IEEE 802.16) network, Wi-Fi™ (IEEE 801.11) access, and the like. Depending upon the underlying communications technology, autonomous systems **232** can present different network characteristics that are relevant to the performance of a transport layer protocol such as TCP.

Content delivery servers **206** are also connected to the global internet **104** and can be connected to corresponding autonomous systems **232**. As shown, content server **206-1** is connected to autonomous system **232-1** by router **236-1** and content server **206-2** is connected to autonomous system **232-2** by router **236-2**. Routers **236** thus provide direct links L1, L2 between servers **206** and their corresponding autonomous systems **232**. In some embodiments, servers **206-1**, **206-2** can be edge servers that are collocated with the autonomous system network infrastructure and provide large bandwidth and fast response times for content distribution to end users in a particular location. In an exemplary embodiment, each server **206** is configured to handle approximately 2000 connections per second and can support a 10 Gbps link to its corresponding autonomous system. Of course, the number of servers, number of connections, and data rates may vary based on the location served, traffic patterns, hardware capabilities, and other factors.

Servers **206** also communicate with origin server **240**. Origin server **240** can act as a source of the content distributed to end users. For example, servers **206** may cache content received from origin server **240** and may use the cached content to fulfill end user requests. If requested content is not found in their respective caches, servers **206** can send a request for the missing content to origin server **240**. When requesting content, servers **206** can report information about the content request as well as the conditionally adapted protocol parameters to the origin server **240**. The origin server **240** can collect, compile, and distribute information to servers **206** for use in adapting connection parameters. For example, the origin server **240** can function as part of a CPIMDS and can distribute URL tables **220** and other protocol performance information to servers **206** as previously discussed.

FIG. **6** is a functional block diagram of content distribution server **206** according one embodiment of the present invention. As shown, server **206** includes a processor **244**, memory **248**, one or more network interfaces **252**, and a data store **220**. Although not shown, server **206** can also include a cache function **208** and a content cache **210** as previously described.

Network interfaces **252** can include a plurality of ports (P1, . . . , PN) for sending and receiving data over a connecting network. In some embodiments, at least one network interface is dedicated to providing a high-bandwidth link to an autonomous system and can be matched to its particular network characteristics. Additional ports and/or network interfaces can provide access to the global internet **104**, origin server **204**, or other parts of a content delivery system.

Processor **244** executes programmable instructions for managing the delivery of content to end user computers **102** and can include one or more general purpose microprocessors or application-specific hardware elements. As shown, processor **244** is coupled to memory **248** which can include any combination of volatile or non-volatile computer-readable storage such as random access memory (RAM), read only memory (ROM), magnetic disk storage, and the like. Memory **248** can provide a data store **220** which, as previously described, can be a table or other data structure including information for modifying transport layer performance parameters. Data store **220** is discussed further in connection with FIG. **8**.

Server **206** can include a number of data sources which respond to content requests from end users. In one embodiment, processor **244** supports a protocol stack that enables changes affecting the performance of the transport layer to be made from higher layers in the stack on a per-connection or per-request basis. This enables server **206** to receive a content request from an end user computer over an existing connection, gather information about the request, and intelligently modify the performance of the connection based on information from the request. In this way, for example, server **206** can modify TCP performance parameters based on known characteristics of the connecting network, the geographic location of the end user, metadata associated with the requested content, a service level of the content provider, link utilization, or any combination of these and other factors.

FIG. **7** illustrates a modified TCP protocol stack **700** such as can be included as part of content server **206** according to one embodiment of the present invention. As shown, layers **710-740** correspond roughly to layers of the standard OSI network model. At the lowest layers **710** in the protocol stack (physical and data link layers), data bits are received at the network interface hardware and assembled into data units for delivery to the next higher layer. Here, a network layer **720** includes an IP module that receives IP packets from the lower layers and determines an appropriate transport layer protocol based on their header information. With transport control protocol, network layer **720** sends TCP messages to the transport layer **730** which, in turn, passes application messages to data sources **750** in the application layer **740**.

In operation, an end user computer **102** can establish a TCP connection with content server **206**. The connection can be created using a collection of TCP parameters which are preconfigured at the server and do not necessarily reflect information about the end user or the way in which the end user computer is connected to the server. For example, referring

again to FIG. **5**, computer **102-1** can initiate a connection C2.1 with server **206-1** to start a TCP session. Once the connection is established, the end user can send a content request over the connection. As illustrated, a single end user computer **102** can establish multiple connections to a given server **206** and each connection can carry multiple content requests. Protocol stack **700** is configured such that TCP settings can be adjusted on a per-connection or even a per-request basis.

When a content request is received at server **206**, it is conveyed through protocol stack **700** to an appropriate data source **750** in the application layer **740**. For example, a web server **750-1** can respond to requests for web pages, a caching application **750-2** can respond to file requests, and an application server **750**-N can respond to requests for application services. Server **206** can include any number or arrangement of data sources **750** and each data source can respond to multiple content requests.

Each data source **750** can interact with a TCP handler **760** at the transport layer **730** to modify its connections. In one embodiment, TCP handler **760** enables the data sources to modify the timing at which packets are sent to the end user computer to be more or less aggressive based on information gathered from the content request. TCP handler **760** can also modify the pace at which packets are sent. Pacing can indicate that a burst of packets should be sent as soon as possible or that data transmission should be spread out over time. The maximum TCP send window ("send buffer size") for a connection can also be adjusted. For example, in some embodiments, TCP handler **760** can adjust the maximum send buffer to be a multiple of a standard size and can permit buffer utilization to increase until it reaches the maximum size. Alternatively, TCP handler **760** can vary the number of bytes allocated for a particular connection directly.

Generally speaking, each content request has at least two pieces of information. These include a source address of the end user computer and an identifier corresponding to the requested content. For example, a data source **750** that responds to the request for sample URL, http://customer1.webserving.com/folderB/directory/logo.gif, would know the IP address of the requesting computer (e.g., abc.def.ghi) as well as the file name of the requested content (logo.gif). From this starting point, server **206** can obtain additional information from data store **220** with which to modify the transport layer parameters of the TCP connection.

FIGS. **8**A-**8**C illustrate partial exemplary data elements **810-850** such as can be maintained in data store **220** and used for determining modified parameters for a TCP connection.

FIG. **8**A illustrates exemplary data corresponding to a requested content object. In particular, table **810** can represent a collection of metadata **810**. Metadata for each requested object can include a file name, file size, file type, and content provider as well as TCP attributes associated with the content object. For example, attr1 can represent pacing on the TCP connection. In some embodiments, pacing is disabled for small files and enabled for large files. This can permit content requests involving a large number of small files to be fulfilled with quick bursts and can facilitate a more even delivery of large content. TCP attributes can also correspond to the type of data such as whether the requested content is text or video information.

Table **820** includes information about content providers. Each content object can be associated with a content provider. The associated content provider can be identified in the file metadata or it can be determined from the URL of the content request. In some cases, content providers can select a service level for the distribution of their content on the content delivery system. Among other possibilities, requests for content from providers that choose a premium service level can be biased in favor of increased performance. For example, requests for provider ABC's content can be preferentially modified (e.g., attr6=fast) and, when appropriate, can be allocated a relatively larger send buffer (e.g., attr4=300,000). By contrast, requests for provider DEF's content at the standard service level can be assigned a smaller send buffer (attr4=100,000) and modified only on a best efforts basis (attr6=slow). Service level can also set on a per-request basis. For example, a customer can elect a high level of service by adding information to the request query string.

FIG. **8**B illustrates exemplary data corresponding to the network address of an end user computer. Table **830** provides an association between IP address, geographic location, and autonomous system number. A source IP address can be obtained from TCP header information and used to identify a geographic region of the end user computer. The geographic region can be a city, state, country, or continent and can provide a rough estimate of the distance or round trip time from the server **206** to the end user computer. In addition, geographic location can also be a rough indicator of service. For example, network communications in Asia may be characterized by a higher latency than network connections in Europe or some other location. These differences can be factored into the TCP attributes so that, for example, more aggressive timing parameters can be used with Asia-based IP addresses.

Table **830** can also store information about primary routes to particular locations. For example, a considerable amount of network traffic destined for South America passes through servers in Florida and other primary gateway locations. The attributes in table **830** can be biased to optimize TCP performance based on conditions at these gateway servers. For example, TCP timing parameters may be adjusted based on traffic statistics and load along a primary route such that transmit timing for South American connections is made more or less aggressive. Many other location-specific adjustments are possible within the scope of the present invention. In some embodiments, cost and path information can also be included. As an example, transit charges and other direct costs of providing service can be tracked as well as indirect or resource costs.

Table **840** provides information about the autonomous systems. Server **206** can determine an autonomous system (AS) number for an end user computer based on the source address of a content request. The AS number, in turn, can be used to obtain additional information for modifying TCP parameters. For example, if it is known that a particular AS is associated with a type of network, the characteristics of the network technology can be used to determine appropriate TCP parameters for a connection. Cable networks can have a relatively high bandwidth and may be less prone to saturation than DSL networks. Satellite connections, on the other hand, are typically associated with high latency. Server **206** can take advantage of these characteristics by matching timing and pacing parameters to the particular type of network.

When server **206** is collocated at an AS data center, link statistics can be maintained and used to determine TCP parameters. As a link nears full capacity, for example, it may be inappropriate to increase the timing or send buffer size of connections. In some embodiments, the preconfigured TCP parameters are used when link utilization exceeds a predetermined threshold. Thus, among other possibilities, the attributes associated with AS information can indicate whether or not TCP parameters should be modified and, if

modification is appropriate, which parameters are best suited for known characteristics of the AS network.

FIG. **8**C illustrates exemplary data corresponding to server utilization. In some embodiments, each content distribution server **206** has a limited amount of bandwidth and is intended to support a certain network load. As the server approaches its limits, it may be appropriate to scale back on resource allocation to TCP connections. Conversely, when the server experiences a light load, it may be appropriate to allocate more system resources to improving the performance of TCP connections. Table **850** provides information for judging load at a content distribution server, including a bandwidth allocation (BW) measures and a connection rate (CPS).

By way of illustration, assume that a particular server **206** can support up to 2,000 connections per second and has available bandwidth of 1 Gbps. When connection rates and bandwidth usage are low, more resources are available for modifying connections. In that case, it may be appropriate to use more aggressive timing, larger buffers and other performance enhancements. Thus, for example, connections to server E**1** may be modified by increasing the maximum send buffer size to 300,000 bytes and biasing towards aggressive timing and/or pacing utilization. On the other hand, server E**56** is nearing full capacity and may therefore bias new connections to the preconfigured TCP parameters.

As will be readily appreciated, many different combinations of factors can affect when and how a TCP connection is modified. Different weights and precedence can be assigned to the different types of information available from the data store **220**. For example, system resources may have the highest precedence, followed by service level, and then by metadata and AS factors, and finally by geographic considerations. Across categories, different weights may be assigned to the attributes so that a data source **750** can determine modified TCP parameters based on the net effect of some or all of the available information.

FIG. **9** shows exemplary performance profiles **900** such as can be utilized to modify TCP parameters according to embodiments of the present invention. Rather than determining parameters by combining individual factors, server **106** can include predetermined profiles for content requests. In the example, profile P**1** provides TCP settings for sending large files to nearby (low-latency) users. As illustrated, a performance increase can be realized by pacing such connections and allowing the TCP send buffer to grow very large. Relatively less aggressive timing adjustments are needed due to the low latency factor. Profile P**2**, on the other hand, represents large file transfers to a latent user. In that case, pacing is still used with the transfer, but more aggressive transmit timing may help to compensate for latency and an intermediate send buffer may be appropriate. Profile P**3** can be used to transfer small files. With small files, it may be desirable to disable pacing and transmit files in bursts. As a result, a large send buffer may not be needed.

FIG. **10** shows an embodiment of a process **1000** for modifying protocol attributes on a connection-by-connection or request-by-request basis. Process **1000** can be performed by a data source **750** or by the caching function **208** of a content distribution server **206**. At block **1010**, a content request R**1** is received over an existing connection C**1** at the server. The request can include the URI of a content object. The content object may be available in content cache **210**, or from origin server **240**, or from some other server accessible through the content delivery system.

The content request R**1** can be conveyed through the lower layers of protocol stack **700** to the appropriate data source **750** in the application layer **740**. Since server **206** is capable of modifying transport layer parameters on a connection-by-connection and even a request-by-request basis, the present process can be repeated for each new request (e.g., R**2**/C**1**) and/or each new connection (e.g., R**1**/C**2**) as determined by the data source **750** or caching function. Note also that that server **206** need not be dedicated to serving a particular type of content but can deliver files, images, video, or any other content available through the content delivery system.

After the request for content is received, the responding data source **750** determines whether the transport layer parameters used with the connection and/or request should be modified. Modifying the transport layer parameters is completely transparent to the end user; the end user is not required to install software or monitoring applications to receive a performance benefit.

At block **1015**, the server **206** makes an initial determination as to whether system load exceeds a predetermined threshold TH**1**. For example, the responding data source **750** can query the information in table **850** to determine current bandwidth usage and system load. If the system is experiencing a heavy load, pre-configured TCP parameters may be used. In that case, the process ends at block **1060**. On the other hand, if system load is below threshold TH**1**, a further determination is made as to link utilization. This can involve, for example, accessing the information in table **840**. If link utilization exceeds a predetermined threshold TH**2**, then the process can terminate at block **1060** and preconfigured TCP parameters can be used with the connection/request.

When there is sufficient system resources and link capacity, a determination can be made regarding the TCP parameters based on file size. The size of a requested file can be determined by accessing the metadata of table **810**. At block **1025**, the file size is compared to a threshold value TH**3** to determine if it is a "large" file. If the file is not a large file, then at block **1030** it is compared to another threshold TH**4** to determine if it is a "small" file. If the requested file does not fit in either category, then the preconfigured TCP settings may be used. Otherwise, for small files, pacing can be disabled and an appropriate send buffer size can be determined at block **1035**. Thereafter, at block **1065**, the responding data source or cache application directs the TCP handler to modify the connection for the small-file transfer.

With large files, it can be useful to make a further determination as to latency. At block **1045**, a round trip time (RTT) from the server to the end user computer is determined. This can be done by sending ICMP messages to the end user's address and measuring the response time. If RTT is less than a predetermined threshold TH**5**, then the connection may be characterized as low-latency. In that case, a relatively large send buffer size and less aggressive TCP timing may be appropriate. At block **1050**, these settings are determined by the data source or cache application either based on information from individual items in data store **220** or by selecting a performance profile. On the other hand, if RTT exceeds the threshold, the connection may be characterized as high-latency. At block **1055**, parameters for the large-file, high-latency transfer are determined. At block **1065**, the data source or cache function modifies the connection through the TCP handler for the large file transfer.

FIG. **11** shows an additional embodiment of a process **1100** for modifying protocol attributes. Process **1100** can be performed by a data source **750** or by the caching function of content distribution server **206**. The process begins at block **1110** when a request R**1** is received from an end user computer over connection C**1**. As previously noted, the process can be repeated for each new request (e.g., R**2**/C**1**) and/or

each new connection (e.g., R1/C2) as determined by the data source **750** or caching function.

At block **1115**, the IP address of the client is determined and the data source or cache function begins to gather information for modifying the connection. Initially, a geographic location and autonomous system of the end user computer are determined based on the IP address (blocks **1120-1125**). If the server has a dedicated link to the AS, link utilization is determined at block **1130** and compared to a predetermined threshold TH2. When the link capacity is below the threshold, the process terminates and the standard or pre-configured TCP parameters are used for the connection/request. If link utilization does not apply to the connection, or if link utilization is below threshold TH2, the process continues.

In this embodiment, a predetermined profile is selected based on the geographic location of the client and the type of connection. For locations in the United States served by cable access networks, blocks **1135-1140**, a first geographic performance profile G1 can be used. Relatively low latency may be assumed for US locations and this profile can adjust TCP timing to take advantage of the relatively high burst capability of cable networks. For locations in the United States served by digital subscriber line (DSL) networks, blocks **1145-1150**, a second geographic performance profile G2 can be used. This profile may use slightly less aggressive timing with a relatively large send buffer. Finally, for US locations served by satellite networks, blocks **1155-1160**, a third geographic profile G3 can be used. This profile may assume high latency but reliable delivery and therefore use relatively more aggressive TCP timing and an intermediate send buffer size.

Customized profiles can be used for non-US locations or when autonomous system information is not available as shown by blocks **1165-1170**. For example, a China-specific profile or an Asia-specific profile can be developed based on historical network performance measures. Similarly, where a primary route to a particular destination is known, profiles may be developed that are customized for the appropriate connecting network elements. When the appropriate geographic performance profile has been selected, the data store or cache function modifies the connection accordingly.

Throughout this document, the terms content delivery and content download are used and can mean either file download or streaming delivery. Additionally, a content object can be either a file or a stream. For example, the content object could be a voice call stream, a video stream, an image file, a music file, a live concert, an animation, an advertisement, a web page, a slide show, data file, hosted software, transactions, executable software or the like. Content can be static or dynamic, can pre-exist on the server, can be created on the server, or can be created or obtained elsewhere in response to a request from a client.

A number of variations and modifications of the disclosed embodiments can also be used. For example, some of the above embodiments discuss use of the TCP protocol or a transport-layer protocol. Other protocols could be modified on a connection-by-connection or request-by-request basis in other embodiments. Also, connection parameters can be modified based on additional information gathered from or associated with content requests such as HTTP request headers (e.g., content-length, cookies, content-type, user agent, etc.), transport layer security (e.g., HTTPS), layer 2 addressing (e.g., the MAC address of the router from which the request was received), port number, IP properties (e.g., TOS—terms of service), hostname, and whether or not a request successfully passed through a rewrite process.

While the principles of the disclosure have been described above in connection with specific apparatuses and methods, it is to be clearly understood that this description is made only by way of example and not as limitation on the scope of the disclosure.

What is claimed is:

1. A method for managing delivery of content in a system comprising a server and an end user computer, comprising:

establishing a first connection at the server for communicating with the end user computer;

receiving a request for content from the end user computer over the first connection, the request include a universal resource locator (URL);

determining one or more parameters relating to the performance of the first connection using information from the request, wherein the determined one or more parameters relate to utilization of available processing or memory capabilities of part or all of a system supporting the first connection;

determining one or more first values of attributes based on the URL and the one or more parameters;

modifying second values of attributes for the first connection at a transport layer to result in the determined one or more first values, the second values of the attributes for the first connection thereafter influencing utilization of the available processing or memory capabilities of the part or all of the system supporting the first connection;

changing, on a connection-specific basis, a connection protocol stack operator based upon the modified values of the attributes; and

sending the requested content from the server to the end user computer such that the transport layer manages delivery of the content in accordance with the modified second values of the attributes.

2. The method of claim **1**, wherein the one or more first values of attributes are further based on an estimated location of the end user computer.

3. The method of claim **1**, further comprising determining a latency characteristic of the first connection, wherein at least one of the second values of the attributes for the first connection is modified based on the latency characteristic.

4. The method of claim **1**, further comprising:

determining a connection type of the end user computer; and

determining a latency characteristic associated with the connection type, wherein at least one of the second values of the attributes for the first connection is modified based on the latency characteristic associated with the connection type of the end user computer.

5. The method of claim **1**, further comprising:

determining the size of the requested content; and

measuring a round trip travel time between the server and the end user computer when the data size exceeds a predetermined value, wherein at least one of the second values of the attributes for the first connection is modified based on the size of the requested content and the round trip travel time.

6. The method of claim **1**, further comprising determining an autonomous system from which the first connection is received, wherein at least one of the second values of the attributes for the first connection is modified based on network characteristics of the autonomous system.

7. The method of claim **1**, further comprising determining a link utilization between the server and an autonomous system of the end user computer, wherein at least one of the second values of the attributes for the first connection is modified based on the link utilization.

8. The method of claim **1**, further comprising determining a predetermined performance profile for the first connection

using the information from the request, wherein at least one of the second values of the attributes for the first connection is modified based on the predetermined performance profile.

9. The method of claim **1**, further comprising:

determining whether the requested content is available at the server;

obtaining the requested content from a second server when the requested content is not available at the server; and

caching the requested content at the server for at least a predetermined time.

10. The method of claim **1**, wherein modifying the second values of the attributes for the first connection thereafter adjusts a timing of data transmission at the transport layer in accordance with the one or more parameters.

11. The method of claim **1**, wherein modifying the second values of the attributes for the first connection thereafter adjusts a transport layer send window associated with the first connection.

12. The method of claim **1**, wherein modifying the second values of the attributes for the first connection thereafter adjusts a burst size of the first connection so as to pace data transmission according to the one or more parameters.

13. A content distribution server, comprising:

a network interface having a plurality of ports configured to send and receive data over a connecting network;

a processor coupled to the network interface and configured to manage a plurality of connections to end user computers;

a protocol handler configured to establish the plurality of connections with the end user computers according to predetermined transport layer parameters of the content distribution server and to manage data transmission over the plurality of connections; and

a data source configured to supply requested content to the end user computers over the plurality of connections, wherein the data source is configured to monitor a first connection for a request to:

determine one or more parameters for the first connection based on the request, the determined one or more transport layer parameters relating to utilization of available processing or memory capabilities of part or all of a system supporting the first connection;

determine one or more first values of attributes based on a URL and the one or more parameters, the request including the URL;

direct the protocol handler to modify second values of attributes for the first connection to result in the determined one or more first values, the second values of the attributes for the first connection thereafter influencing utilization of the available processing or memory capabilities of the part or all of the system supporting the first connection; and

change a connection protocol stack operator based upon the modified second values of the attributes.

14. The content distribution server of claim **13**, wherein the one or more first values of attributes are further based on an estimated location of the end user computer.

15. The content distribution server of claim **13**, further comprising a protocol attribute information store having information relating data associated with a content request to one or more connection parameters, wherein the data source determines at least one of the first values based on information retrieved from the protocol attribute information store.

16. The content distribution server of claim **13**, wherein the data source determines a geographic region corresponding to a destination address of the first connection, and wherein at least one of the one or more first values is further based on the geographic region.

17. The content distribution server of claim **13**, wherein the data source determines an autonomous system associated with the first connection, and wherein at least one of the one or more first values is further based on network characteristics of the autonomous system.

18. The content distribution server of claim **13**, wherein the data source determines a latency characteristic associated with the first connection based on the information from the request and directs the protocol attribute selector to modify at least one of the second values of the attributes based on the latency characteristic.

19. The content distribution server of claim **13**, wherein the protocol handler is configured to adjust a timing of data transmission at a transport layer based on the one or more first values.

20. A computer program product comprising a non-transitory computer-readable medium encoded with one or more sequences of one or more instructions which, when executed by a processor, cause a computer to:

establishing a first connection at the server for communicating with an end user computer;

receiving a request for content from the end user computer over the first connection, the request include a universal resource locator (URL);

determining one or more parameters relating to the performance of the first connection using information from the request, wherein the determined one or more parameters relate to utilization of available processing or memory capabilities of part or all of a system supporting the first connection;

determining one or more first values of attributes based on the URL and the one or more parameters;

modifying second values of attributes for the first connection at a transport layer to result in the determined one or more first values, the second values of the attributes for the first connection thereafter influencing utilization of the available processing or memory capabilities of the part or all of the system supporting the first connection;

changing, on a connection-specific basis, a connection protocol stack operator based upon the modified values of the attributes; and

sending the requested content from the server to the end user computer such that the transport layer manages delivery of the content in accordance with the modified second values of the attributes.

* * * * *

Exhibit C

US008856263B2

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

(10) **Patent No.: US 8,856,263 B2**
(45) **Date of Patent: Oct. 7, 2014**

(54) **SYSTEMS AND METHODS THERETO FOR ACCELERATION OF WEB PAGES ACCESS USING NEXT PAGE OPTIMIZATION, CACHING AND PRE-FETCHING TECHNIQUES**

(71) Applicant: **Limelight Networks, Inc.**, Tempe, AZ (US)

(72) Inventors: **Leonid Fainberg**, Tel Aviv (IL); **Ofir Ehrlich**, Tel Aviv (IL); **Gil Shai**, Tel Aviv (IL); **Ofer Gadish**, Rishon LeZion (IL); **Amitay Dobo**, Tel Aviv (IL); **Ori Berger**, Tel Aviv (IL)

(73) Assignee: **Limelight Networks, Inc.**, Tempe, AZ (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **14/137,598**

(22) Filed: **Dec. 20, 2013**

(65) **Prior Publication Data**

US 2014/0237066 A1 Aug. 21, 2014

Related U.S. Application Data

(60) Continuation of application No. 13/731,438, filed on Dec. 31, 2012, now Pat. No. 8,661,090, which is a continuation of application No. 13/471,211, filed on May 14, 2012, now Pat. No. 8,346,885, which is a division of application No. 12/848,611, filed on Aug. 2, 2010, now Pat. No. 8,321,533.

(60) Provisional application No. 61/213,959, filed on Aug. 3, 2009, provisional application No. 61/308,951, filed on Feb. 28, 2010.

(51) **Int. Cl.**
G06F 15/167 (2006.01)
H04L 29/08 (2006.01)

(52) **U.S. Cl.**
CPC ***H04L 67/2847*** (2013.01); ***H04L 67/02*** (2013.01)
USPC **709/213**; 709/219; 709/203; 715/827

(58) **Field of Classification Search**
USPC 709/200, 203, 213, 219; 715/827
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,182,133	B1 *	1/2001	Horvitz	709/223
7,363,291	B1 *	4/2008	Page	707/706
7,689,663	B2 *	3/2010	Kinnan et al.	709/217
8,156,419	B2 *	4/2012	Choudhary et al.	715/209
2003/0110296	A1 *	6/2003	Kirsch et al.	709/246
2004/0088375	A1 *	5/2004	Sethi et al.	709/218
2004/0205149	A1 *	10/2004	Dillon et al.	709/217

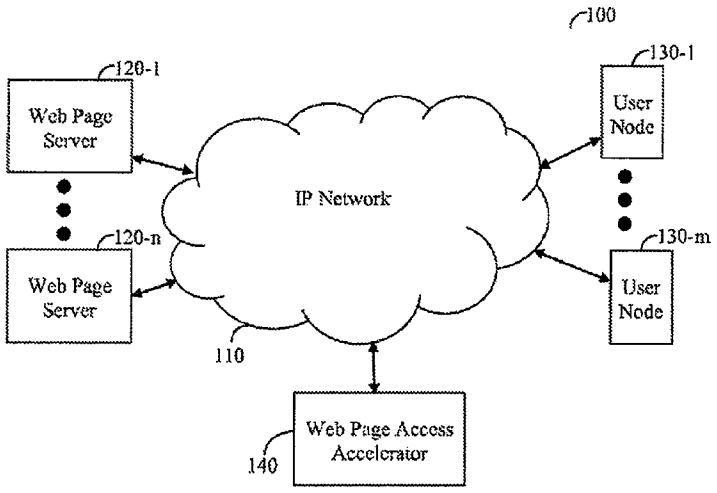
(Continued)

Primary Examiner — El Hadji Sall
(74) *Attorney, Agent, or Firm* — Kilpatrick Townsend & Stockton LLP

(57) **ABSTRACT**

A method and system for acceleration of access to a web page using next page optimization, caching and pre-fetching techniques. The method comprises receiving a web page responsive to a request by a user; analyzing the received web page for possible acceleration improvements of the web page access; generating a modified web page of the received web page using at least one of a plurality of pre-fetching techniques; providing the modified web page to the user, wherein the user experiences an accelerated access to the modified web page resulting from execution of the at least one of a plurality of pre-fetching techniques; and storing the modified web page for use responsive to future user requests.

20 Claims, 5 Drawing Sheets



(56) **References Cited**

U.S. PATENT DOCUMENTS

2008/0195712 A1* 8/2008 Lin et al. 709/206
2008/0228772 A1* 9/2008 Plamondon 707/10
2010/0017696 A1* 1/2010 Choudhary et al. 715/205
2011/0087966 A1* 4/2011 Leviathan 715/745

* cited by examiner

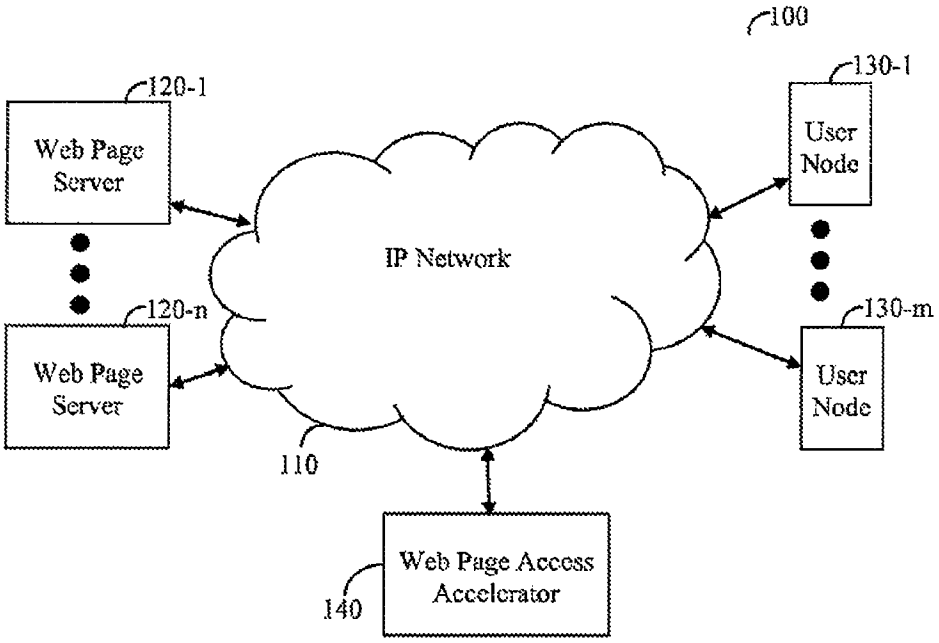

100
120-1
Web Page Server
120-n
Web Page Server
IP Network
110
130-1
User Node
130-m
User Node
140
Web Page Access Accelerator

FIG. 1

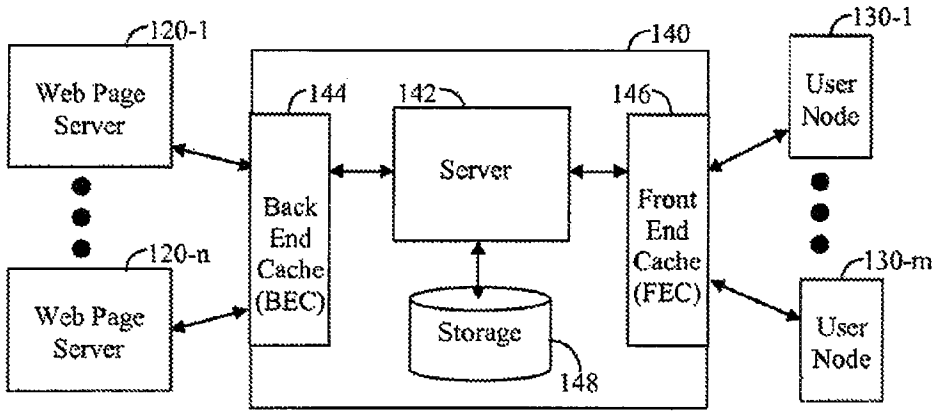

120-1
Web Page Server
120-n
Web Page Server
140
144
Back End Cache (BEC)
142
Server
146
Front End Cache (FEC)
Storage
148
130-1
User Node
130-m
User Node

FIG. 2

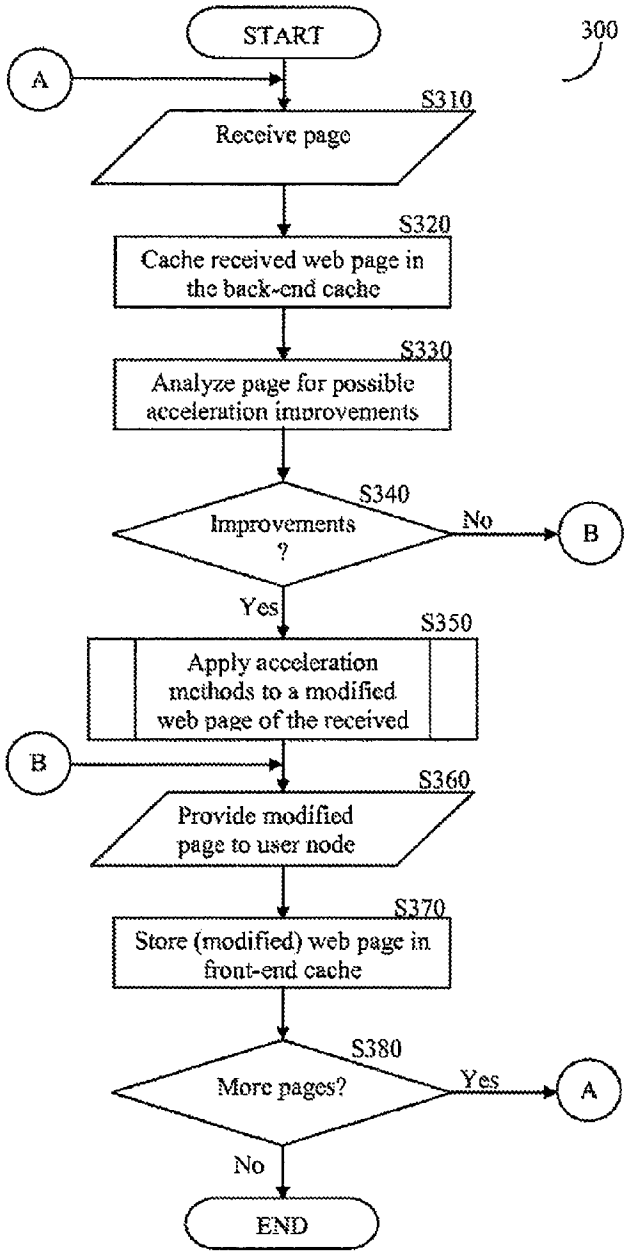

300
START
A
S310
Receive page
S320
Cache received web page in the back-end cache
S330
Analyze page for possible acceleration improvements
S340
Improvements ?
No
B
Yes
S350
Apply acceleration methods to a modified web page of the received
B
S360
Provide modified page to user node
S370
Store (modified) web page in front-end cache
S380
More pages?
Yes
A
No
END

FIG. 3

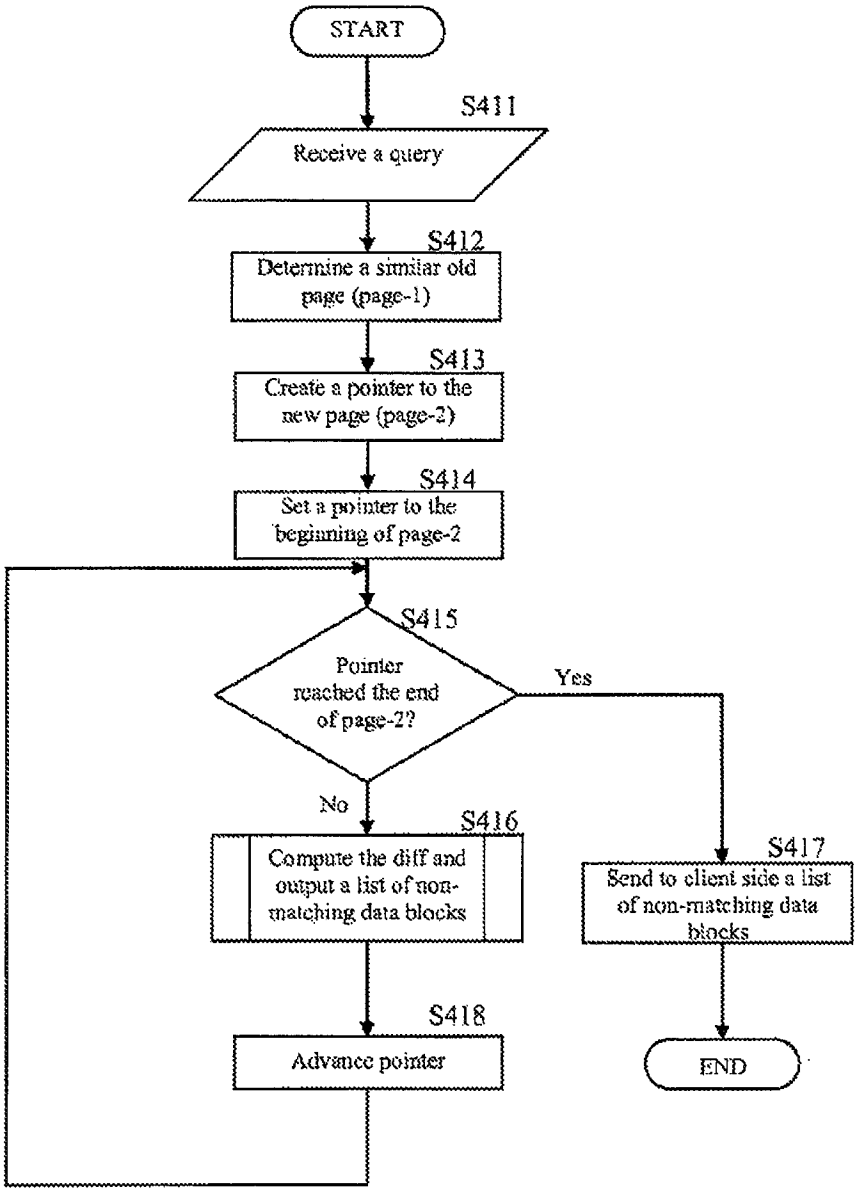

START
S411
Receive a query
S412
Determine a similar old page (page-1)
S413
Create a pointer to the new page (page-2)
S414
Set a pointer to the beginning of page-2
S415
Pointer reached the end of page-2?
Yes
No
S416
Compute the diff and output a list of non-matching data blocks
S417
Send to client side a list of non-matching data blocks
S418
Advance pointer
END

FIG. 4A

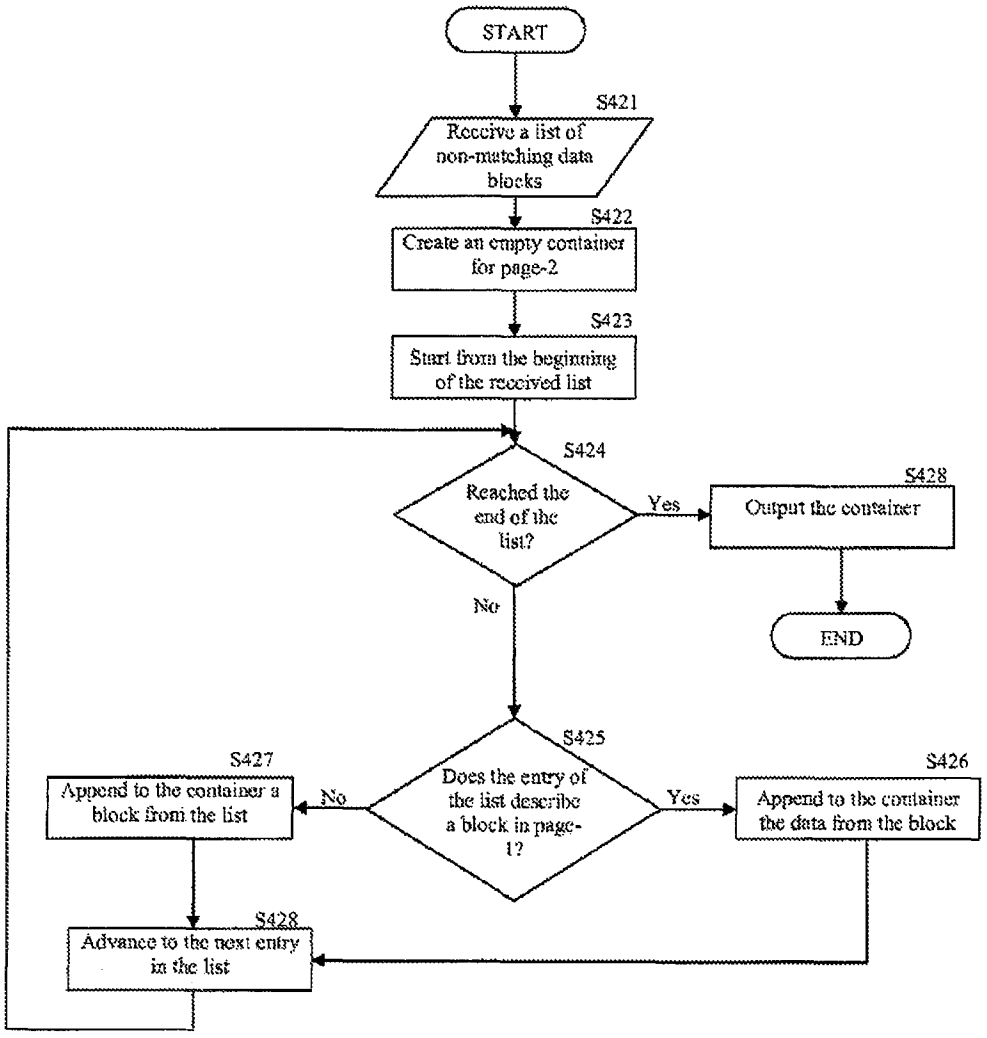

START
S421
Receive a list of non-matching data blocks
S422
Create an empty container for page-2
S423
Start from the beginning of the received list
S424
Reached the end of the list?
Yes
S428
Output the container
END
No
S425
Does the entry of the list describe a block in page-1?
No
S427
Append to the container a block from the list
Yes
S426
Append to the container the data from the block
S428
Advance to the next entry in the list

FIG. 4B

Block Index	Block type	Block Content
0	New Data	yx

FIG. 5A

Block Index	Block type	Start Position	End Position
1	Old Data	1	3

FIG. 5B

Block Index	Block type	Start Position	End Position
2	Old Data	5	7

FIG. 5C

SYSTEMS AND METHODS THERETO FOR ACCELERATION OF WEB PAGES ACCESS USING NEXT PAGE OPTIMIZATION, CACHING AND PRE-FETCHING TECHNIQUES

CROSS-REFERENCES TO RELATED APPLICATIONS

This application is a continuation of U.S. patent application Ser. No. 13/731,438, filed Dec. 31, 2012, entitled "SYSTEMS AND METHODS THERETO FOR ACCELERATION OF WEB PAGES ACCESS USING NEXT PAGE OPTIMIZATION, CACHING AND PRE-FETCHING TECHNIQUES," which is a continuation of U.S. patent application Ser. No. 13/471,211, filed May 14, 2012, entitled "SYSTEMS AND METHODS THERETO FOR ACCELERATION OF WEB PAGES ACCESS USING NEXT PAGE OPTIMIZATION, CACHING AND PRE-FETCHING TECHNIQUES," which is a divisional application of U.S. patent application Ser. No. 12/848,611, filed Aug. 2, 2010, which is a non-provisional application claiming the benefit and priority under 35 U.S.C. §119(3) of U.S. Provisional Patent Application 61/213,959, filed Aug. 3, 2009, and U.S. Provisional Patent Application 61/308,951, filed Feb. 28, 2010. The entire disclosure of each of the above-listed applications is incorporated herein by reference for all purposes.

FIELD OF THE INVENTION

The present invention relates generally to accesses to web pages, and more specifically to the acceleration of access to such web pages from the user's experience perspective.

BACKGROUND OF THE INVENTION

The traffic over the world-wide-web (WWW) using the Internet is growing rapidly as well as the complexity and size of the information moved from sources of information to users of such information. Bottlenecks in the movement of data from web servers of the content suppliers to the users, delays the passing of information and decreases the quality of the user's experience. Traffic is still expected to increase faster than the ability to resolve data transfers over the Internet.

Prior art suggests a variety of ways in an attempt to accelerate web page content delivery from a supplier of the content to the users. However, there are various deficiencies in the prior art still waiting to be overcome. It would be therefore advantageous to overcome these limitations, as it would result in a better user experience and reduction of traffic load throughout the WWW. It would be further advantageous that such solutions be applicable with at least all popular web browsers and/or require neither a plug-in nor a specific browser configuration.

BRIEF SUMMARY OF THE INVENTION

Certain embodiments of the invention include a system for acceleration of access to web pages. The system comprises a network interface enabling communication of one or more user nodes with one or more web servers over a network for accessing web pages stored in the one or more web servers; an acceleration server coupled to the network interface for accelerating access to the web pages to the one or more user nodes using at least one pre-fetching technique; a first cache connected to the acceleration server and the one or more user nodes and operative to cache information associated with requests directed from the one or more the user nodes to the acceleration server; a second cache connected to the acceleration server and the one or more web servers and operative to cache information associated with requests directed from the one or more web servers to the acceleration server; and a memory coupled to the acceleration server and containing a plurality of instructions respective of the at least one pre-fetching technique.

Certain embodiments of the invention further include a method for acceleration of access to a web page. The method comprises receiving a web page responsive to a request by a user; analyzing the received web page for possible acceleration improvements of the web page access; generating a modified web page of the received web page using at least one of a plurality of pre-fetching techniques; providing the modified web page to the user, wherein the user experiences an accelerated access to the modified web page resulting from execution of the at least one of a plurality of pre-fetching techniques; and storing the modified web page for use responsive to future user requests.

Certain embodiments of the invention also include a method for acceleration of access to a web page. The method comprises receiving a request to access a web page; generating a query that includes at least a URL of the requested web site and one more URLs of web pages similar to the requested web page; generating a list of non-matching data blocks between the requested web page and at least one of the similar web pages; and generating a web page respective of the requested web page by combining common data blocks with non-matching data blocks, wherein the common blocks are retrieved from the at least one similar web page and the non-matching blocks are retrieved from the requested web page.

BRIEF DESCRIPTION OF THE DRAWINGS

The subject matter that is regarded as the invention is particularly pointed out and distinctly claimed in the claims at the conclusion of the specification. The foregoing and other objects, features, and advantages of the invention will be apparent from the following detailed description taken in conjunction with the accompanying drawings.

FIG. **1** is a schematic block diagram of a system for acceleration of web pages access;

FIG. **2** is a schematic diagram of the data flow in a system for acceleration of web pages access;

FIG. **3** is a flowchart of the processing performed for the purpose of generating web pages that accelerate access;

FIGS. **4**A and **4**B are flowcharts illustrating the operation one of the perfecting acceleration technique in accordance with an embodiment of the invention; and

FIGS. **5**A, **5**B, and **5**C illustrate an exemplary data structure created by the technique shown in FIGS. **4**A and **4**B.

DETAILED DESCRIPTION OF THE INVENTION

The embodiments disclosed by the invention are only examples of the many possible advantageous uses and implementations of the innovative teachings presented herein. In general, statements made in the specification of the present application do not necessarily limit any of the various claimed inventions. Moreover, some statements may apply to some inventive features but not to others. In general, unless otherwise indicated, singular elements may be in plural and vice versa with no loss of generality. In the drawings, like numerals refer to like parts through several views.

In an exemplary but non-limiting embodiment of the invention, a web access acceleration system is placed in the path between the user nodes and the web servers and is responsible for integrating the acceleration mechanisms to the web pages selected for acceleration.

FIG. **1** depicts an exemplary and non-limiting schematic block diagram of a system **100** for acceleration of web pages access in accordance with an embodiment of the invention. A network **110** is connected to one or more web page servers **120**, each providing content typically using formatted documents using, for example, the hypertext markup language (HTML). The network may be a local area network (LAN), a wide area network (WAN), a metro area network (MAN), the Internet, the world-wide-web (WWW), the like, and any combination thereof. One or more user nodes **130** that are viewers of such web pages content are also connected to the network **110**. A user of a user node **130** typically browses the content using a web browser that is enabled to display the web pages. By using, for example but not by way of limitation, a uniform resource locator (URL) the browser is capable of accessing a desired web page.

The network **110** is also connected a web page access accelerator (WPAA) **140**. In accordance with the invention instead of providing web page content directly from a web page server, for example, a web page server **120-1**, to a user node, for example, a user node **130-1**, traffic is directed through the WPAA **140**, when applicable, i.e., when configured for accelerated access. Accordingly, a request for web page content is directed through WPAA **140** that is equipped with various acceleration mechanisms as further detailed herein below. In one embodiment of the disclosed invention, the web servers **120** are part of a server farm (not shown). In a further embodiment thereof, the WPAA **140** is provided as part of the server farm. In yet another embodiment of the invention, the WPAA **140** is integrated as an integral part of a web page server **120**.

FIG. **2** shows an exemplary and non-limiting schematic diagram of the data flow in a system for acceleration of web pages access in accordance with an embodiment of the invention is shown. In addition, the details of the structure of the WPAA **140** are also shown. For simplicity reasons the network interface is removed, however, a network type interface is the typical way for such components to communicate with each other. The WPAA **140** comprises of an acceleration server **142** that is coupled to storage **148**. The storage **148** typically holds instructions for the execution of one or more acceleration techniques, described herein below in more detail, that result in accelerating the transfer of web pages content to a user wishing to access such content. Under the control of the acceleration server **142**, there is a back-end cache (BEC) **144**, connected to the acceleration server **142** and to the one or more web page servers **120-1** through **120-*n***. The BEC **144** handles requests directed from the acceleration server **142** to the one or more web page servers **120**. By caching information in BEC **144**, overall access to web page content is accelerated. Under the control of acceleration server **142** there is a front-end cache (FEC) **146**, connected to the acceleration server **142** and to the one or more user nodes **130-1** through **130-*m***. The FEC **146** handles requests directed from the one or more user nodes **130** to the acceleration server **142**. By caching information in FEC **146**, the overall access to web page content is further accelerated.

FIG. **3** shows an exemplary and non-limiting flowchart **300** of the processing performed for the purpose of generating web pages that accelerate access in accordance with an embodiment of the invention. In S**310**, a page is received, for example by the WPAA **140**, in response to a request to receive a web page from, for example, web page server **120**. Optionally in S**320**, the received web page is stored in a cache, for example, the BEC **144**. In S**330**, the received web page is analyzed by acceleration server **142** using one or more acceleration and perfecting techniques (methods), to determine whether acceleration improvements may be achieved. In S**340**, it is checked whether improvements were determined to be achievable, and if so execution continues with S**350**; otherwise, execution continues with S**360**. In S**350**, the received web page is modified into a modified web page that contains one or more acceleration techniques discussed herein below in more detail. In S**360**, the modified or the received web page is provided to the user node **130** that requested the web page. Optionally in S**370**, the modified web page or the received web page, as may be appropriate, is stored in a cache, for example FEC **146**. In S**380**, it is checked whether additional pages are to be handled and if so execution continues with S**310**; otherwise, execution terminates.

While reference is made hereinabove to web pages, it can equally refer to portions of web pages, resources of a web page, and the like, without departing from the scope of the invention. Resources of a HTML web page include, but are not limited to, stylesheet files, Javascript and other script files, images, video and any other parts of the pages which are not embedded in the HTML.

The method disclosed above may be performed by the WPAA **140**, but without limitations. May be used in other web acceleration embodiments of the invention, including, integration in a web page server such as a web page server **120**.

While the description hereinabove was made with respect to one particular system, other systems may be deployed to benefit from the teachings hereinabove and herein below. In one exemplary and non-limiting embodiment of the invention, a system that works as a plug-in/filter/extension to one or more web servers is used. The flow of data through the system is the same as described with respect of the system in FIG. **1**, however, it may also utilize knowledge about the data stored on the web site, such as but not limited to, page template and images. In yet another exemplary and non-limiting embodiment, the disclosed pre-fetching acceleration techniques may be implemented in whole or in part as one or more plugins of a web site integrated development environment (IDE). Using a plugin, the inventions herein are integrated into the web site during its development. The plugin therefore enables at "compilation" or "build" process of the IDE, changes to the web site coding made by the user of the web site developer according to the inventions. This may take place during development or automatically implemented during development. In yet another exemplary and non-limiting embodiment of the invention, a utility containing, for example and without limitation, a command line component, a user interface (UI) component or any other interface, is run on the designed web site code after it is ready, and/or in one or more points-in-time during the development thereof, to transform the web site code by employing the inventions herein.

Following are descriptions of acceleration techniques used with respect to, for example, S**350**, discussed above. However, the use of such techniques may be a part of other embodiments which are specifically included herein.

I. Web-Site and Browser Transparent Pre-Fetching

Conventional pre-fetching of resources in web pages may be implemented in one of the following ways: a) installing a browser plug-in or any other desktop software which fetches resources and pages in the background using its own algorithms; b) introducing new tags and syntax into the HTML, HTTP and Javascript to provide "hints" to the browser regard-

ing a potential pre-fetch, however, modern popular browsers do not provide any kind of support to such "hint language"; and c) designing, as a part of the website, a mechanism which pre-fetches resources with a mechanism that the browsers support, however, this puts a burden on the designer of the website to write and maintain this pre-fetch code.

In accordance with an embodiment of the invention, the pre-fetching is performed by deploying the WPAA **140** in the communication path between the web page server **120** and the client or user node **130**. In other embodiments of the invention, pre-fetching of web-pages' resources can be done using a proxy, which is a component running on the same machine as the web server **120** or any other appropriate solutions capable of achieving the teachings below. Neither the web page server **120** nor the browser on the user node **130** is ware that this component exists, i.e., the WPAA **140** is transparent to the operation of web page servers **120** and user nodes **130**. The WPAA **140** analyzes the pages going through it and changes the web page to contain the pre-fetch code that may be created using methods and mechanisms described herein. The definition of which resources to pre-fetch and where on the page to locate the code may be defined in configuration, reached by static analysis of pages or dynamic analysis of pages and traffic, determined using changing statistics, or other applicable techniques as well as any combination of thereof. The code generated instead of the original code of the web page is designed to be understood and processed by modern browsers and does not require any additions to it.

One advantage over prior art, is that even if the web site has not changed, the same page can contain code to pre-fetch different resources every time. This may be advantageous, for example, if or when the usage pattern of a web page changes. Moreover, the fact that neither the user **120** nor the web page server **130** needs to be aware of the existence of the WPAA **140** between them.

In one embodiment, the WPAA **140** intercepts the web page and parses it prior to sending it out to a user node **130**. The original web page may reside in the BEC **144**. The acceleration server **142** based on instructions contained in storage **148** parses the web page in accordance with the invention described above and provides a modified web page, which may also be stored in the FEC **146** for future use by other user nodes **130**.

II. Pre-Fetching Resources of Subsequent or Other Pages

Today's pre-fetch techniques, pre-fetch either whole pages or the HTML part of the web page. This is problematic if when the exact next pages are not necessarily known. For example, if the web page has a dynamic component it may change between accesses to the web page.

According to the principles of the invention there is performed pre-fetching of the resources of other or subsequent pages, with or without the HTML page itself. As many of the resources are common to several or all the pages that may be fetched and therefore pre-fetching such resources is likely to save fetching them for other pages.

In one embodiment, the WPAA **140** intercepts the web page and parses it prior to sending it out to the user. The original web page may reside in the BEC **144**. The acceleration server **142** based on instructions contained in the storage **148** parses the web page in accordance with the invention described above and provides a modified web page, which may also be stored in the FEC **146** for future use by other user nodes **130**.

III. Fetching Linked Pages on Demand Prior to Link Access

In some cases, after browsing a site for a while, some of the new pages load very quickly. Most of their resources are already in the browser's cache, so the only non-cached items are the HTML itself and a few resources which have not been seen on previous pages yet. In such cases, loading the HTML from the network is a big percentage of loading the entire web page, even when the HTML is loaded very quickly.

According to the principles of the invention there is added a script, for example, Javascript, which detects the mouse presence over a link, a button, or any other means pointing by means of a URL, or other applicable means, to another page. This script may be further enabled to detect whether the focus is on the link, the button, or otherwise point of entry to another URL, which is particularly relevant to cases when the navigation is done using the keyboard and not the mouse. After the detection, the script, might or might not wait a while to reduce the number of false positives, before it pre-fetches the relevant page. If, during this time, the mouse moved from location of a URL, or has otherwise lost its focus of the web page, the pre-fetch is canceled. If the page pointed to by the link is small and the server is fast, many times it is possible to bring the page to the browsers cache before the link is actually clicked, thus substantially reducing the load time of that page as it appears to the user.

In one embodiment of the invention, a post-processing tool parses a web page prepared by a developer for adding a detection script implementing the principle described above. In another embodiment, the WPAA **140** intercepts the web page and parses it prior to sending it out to the user. The original web page may reside in the BEC **144**. The acceleration server **142** based on instructions contained in storage **148** parses the web page in accordance with the invention described above and provides the web page with the detection script implementation. The modified web page may also be stored in the FEC **146** for future use by other user nodes **130**.

IV. a Path Dependent Web Page Delivery to a User

On the same web site, many pages have common resources. Thus, it is important to know on a page whether the resources are already in the browser's cache or not. For example, different optimizations should be applied on the page to load it faster.

As for most web sites the resources in the cache typically expire within several hours, it is usually correct to assume that if the page was reached from a different page in the same web site, the common resources will be in the browser's cache and if not, most of them will not. Thus, according to an embodiment of the invention, a web page is processed differently for a case where it was reached from within the web site and for the case it was reached from without the web site. The conclusion about where the page was reached is determined according, for example but without limitation, the HTTP headers of the web page, a special cookie, the existence of a "referrer" header, a configuration or any other technique.

In one embodiment, the WPAA **140** intercepts the web page and sends the user a different version thereof. The original web page may reside in the BEC **144**. The acceleration server **142** based on instructions contained in storage **148** parses the web page in accordance with the invention described above and provides the web page with the detection script implementation. The modified web page may also be stored in the FEC **146** for future use by other user nodes **130**. Other implementations may include, without limitations, having a component as part of the web page server **120** enabled to perform this acceleration technique, or installing a software utility on the user node **130**, enabled to transform the web page differently according to its origin.

V. Caching of Dynamic Data

There are several levels of server-side caching that a web server (e.g., a server **120**) can use to increase its performance. If the web page is generated every time it is requested, one of

this cache levels can be, and many times is, to keep a generated version of the page and serve it every time, re-creating it only when the content of the page is changed. If the HTML content of the page contains a part which differs between several instances of the page, and the number of such different instances is great, it is impossible to keep a cache of the described type as on the HTML level, every different bit means a different page. Some web sites solve it by putting all the dynamic data, which changes between the instances, into a separate HTML page and load it as a "subpage". However, many sites have this type of data embedded into their HTML document, thus cannot maintain a cache of the described type.

According to an embodiment of the invention, caching of dynamic data includes separating the static parts of the HTML page from the dynamic parts. This can be performed by configurations, for example, "marking" parts of the HTML as static or dynamic, or automatically, by studying instances of the same page and deducing which parts are common. Once the static and dynamic parts of the page have been marked, this information can be used in two ways: a) the static part can be processed, for example, to achieve optimizations using, for example, techniques discussed in this document or otherwise, and the processed data kept in cache. Once a request for the page is accepted, the original page is requested and then the dynamic parts of it are "applied" on the processed static parts. The resulting page is then sent to a user node **130**; and b) the static part, processed or not, is sent to a user node **130** from the cache, without a request to the web server to obtain the original page. However, a code is injected into the page which directs the browser, without any need of additional support, to asynchronously send additional one or more requests to retrieve the dynamic data (see, for example, techniques to read resources into the cache). When the addition data is retrieved, it is injected into the DOM to the relevant places. In a preferred embodiment, this technique can be utilized web pages in which the dynamic part is relatively small, for example, the dynamic part includes fields where the username of a user is entered. In such web page, the entire page is first read from the server-side cache (e.g., BEC **144**) and only the username's value is read from the web server (e.g., one of web servers **120**) and is displayed later in the page. As the dynamic data is brought in an asynchronous way, this technique does not delay the loading of the common data.

In one embodiment of the invention, a post-processing tool parses a web page prepared by a developer for separating static and dynamic parts of the HTML page in accordance with the principle described above. In another embodiment, the WPAA **140** intercepts the web page and parses it prior to sending the page out to a user node **130**. The original web page may reside in the BEC **144**. The acceleration server **142** based on instructions contained in storage **148** parses the web page in accordance with the invention described above and separating static and dynamic parts of the HTML page for the modified web page. The modified web page may also be stored in the FEC **146** for future use by other user nodes **130**.

VI. Intelligent Caching of Resources

The cache in browsers operates in two ways. In a first aspect of operation, once a resource is loaded to the browser, the resource may include a header which instructs the browser how long it should be in its cache (either a period of time of the time of the expiry of the cache). While this resource is in the browser's cache, every time the browser needs it, the browser reads the resource from the cache and does not send a request for this resource. One the cache expires, the resource is deleted from the cache and the next time the browser needs the resource, the browser sends a request to get the resource. Any file type, such as but not limited to, HTML, Javascript, images, and CSS, may be affected therefrom. In a second aspect of operation, once a resource's time in the browser's cache is expired, and the browser needs the resource, the browser sends a request to the web server along with some information about the resource that it has in its cache, for example, its last modification time. Then the server may return, instead of the content of the request a message confirming that the existing resource that is up to date and may be used.

A severe limitation is that once a resource is in a browser's cache, it cannot be invalidated, except for explicitly doing so by the user by clicking "clear cache" in the browser. As for this resource, requests to the web server (e.g., one of web servers **120**) are not sent, and furthermore, the server cannot even send a message indicating that the resource is not up to date. Thus, a web server cannot set too long of an expiration time as the resource may change and the browser will not be cognizant of it. On the other hand, any request, even when resulting with the server sending a message indicating that the resource is up to date, is time consuming and often delays the loading time of the web page, thus setting too short an expiration period hurts performance.

In accordance with the invention, every resource is equipped with a version indication and this version is increased every time when the resource is changed. Every whole web page, i.e., a web page along with all its resources, also has a version. This version is a tuple, or otherwise a combination of the versions of all its resources, including the HTML page itself. In one example, a hash function may be used to change the resource's version every time any resource is changed. The version of the page, or an identifier that stands for this version, is sent to the user along with the web page every time it is requested. If the user has already accessed that page once or more, the version of the page the user has, or an identifier which stand for this version, is sent with the request. In one embodiment, this is achieved by means of cookies. All the resources are sent with a very long expiration time. Thus, when the browser encounters these resources it will take them from the browser cache.

Once the web server (e.g., one of web servers **120**) receives a request for web page, the WPAA **140** intercepts the request and checks the difference between the user's version and the current version of a resource in the requested web page. All the resources that have not changed are referenced "as is" and the references to resources which have changed are rewritten to point to the new references. This is done by changing the filename of the pointed resource or its query string. For example, but not by way of limitation, changing the reference of "image.1.jpg" to point to "image.2.jpg" which is the newer version or changing the reference of "image.jpg?ver=1" to point to "image.jpg?ver=2". This way requests are made to the new versions of the changed resources.

The disclosed technique can also be applied to parts of resources. For example, if the difference between the new version of the HTML and the version the user has is only one line and the HTML is big, the browser can request only this one line and run a client-side code which applies this line into the cached data. In order to use this technique on HTML pages, a stub can be used in the following sequence: a) the browser requests a file. This file is very small and the web server directs the browser never to cache it. Along with the request the version of the web page is sent; and, b) the server then directs the browser (can be used in a number of ways: a response "redirect" header or a code embedded in the response which forces the browser to request a new page or to

bring the difference and apply it). The redirected page depends on the version and may be cached.

VII. Processing Links in the Background

When prior to serving a page a web server (e.g., one of web servers **120**), or a proxy, has to do some processing, for example, generate or optimize the page. Such processing delays the loading of the page. In some cases, processing of a page can be performed in the background, but there are web sites with a huge amount of pages, for example, results of different queries, and they cannot be all processed in the background in a reasonable time.

When a page is served by a web server or proxy, the pages which are candidates for being the next pages to be server for the same user nodes are also processed. The candidate web pages can be deduced either statically from analyzing the current page, for example, by looking at all the links, or by collecting statistics and choosing accordingly, by a configuration or a combination, for example, checking if there are many links and the order of their processing is defined by the statistics. This can be done recursively to any depth and repeating the process on all the chosen pages.

VIII. Sending the Common Part of a Plurality of Web Pages Once

Typically, pages of a particular web site contain common data. This is done mostly to keep a consistent look and feel to the web site. This is notable for pages derived from the same template, but also in the case of pages from different templates. Every time a browser requests a page, the response contains the entire page, including the common parts. This is repeated over and over again and of course burdens the bandwidth requirements and slows it unnecessarily as no new data is in fact transferred to the viewer. According to an embodiment of the invention, the common data between pages is sent only once.

Therefore, in accordance with the principles of the invention only the non-common data of a page is sent to the user every time that the user requests the page, while the common data is sent only once. This requires a server side component, or a proxy in the pathway between the user node (e.g., a user node **130**) and a web server (e.g., one of the web servers **120**). In an embodiment of the invention, this acceleration technique is performed by the WPAA **140**.

Specifically, the WPAA **140** computes the differences (also referred to herein as "diff") between the requested pages and sends only such differences to a browser of a user **130**. In addition, only for the first time that the page was requested the common portions are sent as well. A client side component (e.g., a user node **13**) receives the common parts once, and then creates the entire data item using the common parts and differences for a specific page. It should be noted that either item itself or its representation, for example, in DOM format, may be sent.

The diff can be created in various levels and using a variety of algorithms. The diff can be created from the text of the web pages, or from any logical representations thereof, for example and without limitation, the DOM of the page can be used to create a diff, when the pages are compared on the DOM elements level. One example of creating a diff is using a "rsync"-like algorithm. The diff of a page the user navigates to, or for that matter any other page, can be calculated using the current page the user is at or any other pages the algorithm recognizes or can assume that the user has. This way a page for which the diff is the smallest can be chosen. The diff can be applied by loading the non-common part using a synchronous connection and then applying the diff, or using asynchronous communication such as AJAX and then when the diff is ready, adding the diff to the web page.

FIGS. **4**A and **4**B are non-limiting and exemplary flowcharts illustrating the operation of the "sending the common part of a plurality of web pages once" acceleration technique in accordance with an embodiment of the invention. This technique is performed by a client side which may be one of the use nodes **130** and a server side which may be the WPAA **140**.

The processing of a web page according to this technique is initiated once a user clicks on a link. The link may be in a form of a URL directing to another page. According to this embodiment, the user action (i.e., clicking on the linking) invokes an AJAX query to the server side. The query contains the current URL, the URL that the user wants to navigate to, and information about which additional URLs from the same website the user's browser cache contains.

The AJAX query initiates the process performed by the server side and further depicted in FIG. **4**A. In S**411**, the server side receives the AJAX query. At S**412**, it is determined which of the URLs that the user already accessed, has the smallest diff from the requested URL. The (old) page that corresponds to the URL is marked as "page-1" and the page that corresponds to the URL that the user navigates to is marked as "page-2". Thereafter, a pointer is created and further pointed to the beginning of page-2 (S**413** and S**414**). The pointer is used for sequentially scanning of page-2 when the diff is computed. Once the pointer reaches the end of page-2, the process terminates (S**415** and S**418**).

At S**416**, the diff between page-1 and page-2 is computed to create a list of non-matching data blocks and their positions in-page 1. In exemplary embodiment of the invention, S**416** can be performed using any rsync-like algorithms. One with ordinary skill in the art would be familiar with the operation of such algorithms. At S**417**, the server side sends to the client side the created list and the URL of page-1.

The client side, upon receiving the list of non-matching data blocks, applies the diff to page-1 to create the modified page which should is identical to page-2. That is, the nee page to be viewed is a combination of the content of page-1 and the diff as contained in the received list. The client side can use a rsync mechanism to combine the received diff and page-1. It should be noted that page-1 is stored in the client side's cache.

FIG. **4**B shows a non-limiting and exemplary flowchart illustrating the process performed by the client side (e.g., a user node **130**) in accordance with an embodiment of the invention. The process creates a modified page-2 that contains the content of page-1 and non-matching blocks. In S**421**, a list of non-matching block as computed by the server side (e.g., WPAA **140**) is received. In S**422**, an empty container for the modified page-2 is created. The process scans the received list from its beginning unit its end when filling the container (S**423**, S**424** and S**428**). In S**425** it is checked if a current selected entry is the received list describes a data block in page-1, if so, in S**426**, the respective data block from page-1 is append to the container; otherwise, in S**427**, the respective data block from the received list is appended to the container. At S**429**, once all entries in the created list have been checked, the container including the contents of the modified page-2 is output.

This acceleration technique can be used in combination with the use of the browser's cache, with resource combining, or in-lining. When resources are being in-lined inside an HTML page, or for that matter any other container, the cache of the browser is less efficient than when the resources are taken from external files. However, using the described technique, when the same resource is in-lined in two or more different pages, it is not loaded from the server twice, as the

"rsync" algorithm, or another appropriate algorithm, compresses it to several bytes only.

Following is a non-limiting example describing this acceleration technique as applied on resources contained in a web page. In this example, an old resource (resource-1) contains the data "zabcd1234" and a new resource contains (resource-2) the data "yxabc34". The process for creating a list of non-matching data blocks (e.g., as shown in FIG. **4**A) outputs the list in a form of a data structure illustrated in FIG. **5**A through FIG. **5**C.

The process, as described with reference to FIG. **4**B, uses that created list and resource-1 (containing "zabcd1234") to generate a modified resource-2 as follows: after processing block 0, the modified resource-2 container contains "yx", after processing block 1, the modified resource-2 container includes "yxabc", where the new data added being "abc" as it is the data which is located between positions 1 and 3 of resource-1. Then, upon processing of block 2, the modified resource-2 container includes "yxabc123", where the new data being added is "123" located between positions 5 and 7 of the resource-1. As can be noticed the container include the content of the resource-2.

It should be noted that this process accelerates the access to the new page/resource (page-2/resource-2) as instead of loading the new page only differences should be retrieved from the server-side as the content of the old page/resource (page-1/resource-1) is cached at the client side.

IX. A Technique for Measuring the Load Time of a Web Page

The load sequence of a typical web page consists of many different resources. Some of the resources are visible while others are not. Some of the visible resources are part of the viewport and others are not. Additionally, the speed a web page loads, is that which is perceived by a user once the viewport is complete. It is close to impossible to deduce when the page has finished loading from the user's point of view based merely upon network analysis. Current measurement techniques either calculate such time once all the components of the page have finished loading, including those that are invisible, or check the "onload" event of the HTML document, which also has only a small correlation to the actual user perceived load time.

In most websites, the last item to be loaded in the viewport is a graphic item, such as an image of a Flash object. This happens because the size of graphic items is big and takes time to load. Furthermore, graphic items are often loaded later than the textual, i.e., HTML, JavaScript, CSS, etc., elements. Using this assumption, the following acceleration techniques measures the actual perceived load time of a web page:

- For every background image, the server creates invisible dummy images that are marked as loaded when the background image finishes loading.
- The time the page starts loading is saved in memory.
- Once every predefined time interval, that can be set to different values, depending on the desires granularity of the result, the following is performed:
  - All the graphical elements in the web page are checked, for example, by scouting the DOM of the web page, by lists exported by the browser, such as document images, or by any other means. The elements can be images, Flash objects, or any other type of element.
    - For every element found, its position is calculated, for example, by using all the elements starting from the desired elements and finishing at the root of the DOM tree, or by any other way.
    - If the element's position is in the viewport, it is added to the known viewport element list.
  - Save to storage, persistent or not, a graphical snapshot of the screen that may contain only the browser, or any other part of the screen, along with the elapsed time passed since the start load time. These snapshots can be later analyzed to determine the exact time the viewport has finished rendering.
  - Repeat the process for all the known element lists previously created.
    - If any of the elements has not finished loading yet, which can be determined by a readyState property or any other way, then wait for the next iteration.
    - If all the elements are loaded, check if enough time has passed since the last element in the list was loaded and since the list was last changed. If enough time passed, where "enough time" can be defined to be any suitable value, the time the last resource in the list was loaded is marked as the time the page ended loading.
  - Return, display, or otherwise store in memory the load time which is the end of loading time minus the start of loading time, in addition to marking the snapshot corresponding to the time the document perceived to be fully loaded.

X. Using Versioning to Cache Combined Files

One technique to reduce latency when reading multiple resources is to combine resources files, and thereby reducing the overall latency. When creating combined resource files, one loses the advantage of the browser cache. Thus, the same resource which is part of two or more files now combined will not be cached between these different files.

Every resource from a combined file is assigned a unique identifier which includes its version. This can be any unique number and it can be any hash function of the content or the name or URL of the file. For example, the popular hash function MD5 can be used to assign a unique identifier to a hash function. Either the use node **130** or the web server **120** holds in their internal storage the identifiers of the resources already read and have in the respective cache. In case of a user node **130**, the identifiers may be stored in a browser cache, a Flash storage, a local storage, or any other storage type. In case the server holds this data, it holds it for every client, either in memory, or in storage such as a disk or any other location. In this case, every user is uniquely identified (for example using an identification cookie), thus this data can be saved for any user separately.

In the case where the user node **130** (or client) stores the resources data, the web server **120** adds a script at the beginning of every web page that performs the following actions:

- For every relevant resource on the page:
  - Check if this resource, including version, is present in the storage.
    - If it exists, replace the URL of the resource pointed to by the cached inline file, and as may be applicable to a position in it.
    - If it does not exist, add its path to the list of missing resources and replace the URL of the resource pointed to by a new combined file which contains all the missing resources, and as may be applicable to a position in it.
  - Send the list of all the missing resources as part of a request to the server. This request asks for an inlined file with all the missing resources. The names or identifiers of the missing resources may be passed in the query string or in any other way.

This way all the resources which were already seen by the browser in previous combined files are taken from there and all the resources which were not previously seen by the

browser will arrive in a new single combined file. Another way to implement this is to save all the information about every user in the sever side. This way the page which is served by the server already contains the correct URL's (whether those which are already in the cache or new ones) and it needs not to be replaced by a client-side script.

In the case where the web server **120** stores the resources data, the server **120** performs the following action of the web page before sending it to the user:

- Identify the user (for example using a cookie).
- If the user was not found, assume that the user does not have any resource in the browser's cache.
- If the user is identified, get from the storage (either memory, or disk or any other storage) the resources that the user has in the browser's cache and the names of the container that have these resources in them.
- Create one or more empty containers, which will be used for the resources the user does not already have.
- Scan the web page for resources and for each resource performs the following:
  - If the user already has it in the browser's cache, replace the reference to the resource by the reference of the resource in the container the user already has.
  - If the user does not have the resource in the browser's cache, add the resource to one of the prepared containers and change the reference to the resource to point to the resource in the container.
- At the end of the process all the resources the user already has will be referenced to containers the user has in its cache and the new resources will be referenced to the resources in the new containers. Thus only the new resources will be downloaded by the client, combined in the prepared one or more containers.

In one embodiment of the invention, the tasks performed by the web server **120** when combining resources can be performed by the WPAA **140**. According to this embodiment, the WPAA **140** intercepts the page before sending to the user node **130**, determines where the resources data resides, and modifies the web page based on the location of the resources data.

The principles of the invention and embodiments thereto are implemented as hardware, firmware, software or any combination thereof. Moreover, the software is preferably implemented as an application program tangibly embodied in a program storage unit, a non-transitory computer readable medium or a non-transitory machine-readable storage medium that can be in a form of a digital circuit, an analogy circuit, a magnetic medium, or combination thereof. The application program may be uploaded to, and executed by, a machine comprising any suitable architecture. Preferably, the machine is implemented on a computer platform having hardware such as one or more central processing units ("CPUs"), a memory, and input/output interfaces. The computer platform may also include an operating system and microinstruction code. The various processes and functions described herein may be either part of the microinstruction code or part of the application program, or any combination thereof, which may be executed by a CPU, whether or not such computer or processor is explicitly shown. In addition, various other peripheral units may be connected to the computer platform such as an additional data storage unit and a printing unit.

The foregoing detailed description has set forth a few of the many forms that the invention can take. It is intended that the foregoing detailed description be understood as an illustration of selected forms that the invention can take and not as a limitation to the definition of the invention. It is only the claims, including all equivalents that are intended to define the scope of this invention.

What is claimed is:

1. A system for accelerating access to resources of web pages, the system comprising:

a cache for storing resources of web pages;

an acceleration server that:

receives a first web page in response to a request by a user system;

parses the first web page;

identifies, based at least partially on parsing the first web page, a second web page, wherein the second web page is likely to be requested by the user system after the user system accesses the first web page;

identifies a first resource, wherein the first resource is a resource of the second web page;

requests the first resource from a server hosting the first resource before the user system requests the first resource;

receives the first resource from the server hosting the first resource;

stores the first resource in the cache; and

provides the first resource to the user system from the cache in response to a request by the user system for the first resource; and

a network interface that:

enables communication of one or more user nodes with one or more web servers over a network for accessing web pages stored in one or more servers;

receives the request for the first web page, wherein the request for the first web page originates from the user system; and

receives a request for the first resource, wherein the request for the first resource originates from the user system after the acceleration server requests the first resource from the server hosting the first resource.

2. The system for accelerating access to resources of web pages as recited in claim **1**, wherein the network interface receives a request for the second web page and the acceleration server provides the second web page to the user system.

3. The system for accelerating access to resources of web pages as recited in claim **1**, wherein:

the acceleration server further identifies a third web page, such that the first resource is common to both the second web page and the third web page; and

the acceleration server stores the first resource in the cache based on the first resource being common to more than one web page.

4. The system for accelerating access to resources of web pages as recited in claim **1**, wherein the network interface receives the request for the first web page by intercepting the request for the first web page.

5. The system for accelerating access to resources of web pages as recited in claim **1**, further comprising a second cache, separate from the cache, and stores the first web page in the second cache before the first web page is requested by the user system.

6. The system for accelerating access to resources of web pages as recited in claim **1**, wherein the first resource is a static component of the second web page.

7. A method for accelerating access to resources of web pages, the method comprising:

receiving the request for a first web page, wherein the request for the first web page originates from the user system;

receiving the first web page in response to a request by the user system;

providing the first web page to the user system;

parsing the first web page;

identifying, based at least partially on parsing the first web page, a second web page, wherein the second web page is likely to be requested by the user system after the user system accesses the first web page;

identifying a first resource, wherein the first resource is a resource of the second web page;

requesting the first resource from a server hosting the first resource before the user system requests the first resource;

receiving the first resource from the server hosting the first resource;

storing the first resource in a cache; and

receiving a request for the first resource, wherein the request for the first resource originates from the user system after the acceleration server requests the first resource from the server hosting the first resource; and

providing the first resource to the user system from the cache in response to the request by the user system for the first resource.

8. The method for accelerating access to resources of web pages as recited in **7**, further comprising determining the second web page is likely to be accessed after the first web page based on parsing the first web page.

9. The method for accelerating access to resources of web pages as recited in **7**, further comprising identifying a third web page, such that the first resource is common to both the second web page and the third web page.

10. The method for accelerating access to resources of web pages as recited in **7**, further comprising:

identifying a second resource that is common to the second web page and to a third web page; and

storing the second resource in the cache based on the second resource being common to more than one web page.

11. The method for accelerating access to resources of web pages as recited in **7**, wherein the first resource is common to multiple web pages that are configured to be accessed from the first web page.

12. The method for accelerating access to resources of web pages as recited in **7**, wherein the first resource is common to all other web pages that are configured to be accessed from the first web page.

13. The method for accelerating access to resources of web pages as recited in **7**, wherein the first resource is a static component of the second web page.

14. The method for accelerating access to resources of web pages as recited in **7**, further comprising analyzing the first web page for applying an acceleration technique to the first web page.

15. A memory device having instructions that when executed cause one or more processors to perform the following steps for accelerating access to resources of web pages:

receive the request for a first web page, wherein the request for the first web page originates from the user system;

receive the first web page in response to a request by the user system;

provide the first web page to the user system;

parse the first web page;

identify, based at least partially on parsing the first web page, a second web page, wherein the second web page is likely to be requested by the user system after the user system accesses the first web page;

identify a first resource, wherein the first resource is a resource of the second web page;

request the first resource from a server hosting the first resource before the user system requests the first resource;

receive the first resource from the server hosting the first resource;

store the first resource in a cache; and

receive a request for the first resource, wherein the request for the first resource originated from the user system after the acceleration server requested the first resource from the server hosting the first resource; and

provide the first resource to the user system from the cache in response to the request by the user system for the first resource.

16. The memory device having instructions as recited in claim **15**, that when executed further cause the one or more processors to identify a third web page, such that the first resource is common to both the second web page and the third web page.

17. The memory device having instructions as recited in claim **15**, wherein the instructions further cause the one or more processors to determine the second web page is likely to be accessed after the first web page based on parsing the first web page.

18. The memory device having instructions as recited in claim **15**, wherein the first resource is common to multiple web pages that are configured to be accessed from the first web page.

19. The memory device having instructions as recited in claim **15**, wherein the second web page is an HTML web page that uses a plurality of resources, and the first resource is one of the plurality of resources.

20. The memory device having instructions as recited in claim **15**, wherein providing the first resource is performed without providing the second web page to the user system.

* * * * *

Exhibit D

US008683002B2

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

(10) **Patent No.: US 8,683,002 B2**
(45) **Date of Patent: *Mar. 25, 2014**

(54) **CONTENT DELIVERY NETWORK CACHE GROUPING**

(71) Applicant: **Limelight Networks, Inc.**, Tempe, AZ (US)

(72) Inventors: **Bradley B. Harvell**, Chandler, AZ (US); **Nils H. McCarthy**, Seattle, WA (US)

(73) Assignee: **Limelight Networks, Inc.**, Tempe, AZ (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **13/732,570**

(22) Filed: **Jan. 2, 2013**

(65) **Prior Publication Data**

US 2013/0246555 A1 Sep. 19, 2013

Related U.S. Application Data

(63) Continuation of application No. 13/525,671, filed on Jun. 18, 2012, now Pat. No. 8,370,449, which is a continuation of application No. 13/245,797, filed on Sep. 26, 2011, now Pat. No. 8,219,647, which is a continuation of application No. 12/732,942, filed on Mar. 26, 2010, now Pat. No. 8,219,645.

(60) Provisional application No. 61/248,378, filed on Oct. 2, 2009.

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

(52) **U.S. Cl.**
USPC **709/214**; 709/219; 711/122

(58) **Field of Classification Search**
USPC 709/214, 219, 203, 226, 224, 238; 370/351, 389; 718/105; 711/122
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,542,964	B1 *	4/2003	Scharber	711/122
6,785,704	B1 *	8/2004	McCanne	718/105
8,028,090	B2 *	9/2011	Richardson et al.	709/238
8,219,645	B2 *	7/2012	Harvell et al.	709/219
8,219,647	B2 *	7/2012	Harvell et al.	709/219
8,370,449	B2 *	2/2013	Harvell et al.	709/214
2002/0009079	A1 *	1/2002	Jungck et al.	370/389
2007/0025327	A1 *	2/2007	Raciborski et al.	370/351
2008/0071859	A1 *	3/2008	Seed et al.	709/203
2009/0248858	A1 *	10/2009	Sivasubramanian et al.	709/224
2009/0254661	A1 *	10/2009	Fullagar et al.	709/226

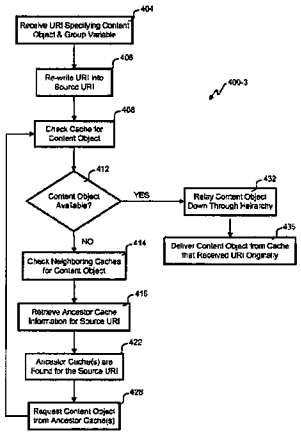
* cited by examiner

Primary Examiner — Jungwon Chang
(74) *Attorney, Agent, or Firm* — Kilpatrick Townsend & Stockton LLP

(57) **ABSTRACT**

One or more content delivery networks (CDNs) that deliver content objects for others is disclosed. Content is propagated to edge servers through hosting and/or caching. End user computers are directed to an edge server for delivery of a requested content object by a universal resource indicator (URI). When a particular edge server does not have a copy of the content object from the URI, information is passed to another server, the ancestor or parent server to find the content object. There can be different parents servers designated for different URIs. The parent server looks for the content object and if not found, will go to another server, the grandparent server, and so on up a hierarchy within the group. Eventually, the topmost server in the hierarchy goes to the origin server to find the content object. The origin server may be hosted in the CDN or at a content provider across the Internet. Once the content object is located in the hierarchical chain, the content object is passed back down the chain to the edge server for delivery. Optionally, the various servers in the chain may cache or host the content object as it is relayed.

20 Claims, 7 Drawing Sheets



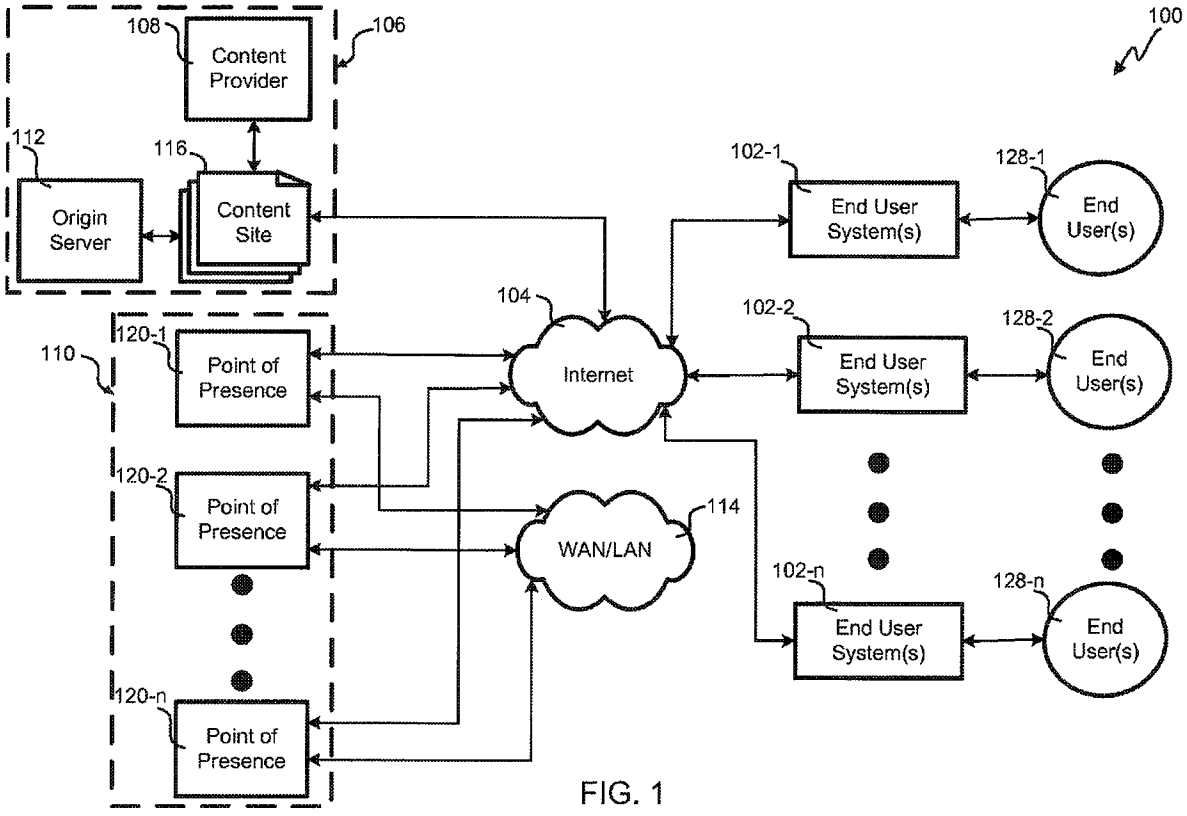

100
106
108
Content Provider
112
Origin Server
116
Content Site
110
120-1
Point of Presence
120-2
Point of Presence
120-n
Point of Presence
104
Internet
114
WAN/LAN
102-1
End User System(s)
128-1
End User(s)
102-2
End User System(s)
128-2
End User(s)
102-n
End User System(s)
128-n
End User(s)

FIG. 1

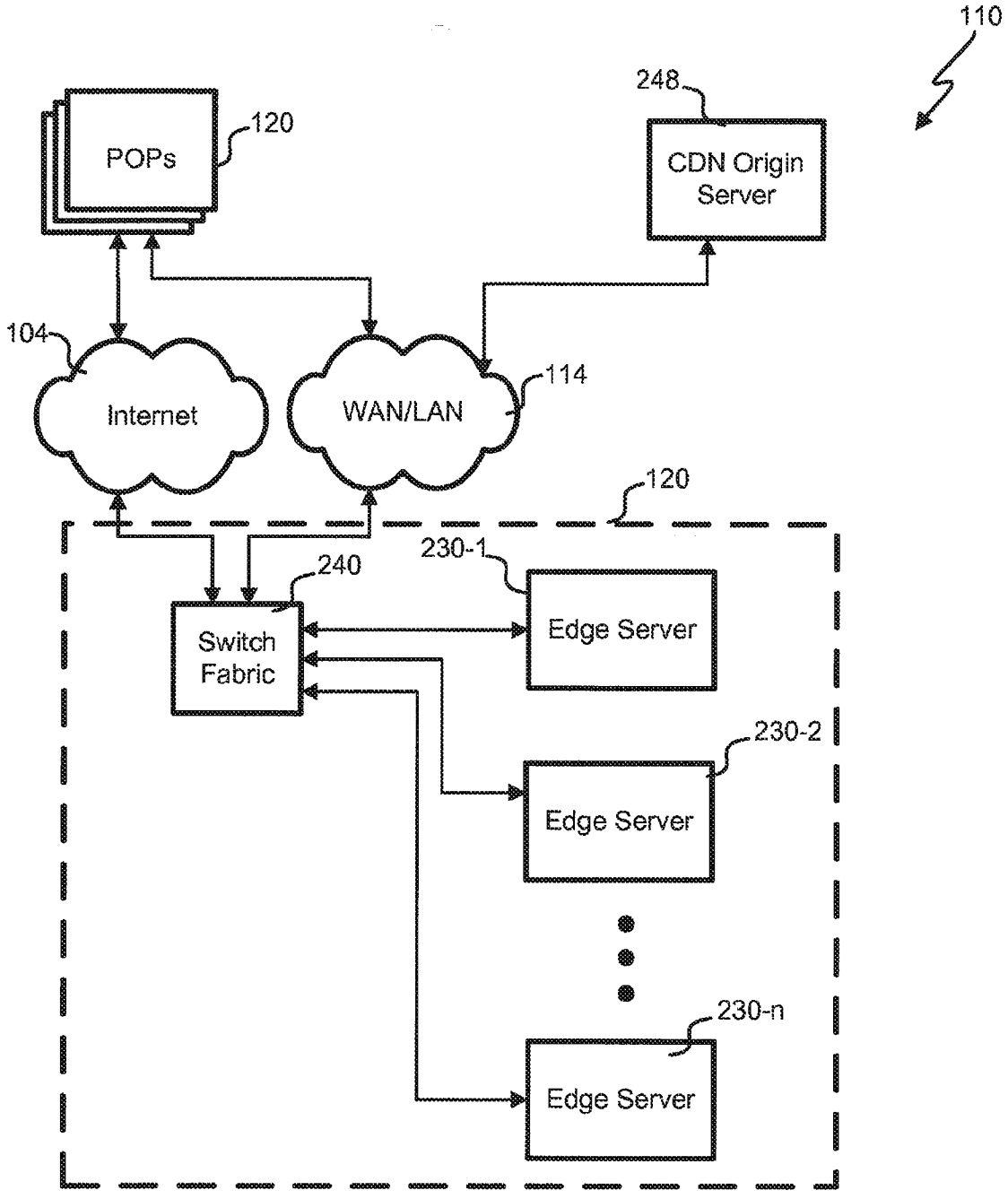

110
248
CDN Origin Server
120
POPs
104
Internet
WAN/LAN
114
120
230-1
240
Switch Fabric
Edge Server
230-2
Edge Server
230-n
Edge Server

FIG. 2

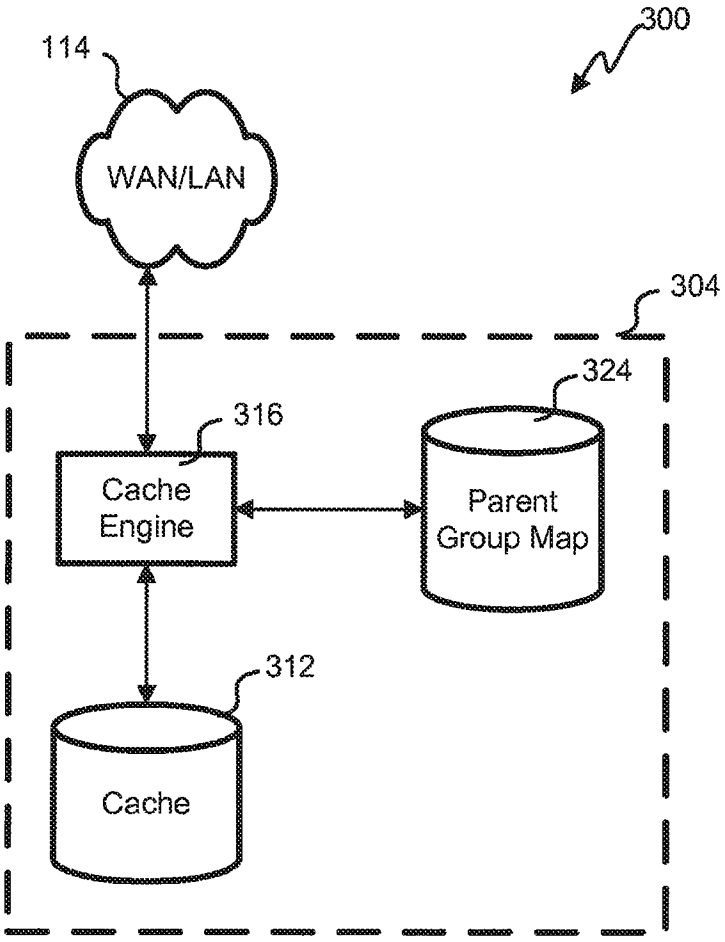

300
114
WAN/LAN
304
324
316
Cache Engine
Parent Group Map
312
Cache

Fig. 3

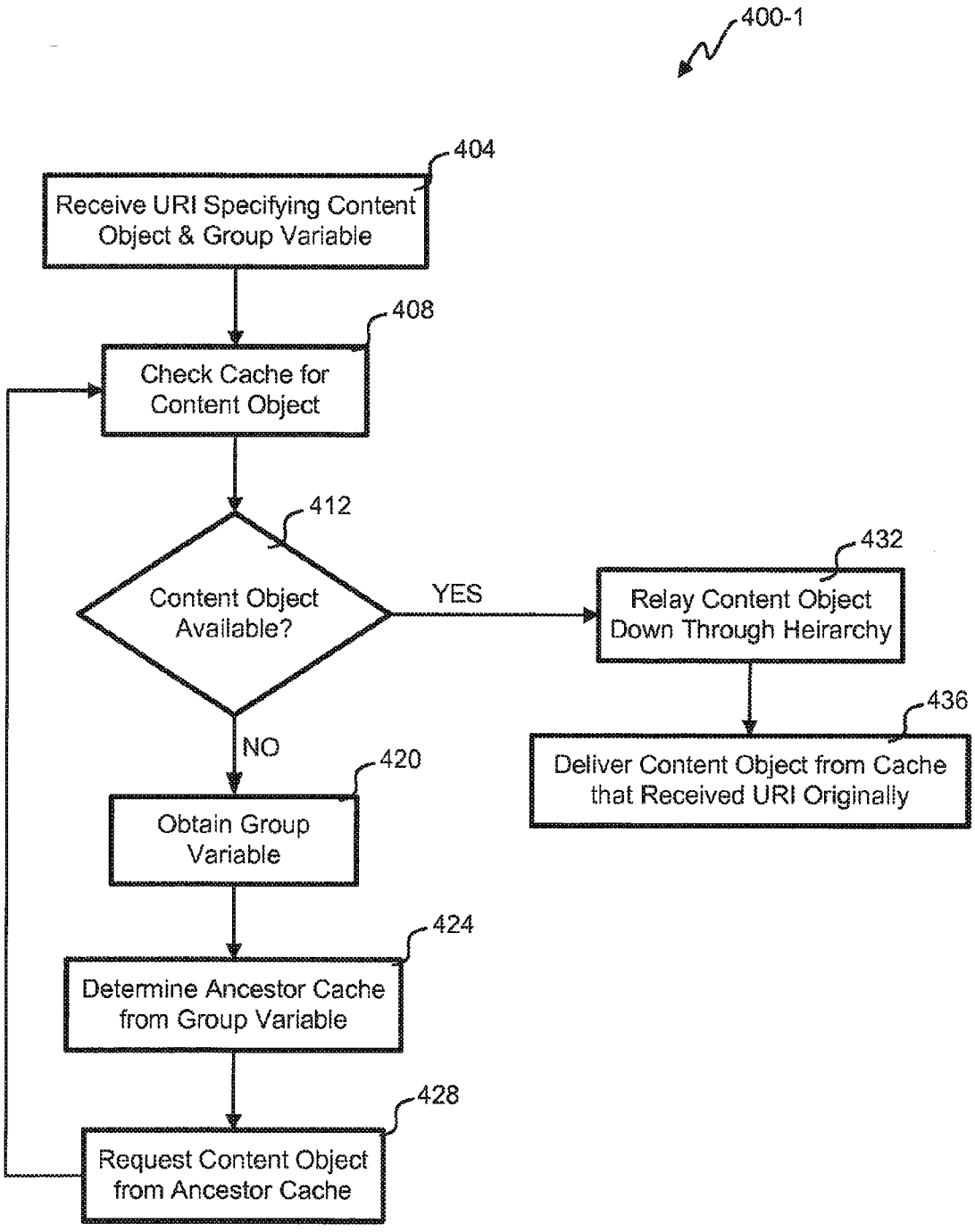

400-1
404
Receive URI Specifying Content Object & Group Variable
408
Check Cache for Content Object
412
Content Object Available?
YES
432
Relay Content Object Down Through Heirarchy
436
Deliver Content Object from Cache that Received URI Originally
NO
420
Obtain Group Variable
424
Determine Ancestor Cache from Group Variable
428
Request Content Object from Ancestor Cache

Fig. 4A

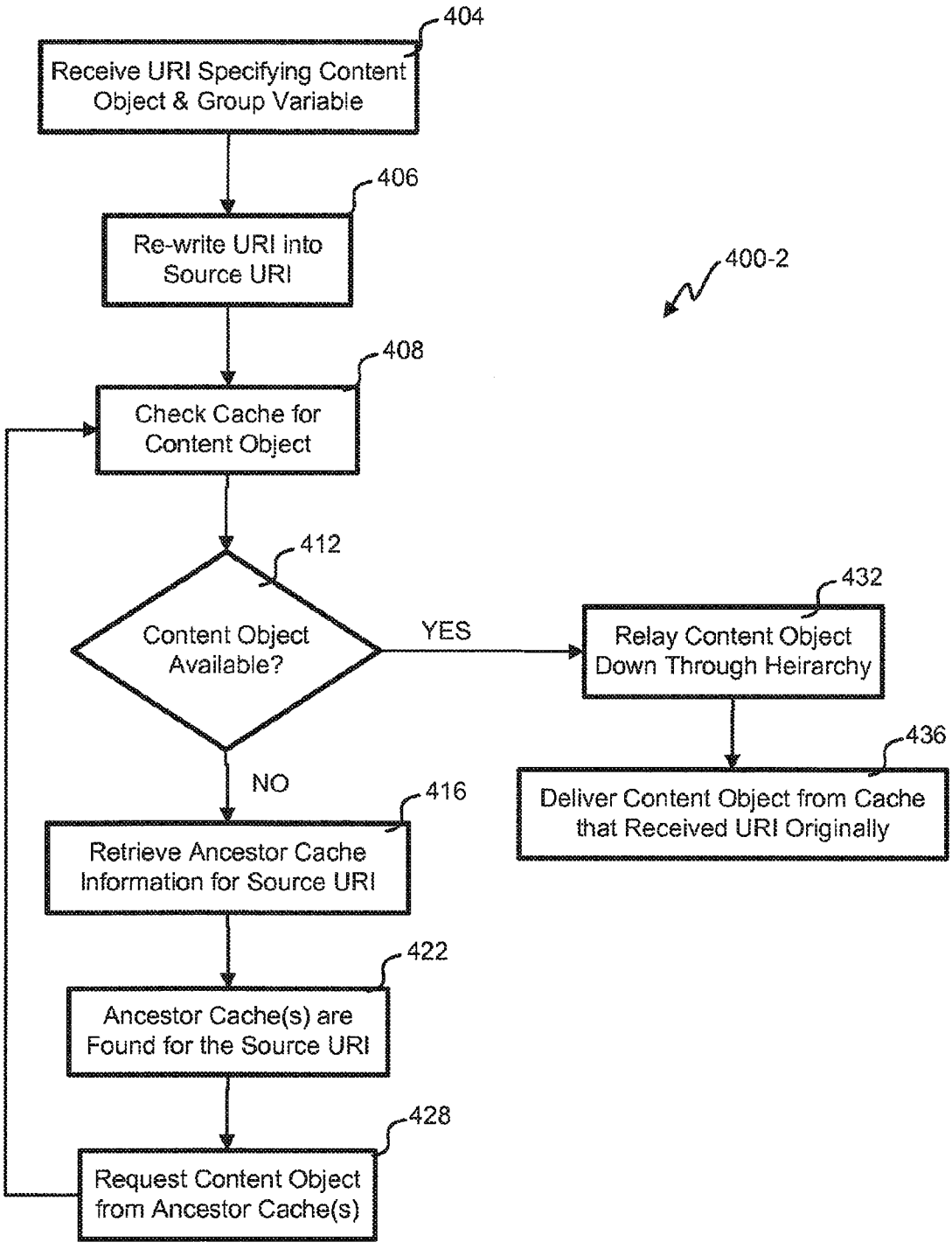

404
Receive URI Specifying Content Object & Group Variable
406
Re-write URI into Source URI
400-2
408
Check Cache for Content Object
412
Content Object Available?
YES
432
Relay Content Object Down Through Heirarchy
436
Deliver Content Object from Cache that Received URI Originally
NO
416
Retrieve Ancestor Cache Information for Source URI
422
Ancestor Cache(s) are Found for the Source URI
428
Request Content Object from Ancestor Cache(s)

Fig. 4B

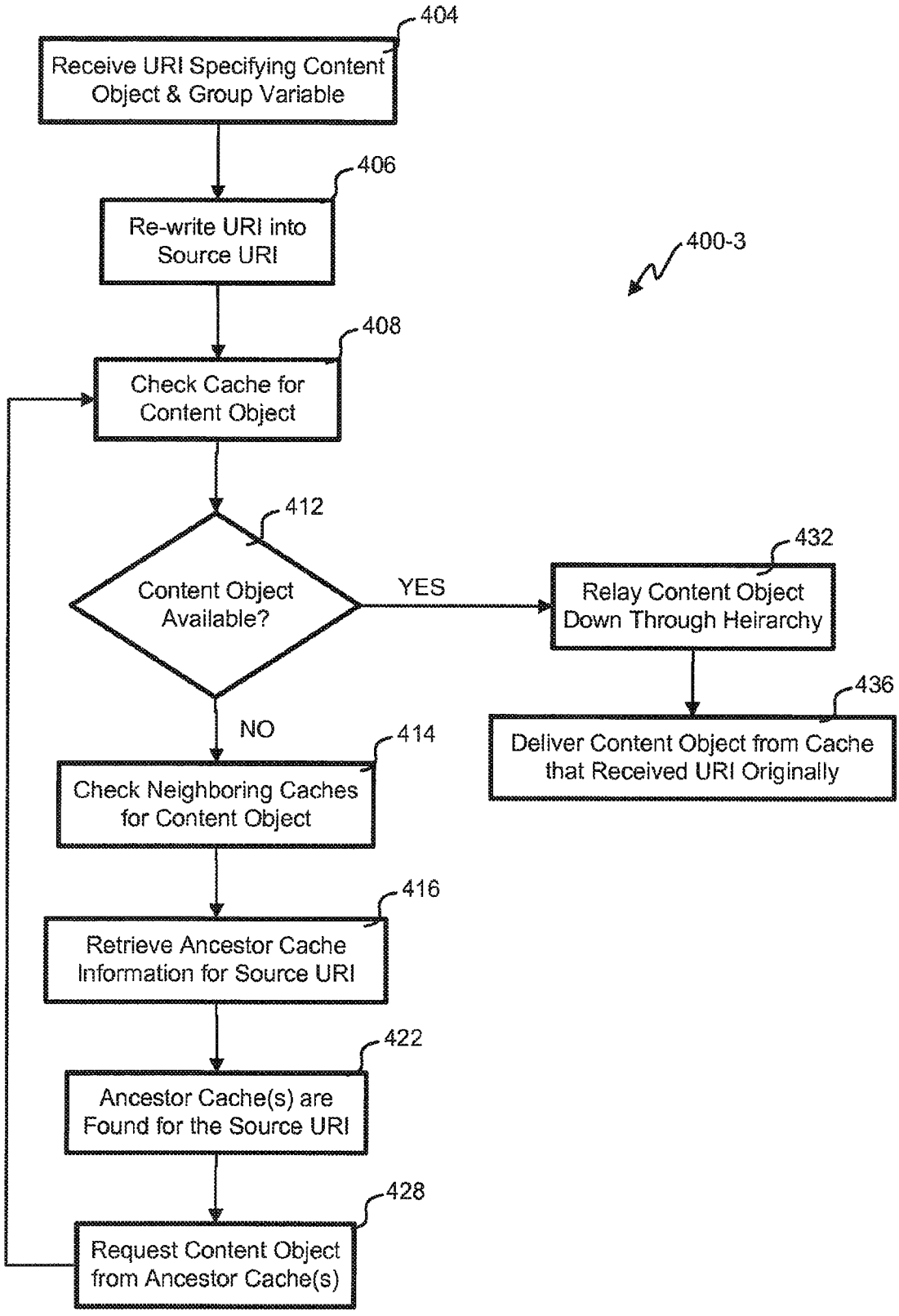

404
Receive URI Specifying Content Object & Group Variable
406
Re-write URI into Source URI
400-3
408
Check Cache for Content Object
412
Content Object Available?
YES
432
Relay Content Object Down Through Heirarchy
436
Deliver Content Object from Cache that Received URI Originally
NO
414
Check Neighboring Caches for Content Object
416
Retrieve Ancestor Cache Information for Source URI
422
Ancestor Cache(s) are Found for the Source URI
428
Request Content Object from Ancestor Cache(s)

Fig. 4C

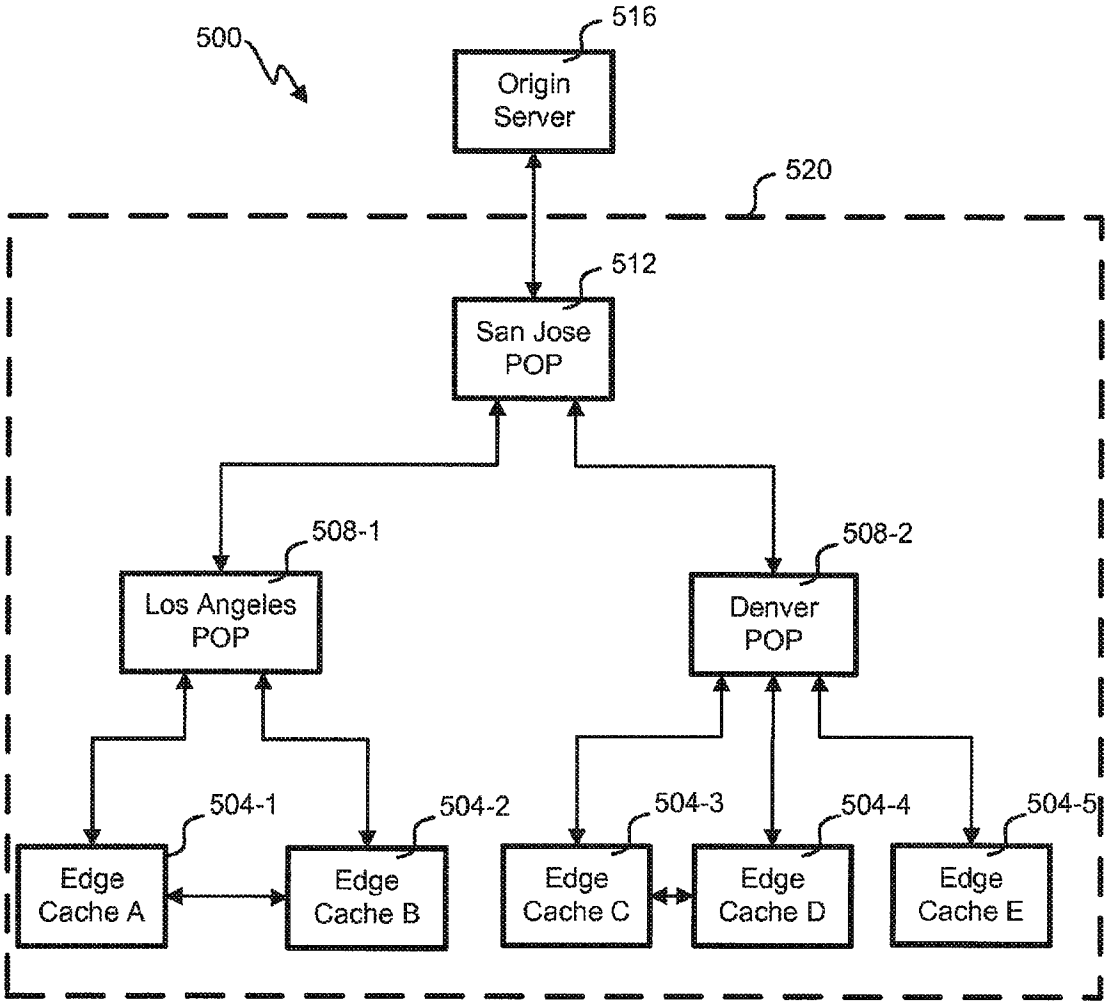

500
516
Origin Server
520
512
San Jose POP
508-1
Los Angeles POP
508-2
Denver POP
504-1
504-2
504-3
504-4
504-5
Edge Cache A
Edge Cache B
Edge Cache C
Edge Cache D
Edge Cache E

Fig. 5

CONTENT DELIVERY NETWORK CACHE GROUPING

CROSS-REFERENCES TO RELATED APPLICATIONS

This is a continuation patent application of U.S. patent application Ser. No. 13/525,671 filed on Jun. 18, 2012, which is a continuation of U.S. patent application Ser. No. 13/245,797 filed on Sep. 26, 2011, which is a continuation application of U.S. patent application Ser. No. 12/732,942 filed on Mar. 26, 2010, which claims the benefit of U.S. Application No. 61/248,378 filed Oct. 2, 2009. The entire disclosures of the above-listed applications are incorporated by reference in their entirety for all purposes.

BACKGROUND

This disclosure relates in general to content delivery networks and, but not by way of limitation, to serving content objects from edge server caches of a content delivery network.

Content delivery networks (CDNs) are in the business of delivering content for others. CDNs will either cache and/or host content for its customers. Efficiently delivering content for a large number of customers creates difficulty. It would not be practical to store every possible content object serviced by the CDN on every edge server. Often caches are used on the edge servers to store popular or important content at the edges of the CDN. Popular content is less likely to have delivery latency, while less popular content is more likely to take a longer time to locate and deliver.

In some cases, the content object is not available on the edge server. This situation is sometimes referred to as a cache miss. A universal resource locator (URL) provided to the CDN from a requestor is used to find the content with a cache miss. The content may be hosted internal to the CDN or with a content provider. Finding the content object can be time intensive and affect the quality of service (QoS) perceived by the requestor. This is especially true for content that cannot be located in the CDN and requires a request to an external origin server to find the content.

CDNs are typically comprised of a number of different locations that serve content from, so called points of presence (POPs). In some cases, these different POPs are interconnected using the Internet and/or private backbones. Content not found in one POP may be readily available from another POP. Even within a POP, there are typically a number of different edge servers that each fulfill requests for content. These different edge servers have different capabilities and different content in their cache. A cache miss at a particular edge server would be expensive in QoS terms to fulfill from another server or even outside the CDN.

SUMMARY

In one embodiment, one or more content delivery networks (CDNs) deliver content objects for others. Content is propagated to edge servers through hosting and/or caching. End user computers are directed to an edge server for delivery of a requested content object by a universal resource indicator (URI). When a particular edge server does not have a copy of the content object referenced in the URI, information is passed to another server, the ancestor or parent server to find the content object. There can be different parents servers designated for different URIs. The parent server looks for the content object and if not found, will go to another server, the grandparent server, and so on up a hierarchy within the group. Eventually, the topmost server in the hierarchy goes to the origin server to find the content object. The origin server may be hosted in the CDN or at a content provider across the Internet. Once the content object is located in the hierarchical chain, the content object is passed back down the chain to the edge server for delivery. Optionally, the various servers in the chain may cache or host the content object as it is relayed.

Further areas of applicability of the present disclosure will become apparent from the detailed description provided hereinafter. It should be understood that the detailed description and specific examples, while indicating various embodiments, are intended for purposes of illustration only and are not intended to necessarily limit the scope of the disclosure.

BRIEF DESCRIPTION OF THE DRAWINGS

The present disclosure is described in conjunction with the appended figures:

FIG. **1** depicts a block diagram of an embodiment of a content distribution system;

FIG. **2** depicts a block diagram of an embodiment of a content delivery network (CDN);

FIG. **3** depicts a block diagram of an embodiment of a portion of a content delivery network (CDN) that includes a server coupled to a CDN network;

FIGS. **4A**, **4B** and **4C** illustrate flowcharts of embodiments of a process for finding a content object through various hierarchies; and

FIG. **5** depicts a block diagram of an embodiment of a lookup tree.

In the appended figures, similar components and/or features may have the same reference label. Further, various components of the same type may be distinguished by following the reference label by a dash and a second label that distinguishes among the similar components. If only the first reference label is used in the specification, the description is applicable to any one of the similar components having the same first reference label irrespective of the second reference label.

DETAILED DESCRIPTION

The ensuing description provides preferred exemplary embodiment(s) only, and is not intended to limit the scope, applicability or configuration of the disclosure. Rather, the ensuing description of the preferred exemplary embodiment(s) will provide those skilled in the art with an enabling description for implementing a preferred exemplary embodiment. It being understood that various changes may be made in the function and arrangement of elements without departing from the spirit and scope as set forth in the appended claims.

Referring first to FIG. **1**, a block diagram of an embodiment of a content distribution system **100** is shown. The content originator **106** offloads delivery of the content objects to a content delivery network (CDN) **110** in this embodiment. The content originator **106** produces and/or distributes content objects and includes a content provider **108**, a content site **116**, and an origin server **112**. The CDN **110** can both cache and/or host content in various embodiments for third parties to offload delivery and typically provide better quality of service (QoS) to a broad spectrum of end user systems **102** distributed worldwide.

In this embodiment, the content distribution system **100** locates the content objects (or portions thereof) and distributes the content objects to an end user system **102**. The

content objects are dynamically cached within the CDN **110**. A content object is any content file or content stream and could include, for example, video, pictures, data, audio, software, and/or text. The content object could be live, delayed or stored. Throughout the specification, references may be made to a content object, content, content stream and/or content file, but it is to be understood that those terms could be used interchangeably wherever they may appear.

Many content providers **108** use a CDN **110** to deliver the content objects over the Internet **104** to end users **128**. The CDN **110** includes a number of points of presence (POPs) **120**, which are geographically distributed through the content distribution system **100** to deliver content. Various embodiments may have any number of POPs **120** within the CDN **110** that are generally distributed in various locations around the Internet **104** so as to be proximate to end user systems **102**. Multiple POPs use the same IP address such that an Anycast routing scheme is used to find a POP likely to be close to the end user in a network sense for each request. In addition to the Internet **104**, a wide area network (WAN) and/or local area network (LAN) **114** or other backbone may couple the POPs **120** with each other and also couple the POPs **120** with other parts of the CDN **110**.

When an end user **128** requests a web page through its respective end user system **102**, the request for the web page is passed either directly or indirectly via the Internet **104** to the content originator **106**. The content originator **106** is the source or re-distributor of content objects. The content site **116** is an Internet web site accessible by the end user system **102**. In one embodiment, the content site **116** could be a web site where the content is viewable with a web browser. In other embodiments, the content site **116** could be accessible with application software other than a web browser. The content provider **108** directs content requests to a CDN **110** after they are made or formulates the delivery path by embedding the delivery path into the universal resource indicators (URIs) for a web page. In any event, the request for content is handed over to the CDN **110** in this embodiment by using an Anycast IP address corresponding to two or more POPs **120**.

Once the request for a content object is passed to the CDN **110**, the request is associated with a particular POP **120** within the CDN **110** using the Anycast routing scheme. The particular POP **120** may retrieve the portion of the content object from the content provider **108**. Alternatively, the content provider **108** may directly provide the content object to the CDN **110** and its associated POPs **120** through prepopulation, i.e., in advance of the first request. In this embodiment, the content objects are provided to the CDN **110** and stored in one or more CDN servers such that the portion of the requested content may be hosted from the CDN **110**. The CDN servers include edge servers in each POP **120** that actually serve end user requests. The origin server **112** holds a copy of each content object for the content originator **106**. Periodically, the content of the origin server **112** may be reconciled with the CDN **110** through a cache, hosting and/or pre-population algorithm. Some content providers could use an origin server within the CDN **110** to host the content and avoid the need to maintain a copy.

Once the content object is retrieved, the content object is stored within the particular POP **120** and is served from that POP to the end user system **102**. The end user system **102** receives the content object and processes it for use by the end user **128**. The end user system **102** could be a personal computer, media player, handheld computer, Internet appliance, phone, IPTV set top, streaming radio or any other device that receives and plays content objects. In some embodiments, a number of the end user systems **102** could be networked together. Although this embodiment only shows a single content originator **106** and a single CDN **110**, it is to be understood that there could be many of each in various embodiments.

With reference to FIG. **2**, a block diagram of an embodiment of a CDN **110** is shown. Although only one POP **120** is shown in detail, there are a number of POPs **120** similarly configured throughout the CDN **110**. The POPs communicate through a WAN/LAN **114** and/or the Internet **104** when locating content objects. An interface to the Internet **104** to the POP **120** accepts requests for content objects from end user systems **102**. The request comes from an Internet protocol (IP) address in the form of a URI.

Switch fabric **240** assigns the request to one of the edge servers **230** according to a routing scheme such as round robin, load balancing, etc. In this embodiment, the switch fabric is aware of which edge servers **230** have what capabilities and assigns within the group having the capability to store and serve the particular content object referenced in the URI. A protocol such as cache array routing protocol (CARP) is used in this embodiment to disperse the URIs between the group of edge servers **230**. Every time that a particular URI is requested from the group, it is assigned to the same edge server **230** using CARP. The caches gathered in a particular group as neighbors can be the other servers in the current POP, less loaded servers in the current POP, servers having the capability to process the content object, a subset of servers assigned to a customer using the CDN to serve the content object, or some other grouping of servers in the POP **120**.

In another embodiment, the switch fabric **240** assigns the request to one of the edge servers **230**, which performs CARP to either service the request or reassign it to a neighboring edge server **230**. The switch fabric **240** sends each packet flow or request to an edge server **230** listed in the configuration of the switch fabric **240**. This embodiment does not have awareness of the particular capabilities of any edge server **230**. The assignment can be performed by choosing the edge server with the least amount of connections or the fastest response time, but the switch fabric in this embodiment assigns the packet flow somewhat arbitrarily using round robin or random methodologies. When the chosen edge server **230** receives the packet flow, an algorithm like CARP is used by the chosen edge server **230** to potentially reassign the packet flow between a group of edge servers to the one dictated by the algorithm. For example, the switch fabric **240** could choose a second edge server **230-2** being the next in the round robin rotation. The second edge server **230-2** would perform CARP on the request and find that the first edge server **230-1** is being assigned this type of request. The request would be reassigned to the first edge server **230-1** to fulfill.

In some cases, the CDN **110** is used to host content for others. Content providers **108** upload content to a CDN origin server **248**. Although only one CDN origin server **248** is shown, it is to be understood that there could be many spread among a number of locations. The content object can be stored in the CDN origin server **248**. The CDN origin server **248** serves the content object within the CDN **110** to various edge servers **230** in various POPs **120**. After the content provider **108** places a content object on the CDN origin server **248** it need not be hosted on the origin server **112** redundantly.

Requests from end user systems **102** are assigned to an edge server **230** that may cache the requested content object. On occasion, the edge server **230** receiving a request does not have the content object stored for immediate serving. This so-called "cache miss" triggers a process within the CDN **110** to effectively find the content object (or portion thereof) while providing adequate QoS. The content may be found in neigh-

boring edge servers in the same POP **120**, in another POP **120**, in a CDN origin server **248**, or even an external origin server **112**. The various edge and origin servers **230**, **248** are grouped for various URIs uniquely. In other words, one URI may look to one group of servers **230**, **248** on a cache miss while another URI will look to a different group of servers **230**, **248**.

Referring first to FIG. **3**, an embodiment of a portion **300** of a content delivery network (CDN) that includes an edge and/or origin server **304** coupled to the WAN/LAN **114** is shown. The server could be an edge server, a host server or any other server that can supply content objects. It may be at the bottom of a hierarchy or a topmost position of the hierarchy within the CDN. Although this embodiment shows the server **304** as operating as a cache, the content objects could be sticky within the cache such that the server **304** can also act as a host. Indeed, all the content on the server **304** maybe hosted in one embodiment.

Each server **304** in a CDN **110** can belong to any number of groups. The grouping defines where content in the universal resource indicators (URIs) will be searched when not found at a particular server **304**. When the server **304** cannot find a content object that is requested, it will go to the WAN/LAN **114** or another network to find the content object so long as all options within the CDN are not exhausted. The URIs may or may not belong to a group, but when they do, a particular ancestor server will be handed the URI for fulfillment after a cache miss that is potentially different from ancestor servers for other groups. At the top of any hierarchy of lookup tree, a server **304** experiencing a cache miss may go to the Internet rather than the WAN/LAN **114** to obtain a content object from an origin server of the content provider. The server **304** includes a cache engine **316**, a parent group map **324**, and a cache **312**.

The cache engine **316** receives the URI or request for content to fulfill the request by serving the content object to the end user or server down the lookup tree. The cache engine **316** checks the cache **312** for the content object. Where there is a cache miss, the cache engine **316** finds the ancestor server to check for the content object. The cache engine **316** receives the group variable that is derived from the original URI or can derive a tree of ancestor caches from the URI itself.

In one embodiment, a universal resource indicator (URI) is requested and indicates a content object and optionally a group variable. In another embodiment, the group variable is not expressly within the URI, but the URI can be correlated to ancestor caches or groups using a lookup table. Optionally, the URI can also include a path, origin location, variable(s), a prefix, etc. In some form, the URI is passed to various servers in an attempt to find a requested content object. It is to be understood that when the term URI is used, it doesn't necessarily require any format and just conveys at least where to find a content object and the file or stream name. The URI either has the group variable or can be otherwise correlated to parent cache(s) or host. For example, ACME.llnw.net/videos/sports/game.mov?lex5 is a URI with an ACME prefix, a llnw.net domain, a videos/sports path, a game.mov filename, and a lex5 group variable. The URI itself, the ACME prefix and/or lex5 in this example could be used by servers to look-up where to look for a content object when not found locally.

TABLE I

URI Grouping

Prefix	Ancestor Server POP
ACME.llnw.net	San Jose
Smith.llnw.net	Dallas
ShoeExpress.llnw.com	Phoenix
Vinex.llnw.com	San Jose
SDDT.llnw.com	Denver

The URI grouping happens because each server is aware of its ancestor server to use for each URI. The net effect of each server knowing the ancestor server to refer to is to have a hierarchical tree that defines the group. Table I is an example of grouping that could be stored in the parent group map **324**. When a URI containing the ACME prefix is not found in a server, the request is relayed to the San Jose POP **120** where it is assigned to another server for fulfillment.

Grouping can be used to provide different levels of QoS. Table II shows sub-groups for a particular prefix that can be used to specify a sub-group of servers within a POP **120**. For example, the ACME customer designated with the ACME prefix in the URI may offer end user systems **102** three possible levels of QoS. ACME could charge different rates for the various levels of QoS. The Q1 URI variable would specify the fastest servers with the largest caches in the most favorable POP **120**. The Q2 variable would assign a lower caliber of server in the most favorable POP **120**. User systems **102** presenting the Q3 variable would be assigned a less favorable POP when the content object is not found.

TABLE II

URI QoS Sub-Grouping

Prefix	Ancestor Server POP
ACME . . . Q1?	San Jose -Edge Group A
ACME . . . Q2?	San Jose - Edge Group B
ACME . . . Q3?	Denver

Each server looks to the variable from the URI to determine the next ancestor server up the hierarchy to query to when a content object is not located locally. Where there is no ancestor cache for a URI, each server has a default hierarchy to find ancestor caches. There can be any number of possible servers up the hierarchy specified by any number of URIs. In effect, the group variable or URI defines a tree that will specify a flow that ultimately ends in an origin server or host somewhere. These trees can be selected differently for a number of URIs rather than relying on some default lookup tree.

The parent group map **324** stores at least one ancestor server location for each group variable or URI string. If there is no ancestor server specific to the particular group variable or URI, a default ancestor server can be used. In any event, the parent group map **324** returns ancestor server locations or addresses to the cache engine **316**. For example, a parent cache and grandparent cache would be returned for a particular URI or group variable. Should the parent cache not respond for whatever reason, the grandparent cache would be queried. The parent group map **324** can be a database or look-up table that is populated by the CDN to implement a lookup tree.

The cache engine **316** requests the content object from the parent server or grandparent server. Once the ancestor server responds, it will find the content object locally or will look to its ancestor servers. This cycle can repeat many times through

various levels in a hierarchy to ultimately find the content object. The content object is relayed back down the hierarchy to the cache engine **316** that places the content object in the cache **312** as the content object is passed to the end user or down the hierarchy.

Although this embodiment uses a chained approach to finding a server with the content object, other embodiments could use a star approach. In the star approach, the edge server receiving the URI would hold the entire lookup tree in its parent group map **324**. The higher levels in the hierarchy could be successively queried for the content object. Those queries could be done overlapping in time to speed the lookup process. The content object is provided directly from the server higher in the hierarchy without involving every level in the hierarchy. The servers at various levels in the hierarchy could decide to store the content object or not in various embodiments.

With reference to FIG. **4**A, a flowchart of an embodiment of a process **400-1** for finding a content object through various hierarchies is shown. On a URI-by-URI basis, the lookup tree can change. The depicted portion of the process begins in block **404** where the edge server receives a URI. The URI indicates an address that was used to find the edge server, a group variable and information to find the content object along with other information. Initially, the edge server checks its cache **312** for the content object.

In block **412**, it is determined if the content object is available in the edge server **412**. We will cover the scenario where it is not found initially by progressing to block **420**. The group variable is obtained from the URI or it might be passed by a prior server lower in the hierarchy. The ancestor server is determined by referencing the parent group map **324** in block **424** with the group variable value. Although not shown, one or more back-up ancestor servers could be queried if the ancestor server does not respond. In block **428**, the content object is requested from the ancestor server determined in block **424**. Next, processing loops back to block **408** to see if the next higher server in the lookup tree has the content object. These iterations continue until the content object is found in block **412**.

Should the content object be found in the present iteration or in the higher levels in the hierarchy of the lookup tree in block **412**, processing continues to block **432**. In the simple case, the edge server has the content object already before ever going through the loop. Where that is not the case, the content object is relayed down through the hierarchy from the server that had the content object to the edge server in block **432**. Each server in that chain may cache or otherwise store the content object. In block **436**, the edge server that originally received the request for the content object serves the content object through a stream or download to an end user.

With reference to FIG. **4**B, a flowchart of another embodiment of a process **400-2** for finding a content object through various hierarchies is shown. The depicted portion of the process **400-2** begins in block **402** where a URI request is received an edge server at the POP that specifies a content object. The URI is rewritten in block **406**. There may be many different versions of a URI that correspond to a single content object. A look-up table is used to match as much of the URI as possible to an authoritative name or source URL. The caches store content objects based upon the source URL, which points to an origin server that can be used to retrieve a content object not in the CDN.

It is determined in block **408** if the edge server receiving the request has the content object available locally. In block **412**, a determination is made if the content object is available and processing continues to block **432** and **436** if the content object is within the edge server cache in the same manner as the embodiment of FIG. **4**A.

Should the content object not be in the cache of the edge server as determined in block **412**, processing continues to block **416**. During the rewrite process of block **406**, many parameters such as the ancestor cache(s) are retrieved for the source URL and retrieved for use in block **416**. The ancestor cache(s) is the start of a potentially iterative process that defines the tree of the cache group. In block **422**, ancestor caches are found using the parameters associated with the source URL. As each source URLs could have different ancestor caches, different cache groups form on a URI-by-URI basis. The cache group is a function of the POP receiving the request and ancestor cache preferences for each source URI.

The ancestor cache for a particular URI may be chosen for any number of reasons. Which ancestors are used may adjust on a server, POP-wide or CDN-wide basis with periodic (e.g., hourly, daily, weekly, monthly, or some other period) or real-time updates that react to health and loading of particular servers and POPs. An ancestor may be chosen based upon whether the content is hosted and/or cached in the CDN, the capability of a server to stream or process different content objects with different media types, loading of a server, a server going down, a server having health problems, loading of network connections of a server, or other issues that would affect the suitability of a particular ancestor server temporarily or permanently. For example, a data connection between a cache and an ancestor cache may be overloaded and the ancestor cache would change to one that was suitable.

A table or database stores ancestor cache information for each source URL or group of source URLs. Typically, there is a primary ancestor and at least one back-up should the primary not respond for whatever reason. In block **428**, the source URI is requested of the primary ancestor cache and any back-up ancestor caches, if necessary. Processing then loops back to block **408** to repeat block **412**, **416**, **422**, and **428** in a loop to work through the hierarchy. Although not shown, the highest level in the hierarchy would refer to the origin server to retrieve the content object.

With reference to FIG. **4**C, a flowchart of another embodiment of a process **400-3** for finding a content object through various hierarchies is shown. This embodiment is similar to the embodiment of FIG. **4**B, but adds a block **414** between blocks **412** and **416**. In block **414**, neighboring servers are searched for the content object before resorting to an ancestor cache out side those neighbors. The neighboring servers could be a group of the entire POP **120** or a sub-group of the servers within the POP **120**. The serving of the content object may be reassigned to the server holding the content object or may be relayed or proxied to the server currently assigned to serve the content in various embodiments.

Referring to FIG. **5**, a block diagram of an embodiment of a lookup tree **500** is shown. This lookup tree **500** is simplified as there could be hundreds or thousands of blocks on the lookup tree for a CDN. Embodiments could have different lookup trees **500** for different customers, content formats, digital rights management, delivery methods, end user Internet service providers, bitrates, loading levels, etc. This embodiment shows three levels of hierarchy within the CDN **520** prior to requesting a content object an external origin server **516**. By loading the parent group maps **324** for all the servers in the lookup tree **500** will organize the hierarchy for a particular group variable.

In the first level of the hierarchy for the lookup tree **500**, there are five edge servers **504** that cache content objects. The

edge servers **504** could be distributed around the Internet in different ways. For example, edge cache A and B **504-1**, **504-2** could be in the same POP or in different POPs. When a request goes edge cache A **504-1** and cannot be fulfilled internally, edge cache B **504-2** is checked next for the content object. Should the content object not be found in edge cache B **504-2**, the request would go to another POP in Los Angeles **508-1**. One or all the caches in the Los Angeles POP **508-1** would be queried for the content object.

Should the Los Angeles POP **508-1** not have the content object, it would query the POP having the San Jose POP **512**. If not found in the San Jose POP **512**, a server of the San Jose POP **512** would go back to the origin server **516** to retrieve the content object. In another embodiment, the CDN **520** hosts the content objects to serve as the origin server **516**.

In another example, the content object request starts with edge cache D **504-4**. If not found there, a request would be made by edge cache D **504-4** to edge cache C **504-3**, but not to edge cache E. For example, edge cache E may be in a different location or not suited to store and serve the content object. Should the content object not be found in edge cache C, the Denver POP **508-8** would be queried by edge cache C. If not found, the next request goes from the Denver POP **508-8** to the San Jose POP **512**. Finally, the origin server **516** would be queried if the content object is not found in the San Jose POP **512**.

In some cases, the search for a server with the content object can get caught in a loop. A misconfiguration in the parent group maps **324** could cause this problem perhaps as a update to the lookup tree **500** is partially rolled out. When a server receives a request for the content object from another server, the server checks to see if in the chain of servers relaying the request the receiving server is already listed. This would indicate a loop had developed. The loop could be broken by referring the request to another POP or even to the origin server, for example.

Specific details are given in the above description to provide a thorough understanding of the embodiments. However, it is understood that the embodiments may be practiced without these specific details. For example, circuits may be shown in block diagrams in order not to obscure the embodiments in unnecessary detail. In other instances, well-known circuits, processes, algorithms, structures, and techniques may be shown without unnecessary detail in order to avoid obscuring the embodiments.

Also, it is noted that the embodiments may be described as a process which is depicted as a flowchart, a flow diagram, a data flow diagram, a structure diagram, or a block diagram. Although a flowchart may describe the operations as a sequential process, many of the operations can be performed in parallel or concurrently. In addition, the order of the operations may be re-arranged. A process is terminated when its operations are completed, but could have additional steps not included in the figure. A process may correspond to a method, a function, a procedure, a subroutine, a subprogram, etc. When a process corresponds to a function, its termination corresponds to a return of the function to the calling function or the main function.

Furthermore, embodiments may be implemented by hardware, software, scripting languages, firmware, middleware, microcode, hardware description languages, and/or any combination thereof. When implemented in software, firmware, middleware, scripting language, and/or microcode, the program code or code segments to perform the necessary tasks may be stored in a machine readable medium such as a storage medium. A code segment or machine-executable instruction may represent a procedure, a function, a subprogram, a program, a routine, a subroutine, a module, a software package, a script, a class, or any combination of instructions, data structures, and/or program statements. A code segment may be coupled to another code segment or a hardware circuit by passing and/or receiving information, data, arguments, parameters, and/or memory contents. Information, arguments, parameters, data, etc. may be passed, forwarded, or transmitted via any suitable means including memory sharing, message passing, token passing, network transmission, etc.

While the principles of the disclosure have been described above in connection with specific apparatuses and methods, it is to be clearly understood that this description is made only by way of example and not as limitation on the scope of the disclosure.

What is claimed is:

1. A method to retrieve content objects in a content delivery network (CDN) having a plurality of points of presence (POPs) distributed geographically by checking one or more neighboring servers before checking a parent server, the method comprising:

receiving a first universal resource identifier (URI) at a first edge server having a first cache wherein:

the first edge server is in a first POP of the plurality of POPs, and

the first URI specifies a first content object;

determining that the first cache does not hold the first content object;

querying the one or more neighboring servers for the first content object, wherein the one or more neighboring servers each have a cache;

determining that one or more caches of the one or more neighboring servers stores the first content object;

serving the first content object from the one or more caches of the one or more neighboring servers;

receiving a second URI at the first edge server, wherein the second URI specifies a second content object;

determining that the first cache does not hold the second content object;

querying the one or more neighboring servers for the second content object;

determining that the one or more caches of the one or more neighboring servers do not hold the second content object;

analyzing the second URI to determine the parent server, and

requesting the second content object from the parent server after determining that the one or more caches of the one or more neighboring servers do not hold the second content object.

2. The method to retrieve content objects in the CDN having the plurality of POPs distributed geographically as recited in claim **1**, wherein the one or more caches of the one or more neighboring servers are located within the first POP.

3. The method to retrieve content objects in the CDN having the plurality of POPs distributed geographically as recited in claim **2**, wherein the one or more caches of the one or more neighboring servers comprise each cache, for storing content objects, within the first POP.

4. The method to retrieve content objects in the CDN having the plurality of POPs distributed geographically as recited in claim **1**, wherein:

the parent server is in a second POP of the plurality of POPs, and

the second POP is different from the first POP.

5. The method to retrieve content objects in the CDN having the plurality of POPs distributed geographically as

recited in claim 1, wherein the serving the first content object from the one or more caches of the one or more neighboring servers comprises serving the first content object to an end-user system from the first edge server acting as a proxy for the one or more neighboring servers.

6. The method to retrieve content objects in the CDN having the plurality of POPs distributed geographically as recited in claim 1, wherein serving of the first content object is reassigned from the first edge server to the one or more neighboring servers.

7. The method to retrieve content objects in the CDN having the plurality of POPs distributed geographically as recited in claim 1, wherein querying one or more neighboring servers for the second content object comprises:

querying a second edge server; and

querying a third edge server, wherein the querying the second edge server and the querying the third edge server are performed overlapping in time.

8. A content delivery network (CDN) for retrieving content objects from one or more neighboring servers before requesting content objects from a parent server, the CDN comprising:

a plurality of POPs distributed geographically;

a first edge server having a first cache, wherein:

the first edge server is located within a first POP of the plurality of POPs,

the first edge server receives a first universal resource identifier (URI),

the first URI specifies a first content object,

the first edge server determines that the first cache does not hold the first content object,

the first edge server queries one or more neighboring servers for the first content object, wherein the one or more neighboring servers each have a cache,

the first edge server receives an indication that the first content object is stored in one or more caches of the one or more neighboring servers,

the first edge server receives a second URI,

the second URI specifies a second content object,

the first edge server determines that the first cache does not hold the second content object,

the first edge server determines that the one or more caches of the one or more neighboring servers do not store the second content object, and

the first edge server requests the second content object from the parent server after determining that the one or more caches of the one or more neighboring servers do not hold the second content object; and

a second POP, wherein:

the second POP is different from the first POP, and

the second POP comprises the parent server.

9. The CDN for retrieving content objects from the one or more neighboring servers before requesting content objects from the parent server, as recited in claim 8, wherein the one or more neighboring servers are located within the first POP.

10. The CDN for retrieving content objects from the one or more neighboring servers before requesting content objects from the parent server, as recited in claim 8, wherein an end user is served the first content object from the one or more caches of the one or more neighboring servers by the first edge server acting as a proxy for the one or more neighboring servers.

11. The CDN for retrieving content objects from the one or more neighboring servers before requesting content objects from the parent server, as recited in claim 8, the CDN further comprising the one or more neighboring servers, wherein serving the first content object to an end-user system is reassigned from the first edge server to the one or more neighboring servers.

12. The CDN for retrieving content objects from the one or more neighboring servers before requesting content objects from the parent server, as recited in claim 8, the CDN further comprising:

a first neighboring server, wherein:

the first neighboring server is part of the one or more neighboring servers, and

the first edge server makes a first query of the first neighboring server for the first content object; and

a second neighboring server, wherein:

the second neighboring server is part of the one or more neighboring servers,

the first edge server makes a second query of the second neighboring server for the first content object, and

the first query and the second query overlap in time.

13. The CDN for retrieving content objects from the one or more neighboring servers before requesting content objects from the parent server, as recited in claim 8, wherein the first URI and the second URI correspond to different content objects.

14. The CDN for retrieving content objects from the one or more neighboring servers before requesting content objects from the parent server, as recited in claim 8, wherein the first URI is a portion of a third URI.

15. A content delivery network (CDN) having a plurality of points of presence (POPs) for retrieving content objects, the CDN comprising one or more servers programmed for:

receiving a first universal resource identifier (URI) at a first server having a first cache, wherein:

the first server is in a first POP of the plurality of POPs, and

the first URI specifies a first content object;

determining that the first cache does not hold the first content object;

querying one or more neighboring servers for the first content object, wherein the one or more neighboring servers each have a cache;

determining that one or more caches of the one or more neighboring servers stores the first content object;

serving the first content object from the one or more caches of the one or more neighboring servers;

receiving a second URI at the first server, wherein the second URI specifies a second content object;

determining that the first cache does not hold the second content object;

determining that the one or more caches of the one or more neighboring servers do not hold the second content object;

analyzing the second URI to determine a cache hierarchy for the second URI, and

requesting the second content object from the cache hierarchy after determining that the one or more caches of the one or more neighboring servers do not hold the second content object.

16. The CDN having the plurality of POPs for retrieving content objects as recited in claim 15, wherein the one or more neighboring servers are in the first POP.

17. The CDN having the plurality of POPs for retrieving content objects as recited in claim 15, wherein:

the cache hierarchy comprises a cache in a second POP; and

the second POP is different from the first POP.

18. The CDN having the plurality of POPs for retrieving content objects as recited in claim 15, wherein an end user is

served the first content object from the one or more caches of the one or more neighboring servers by the first server acting as a proxy for the one or more neighboring servers.

19. The CDN having the plurality of POPs for retrieving content objects as recited in claim **15**, wherein querying one or more neighboring servers for the first content object comprises:

querying a second server; and

querying a third server, wherein the querying the second server and the querying the third server is performed overlapping in time.

20. The CDN having the plurality of POPs for retrieving content objects as recited in claim **15**, wherein the first content object is not the same as the second content object.

* * * * *

Exhibit E

US009015348B2

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

(10) **Patent No.: US 9,015,348 B2**
(45) **Date of Patent: Apr. 21, 2015**

(54) **DYNAMICALLY SELECTING BETWEEN ACCELERATION TECHNIQUES BASED ON CONTENT REQUEST ATTRIBUTES**

(71) Applicant: **Limelight Networks, Inc.**, Tempe, AZ (US)

(72) Inventors: **Jason Hofmann**, New York, NY (US); **Hemdat Cohen-Shraga**, Elkana (IL); **Erez Yaffe**, Tel Aviv (IL)

(73) Assignee: **Limelight Networks, Inc.**, Tempe, AZ (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 42 days.

(21) Appl. No.: **13/946,777**

(22) Filed: **Jul. 19, 2013**

(65) **Prior Publication Data**

US 2015/0026239 A1 Jan. 22, 2015

(51) **Int. Cl.**
G06F 15/173 (2006.01)
H04L 29/06 (2006.01)

(52) **U.S. Cl.**
CPC ***H04L 29/06047*** (2013.01)

(58) **Field of Classification Search**
None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,802,292	A	9/1998	Mogul
5,826,031	A	10/1998	Nielsen
5,884,098	A	3/1999	Mason, Jr.
5,894,554	A	4/1999	Lowery et al.
6,023,726	A	2/2000	Saksena
6,055,572	A	4/2000	Saksena
6,067,565	A	5/2000	Horvitz
6,085,226	A	7/2000	Horvitz
6,098,064	A	8/2000	Pirolli et al.
6,272,534	B1	8/2001	Guha
6,338,096	B1	1/2002	Ukelson
6,385,641	B1	5/2002	Jiang et al.
6,553,393	B1	4/2003	Eilbott et al.
6,578,073	B1	6/2003	Starnes et al.
6,654,807	B2	11/2003	Farber et al.

(Continued)

FOREIGN PATENT DOCUMENTS

KR	20030029244	A	4/2003

OTHER PUBLICATIONS

Bartolini, "A Walk through Content Delivery Networks", Retrieved on Jul. 4, 2012 from http://wwwusers.di.uniroma1.it/~novella/articoli/CDN_tutorial.pdf, 2012, p. 1-25.

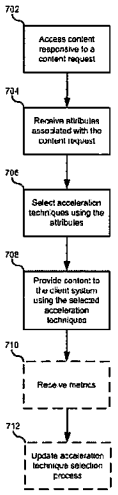
(Continued)

Primary Examiner — Scott B Christensen
(74) *Attorney, Agent, or Firm* — Kilpatrick Townsend & Stockton, LLP

(57) **ABSTRACT**

A system for dynamically selecting from among a plurality of acceleration techniques implemented in a Content Delivery Network (CDN) using attributes associated with content requests may include a network interface that receives a content request from a client system for content, where the request is associated with one or more attributes. The system may also include an intermediate server that accelerates access to the content stored in the CDN edge servers. The intermediate server may include a processor configured to access the one or more attributes associated with the content request, select one or more acceleration techniques from the plurality of acceleration techniques where the one or more acceleration techniques are selected based on the one or more attributes, and use the one or more acceleration techniques to provide the content to the client system.

19 Claims, 13 Drawing Sheets



(56) **References Cited**

U.S. PATENT DOCUMENTS

6,665,658	B1	12/2003	DaCosta et al.
6,799,214	B1	9/2004	Li
6,834,297	B1	12/2004	Peiffer et al.
6,976,090	B2	12/2005	Ben-Shaul et al.
6,993,591	B1	1/2006	Klemm
7,007,237	B1	2/2006	Sharpe
7,012,612	B1	3/2006	O'Neill et al.
7,084,877	B1	8/2006	Panusopone et al.
7,100,111	B2	8/2006	McElfresh et al.
7,107,338	B1	9/2006	Nareddy et al.
7,113,935	B2	9/2006	Saxena
7,240,100	B1	7/2007	Wein et al.
7,243,309	B2	7/2007	Koay et al.
7,249,196	B1	7/2007	Peiffer et al.
7,308,490	B2	12/2007	Peiffer et al.
7,337,248	B1	2/2008	Rao et al.
7,373,599	B2	5/2008	McElfresh et al.
7,483,941	B2	1/2009	Carlson et al.
7,594,003	B2	9/2009	Davidson et al.
7,594,013	B2	9/2009	Wang et al.
7,636,770	B2	12/2009	Bennett et al.
7,689,663	B2	3/2010	Kinnan et al.
7,814,172	B2	10/2010	Martin et al.
7,818,686	B2	10/2010	Cooke
7,860,881	B2	12/2010	Haselden et al.
7,886,218	B2	2/2011	Watson
7,941,483	B2	5/2011	Narayanan et al.
7,958,232	B1	6/2011	Colton et al.
8,028,090	B2	9/2011	Richardson et al.
8,060,518	B2	11/2011	Timmons
8,112,703	B2	2/2012	Kumar et al.
8,122,102	B2	2/2012	Wein et al.
8,166,079	B2	4/2012	Lewin et al.
8,219,633	B2	7/2012	Fainberg et al.
8,219,647	B2	7/2012	Harvell et al.
8,250,457	B2	8/2012	Fainberg et al.
8,346,784	B1	1/2013	Potekhin et al.
8,495,171	B1	7/2013	Potekhin et al.
2002/0063714	A1	5/2002	Haas et al.
2002/0078087	A1	6/2002	Stone
2002/0078165	A1	6/2002	Genty et al.
2003/0101412	A1	5/2003	Eid
2004/0030717	A1	2/2004	Caplin
2004/0088375	A1	5/2004	Sethi et al.
2004/0215665	A1	10/2004	Edgar et al.
2004/0225562	A1	11/2004	Turner
2005/0044491	A1	2/2005	Peterson
2005/0138143	A1	6/2005	Thompson
2005/0154781	A1	7/2005	Carlson et al.
2005/0198191	A1	9/2005	Carlson
2006/0093030	A1	5/2006	Francois et al.
2006/0218305	A1	9/2006	Kinnan et al.
2007/0022102	A1	1/2007	Saxena
2007/0156845	A1	7/2007	Devanneaux et al.
2007/0162434	A1	7/2007	Alessi et al.
2007/0208610	A1	9/2007	Pisaris-Henderson et al.
2007/0256003	A1	11/2007	Wagoner et al.
2007/0260748	A1	11/2007	Talkington
2008/0005672	A1	1/2008	Mestres et al.
2008/0071859	A1	3/2008	Seed et al.
2008/0139191	A1	6/2008	Melnyk et al.
2008/0155425	A1	6/2008	Murthy et al.
2008/0195712	A1	8/2008	Lin et al.
2008/0228772	A1	9/2008	Plamondon
2008/0228911	A1	9/2008	Mackey
2009/0037454	A1	2/2009	Sampson et al.
2009/0125481	A1	5/2009	Mendes da Costa et al.
2009/0240698	A1	9/2009	Shukla et al.
2009/0300111	A1	12/2009	Rana
2010/0017696	A1	1/2010	Choudhary et al.
2010/0149091	A1	6/2010	Kota et al.
2010/0169455	A1	7/2010	Gorham
2010/0269050	A1	10/2010	Kirkby et al.
2010/0281357	A1	11/2010	Fu et al.
2010/0299589	A1	11/2010	Yamada
2010/0306643	A1	12/2010	Chabot et al.
2011/0016180	A1	1/2011	Bharadhwaj et al.
2011/0029899	A1	2/2011	Fainberg et al.
2011/0066676	A1	3/2011	Kleyzit et al.
2011/0087966	A1	4/2011	Leviathan et al.
2011/0113000	A1	5/2011	Marlow
2011/0276446	A1	11/2011	Gupta et al.
2011/0289486	A1	11/2011	Revinskaya et al.
2011/0302321	A1	12/2011	Vange et al.
2012/0030224	A1	2/2012	Cohen et al.
2012/0054595	A1	3/2012	Mylroie et al.
2012/0079057	A1	3/2012	Fainberg et al.
2012/0185370	A1	7/2012	Davie et al.
2012/0198022	A1	8/2012	Black et al.
2013/0227078	A1*	8/2013	Wei et al. 709/219
2013/0326022	A1	12/2013	Ehrlich et al.

OTHER PUBLICATIONS

Bogdanov et al., "A prototype of online privacy-preserving questionnaire system," 2010, 6 pages.

Chen, G., et al., "Building a Scalable Web Server with Global Object Space Support on Heterogeneous Clusters," Third IEEE International Conference on Cluster Computing (CLUSTER'01), 2001, 8 pages.

Gardner et al., "DOM: Towards a Formal Specification," Plan-X ' 08, Jan. 9, 2008, San Francisco, 2008, 10 pages.

Jansen, "Guidelines on Active Content and Mobile Code", Retrieved on Jul. 4, 2012 from http://csrc.nist.gov/publications/nistpubs/800-28-ver2/SP800-28v2.pdf, NIST Special Publication 800-28 Version 2, Mar. 2008, p. 1-62.

Jing, J., et al., "Client Server Computing in Mobile Environments," ACM Computing Surveys, Jun. 1999, pp. 117-157, vol. 31, No. 2.

Mor, "Evaluation of Delivery Techniques for Dynamic Web Content", Retrieved on Jul. 2, 2012, http://infolab.stanford.edu/~mor/research/eodposter.pdf, pp. 1-2.

Pallis et al., "Insight and Perspectives for Content Delivery Network", Retrieved on Sep. 5, 2012 from http://gridsec.usc.edu/files/EE657/P2P-CDNetworks-Survey-2006.pdf, Jan. 2006, vol. 49, p. 1-6.

Rayburn, "How Dynamic Site Acceleration Works, What Akamai and Cotendo Offer", Retrieved on Jul. 4, 2012 from http://blog.streamingmedia.com/the_business_of_online_vi/2010/10/how-dynamic-site-acceleration-works-what-akamai-and-cotendo-offer.html, Oct. 18, 2010, p. 1-4.

Seifert et al., "Identification of Malicious Web Pages with Static Heuristics," IEEE, 2008, pp. 91-96.

U.S. Appl. No. 13/245,841, Office Action mailed Mar. 5, 2012, 17 pages.

U.S. Appl. No. 13/245,841, Notice of Allowance mailed Jul. 2, 2012, 4 pages.

U.S. Appl. No. 13/245,711, Notice of Allowance mailed Apr. 24, 2012, 10 pages.

U.S. Appl. No. 12/848,559, Office Action mailed Aug. 3, 2012, 26 pages.

U.S. Appl. No. 13/471,211, Office Action mailed Aug. 8, 2012, 14 pages.

Wikipedia, "Web accelerator", Retrieved on Mar. 14. 2012, from http://en.wikipedia.org/wiki/Web_accelerator, Jan. 11, 2012, p. 1-3.

* cited by examiner

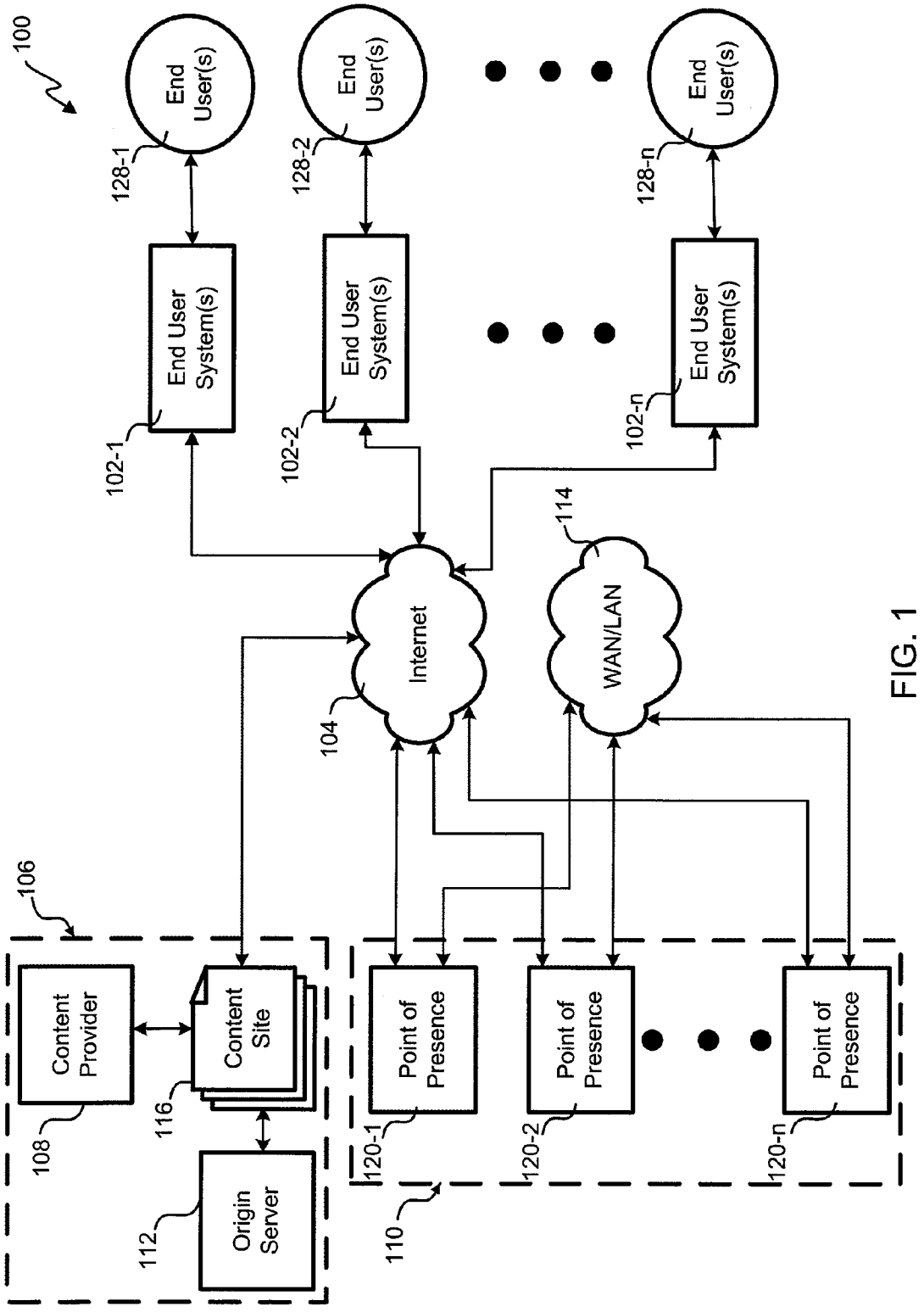

100
End User(s)
128-1
End User(s)
128-2
End User(s)
128-n
End User System(s)
102-1
End User System(s)
102-2
End User System(s)
102-n
Internet
104
WAN/LAN
114
106
Content Provider
108
Content Site
116
Origin Server
112
Point of Presence
120-1
Point of Presence
120-2
Point of Presence
120-n
110

FIG. 1

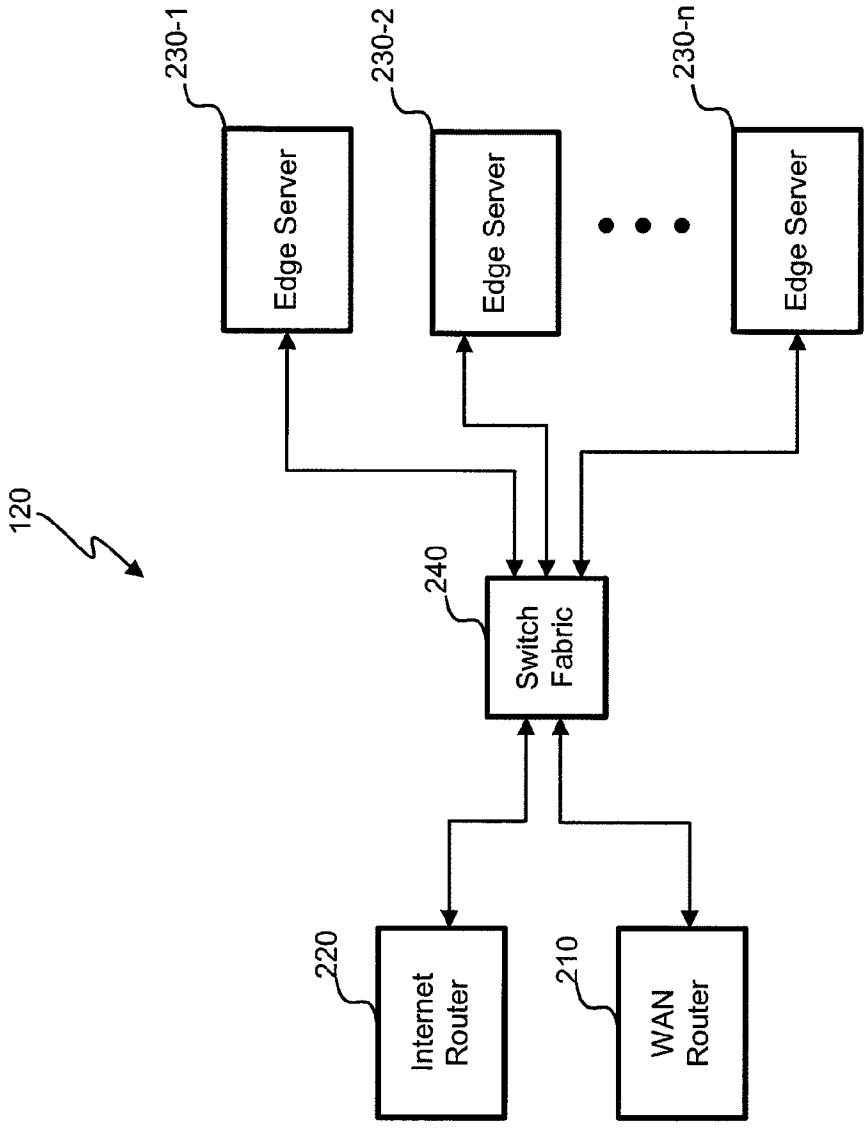

120
230-1
Edge Server
230-2
Edge Server
230-n
Edge Server
240
Switch Fabric
220
Internet Router
210
WAN Router

FIG. 2

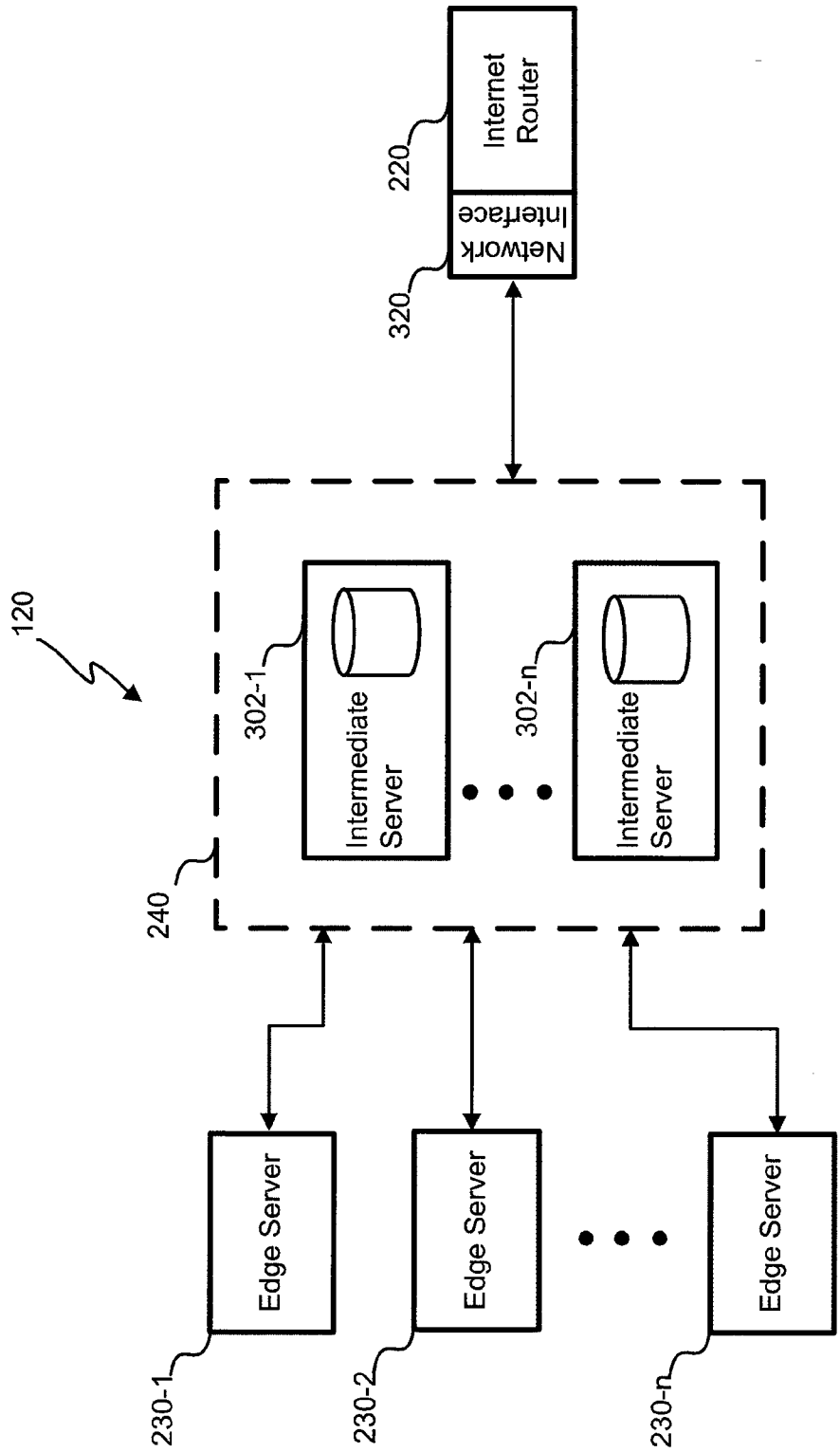

120
240
302-1
Intermediate Server
302-n
Intermediate Server
320
Network Interface
220
Internet Router
230-1
Edge Server
230-2
Edge Server
230-n
Edge Server

FIG. 3A

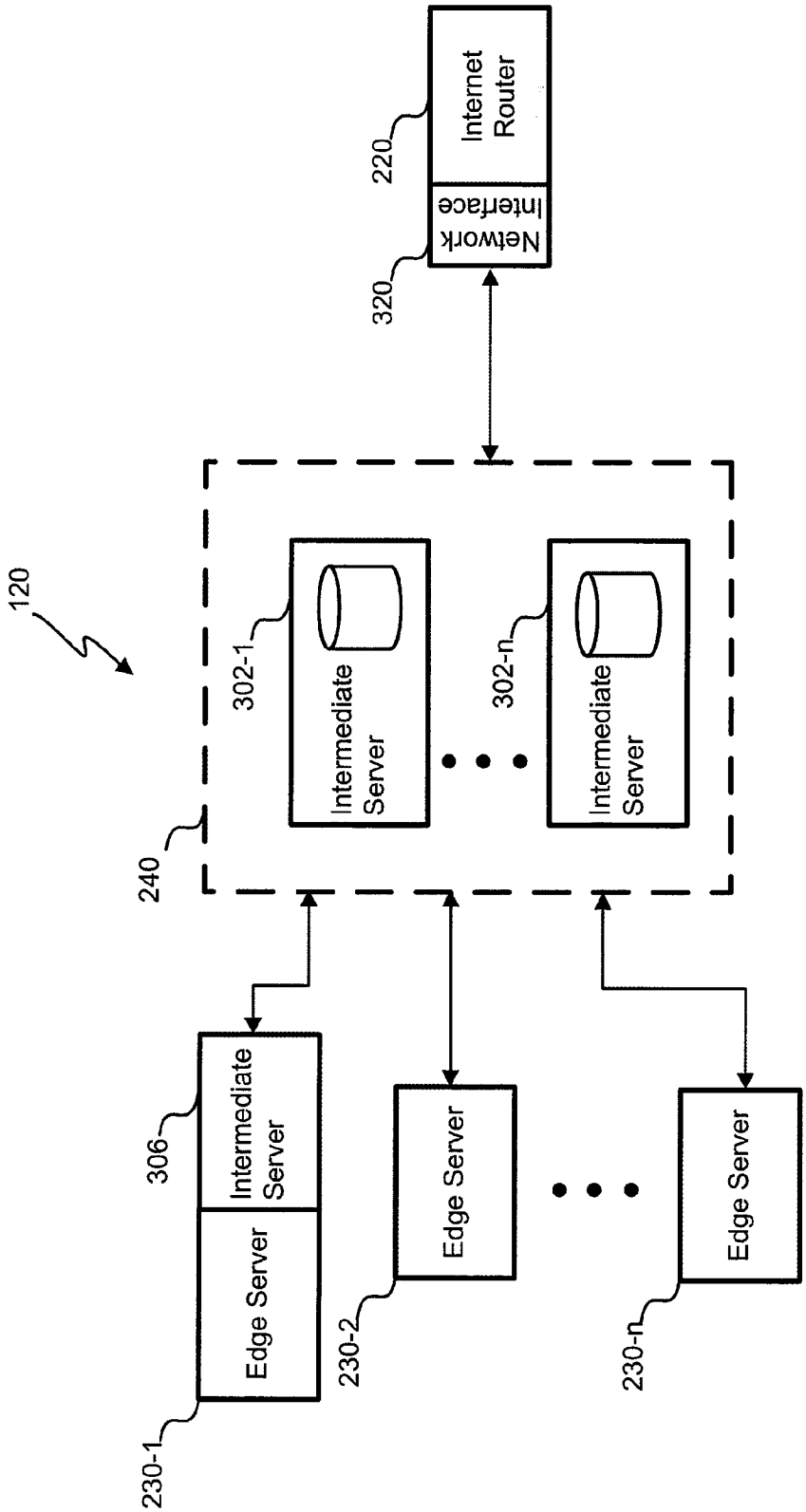

120
240
Internet Router
220
Network Interface
320
Intermediate Server
302-1
Intermediate Server
302-n
Intermediate Server
306
Edge Server
230-1
Edge Server
230-2
Edge Server
230-n

FIG. 3B

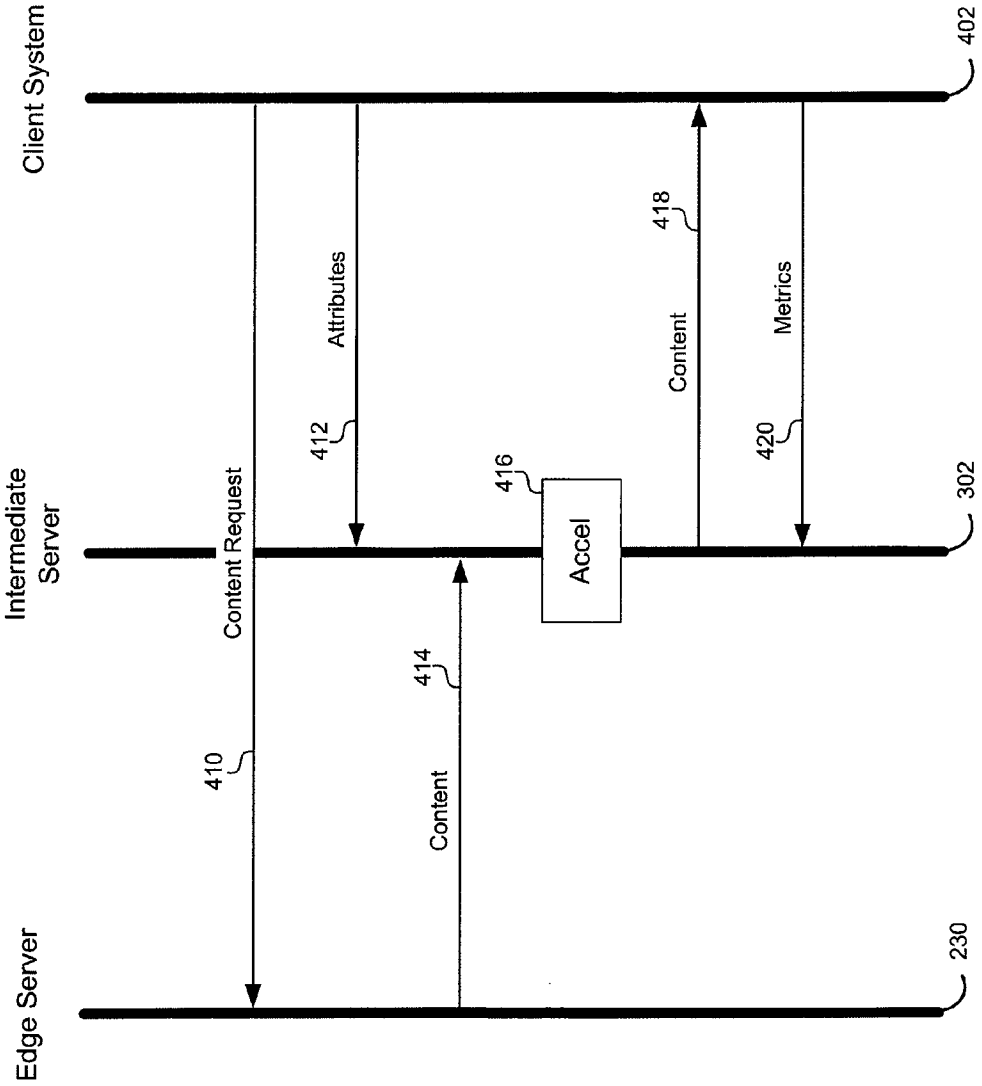

Client System
402
Attributes
412
418
Content
Metrics
420
Intermediate Server
416
Accel
302
Content Request
414
410
Content
Edge Server
230

FIG. 4A

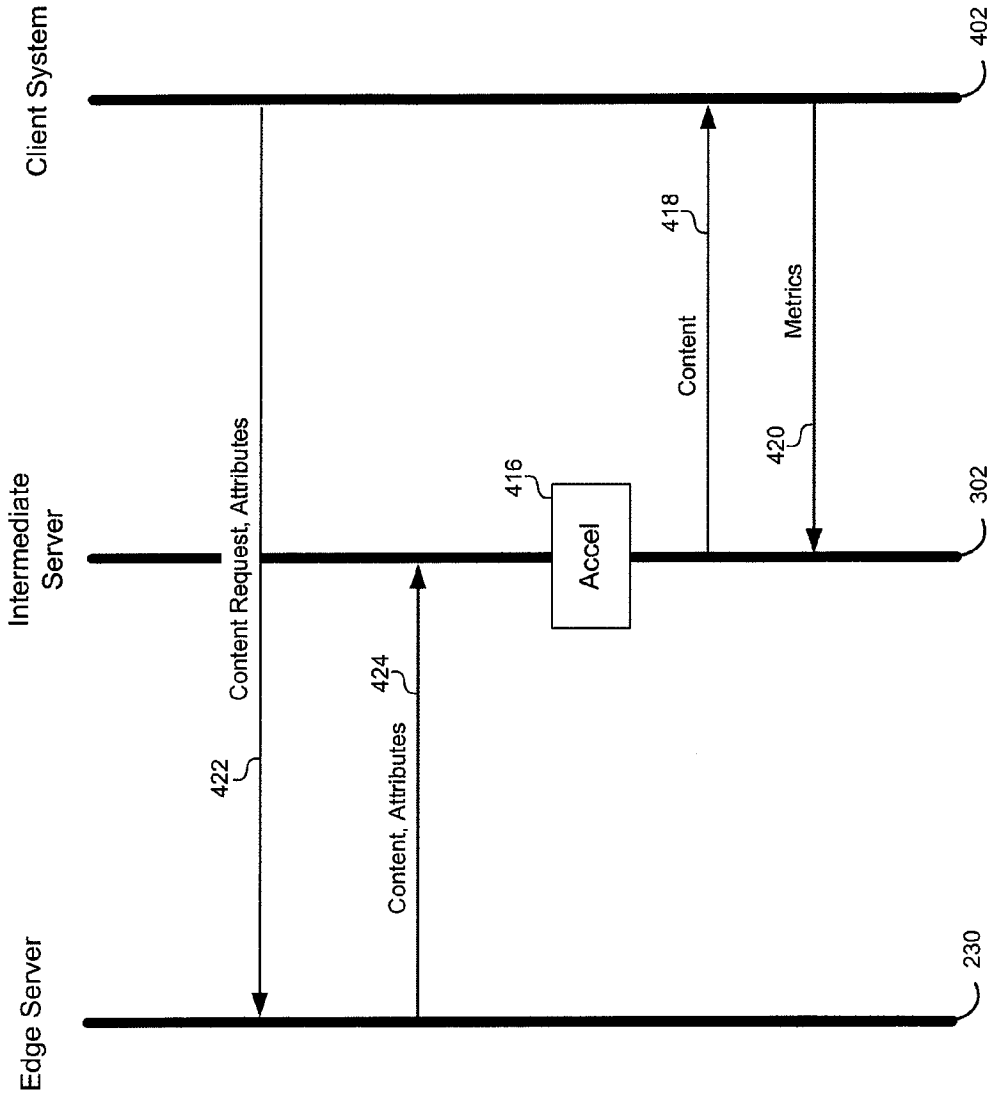

Client System
402
418
Content
Metrics
420
416
Accel
Intermediate Server
302
Content Request, Attributes
422
424
Content, Attributes
Edge Server
230

FIG. 4B

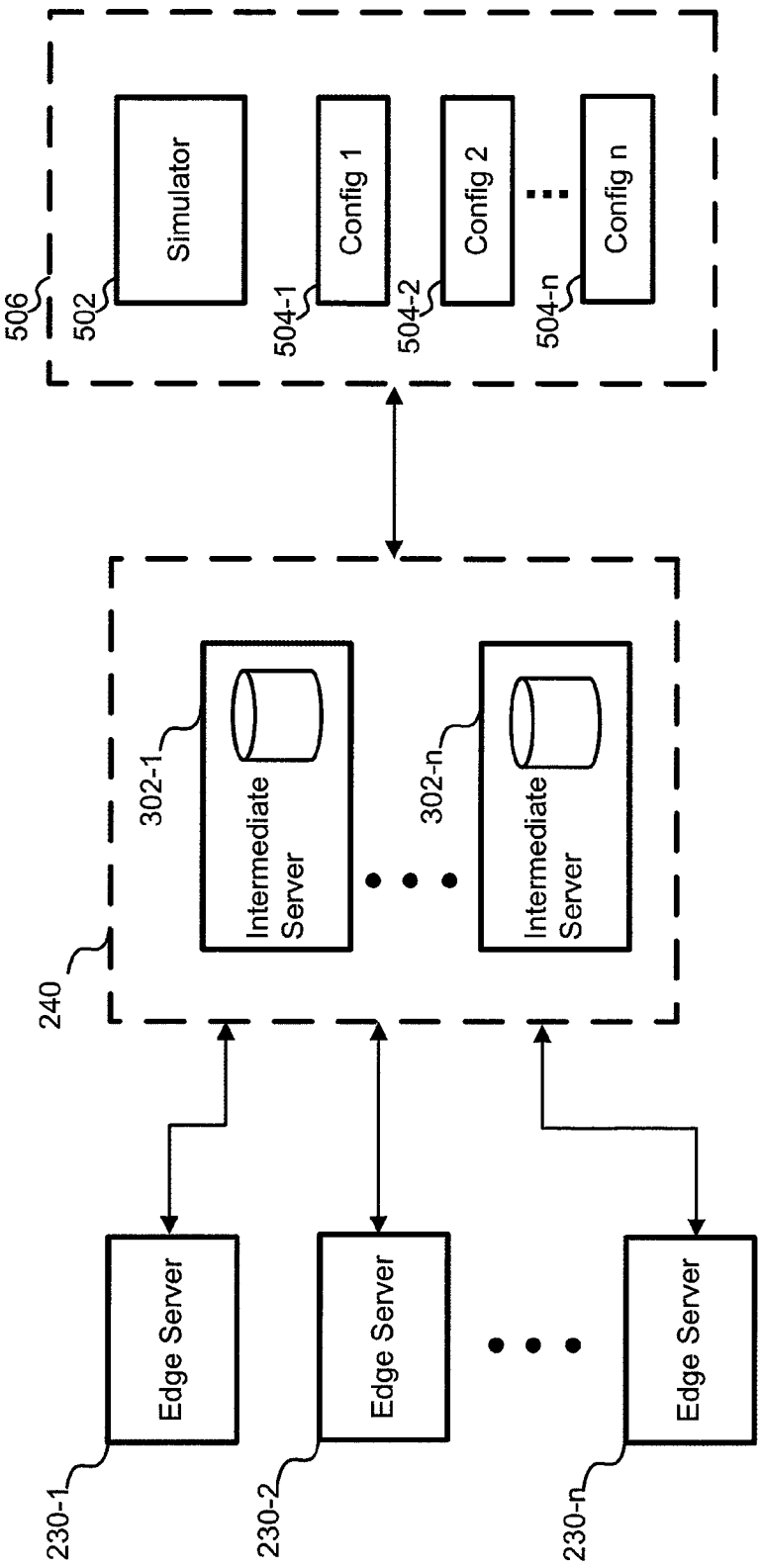

506
502
Simulator
504-1
Config 1
504-2
Config 2
504-n
Config n
240
302-1
Intermediate Server
302-n
Intermediate Server
230-1
Edge Server
230-2
Edge Server
230-n
Edge Server

FIG. 5

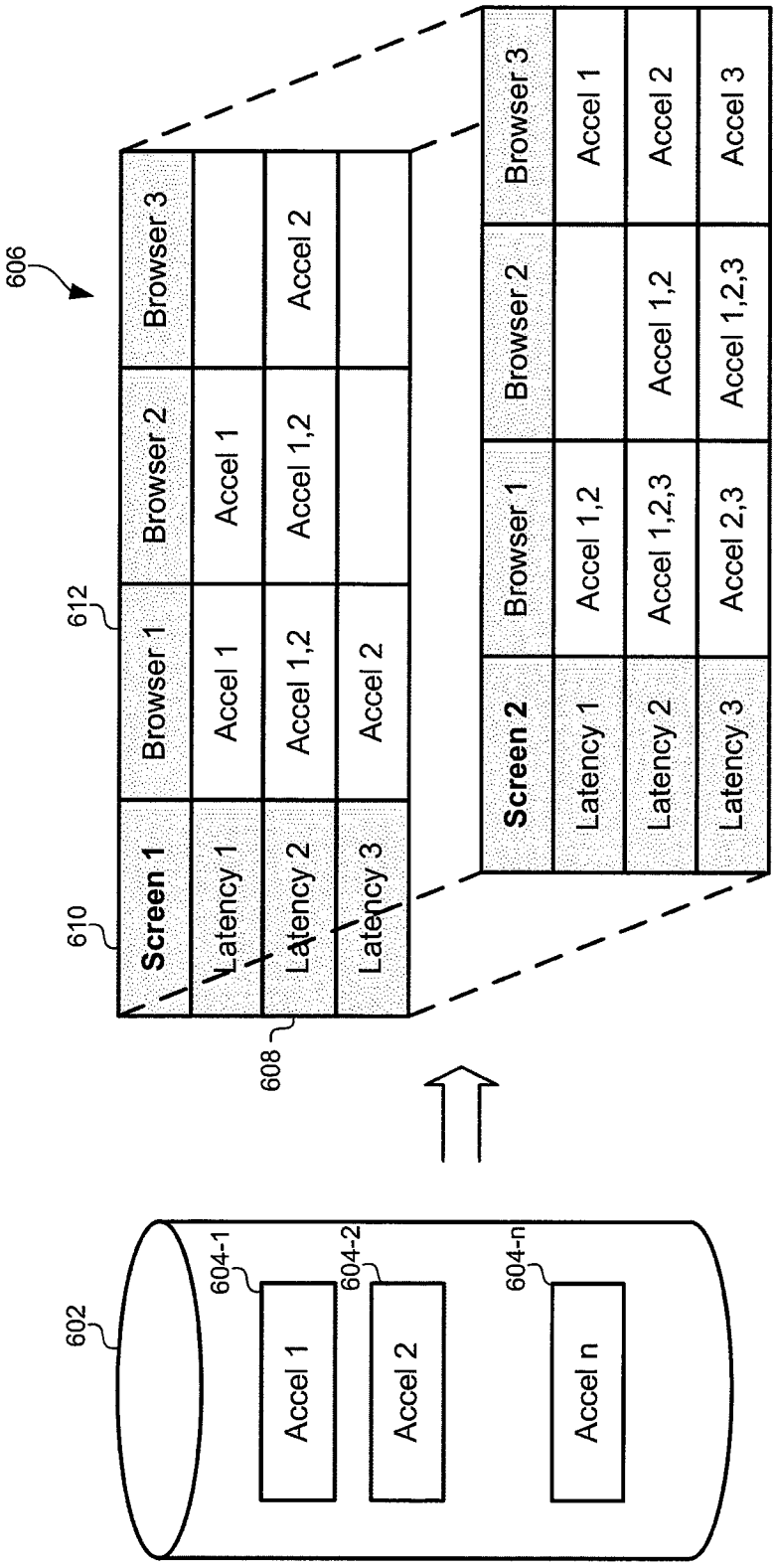

606
612
610
608
602
604-1
604-2
604-n
Screen 1
Browser 1
Browser 2
Browser 3
Latency 1
Latency 2
Latency 3
Accel 1
Accel 1,2
Accel 2
Screen 2
Browser 1
Browser 2
Browser 3
Latency 1
Latency 2
Latency 3
Accel 1,2
Accel 1,2,3
Accel 2,3
Accel 1
Accel 2
Accel 1,2
Accel 1,2,3
Accel 3
Accel 1
Accel 2
Accel n

FIG. 6A

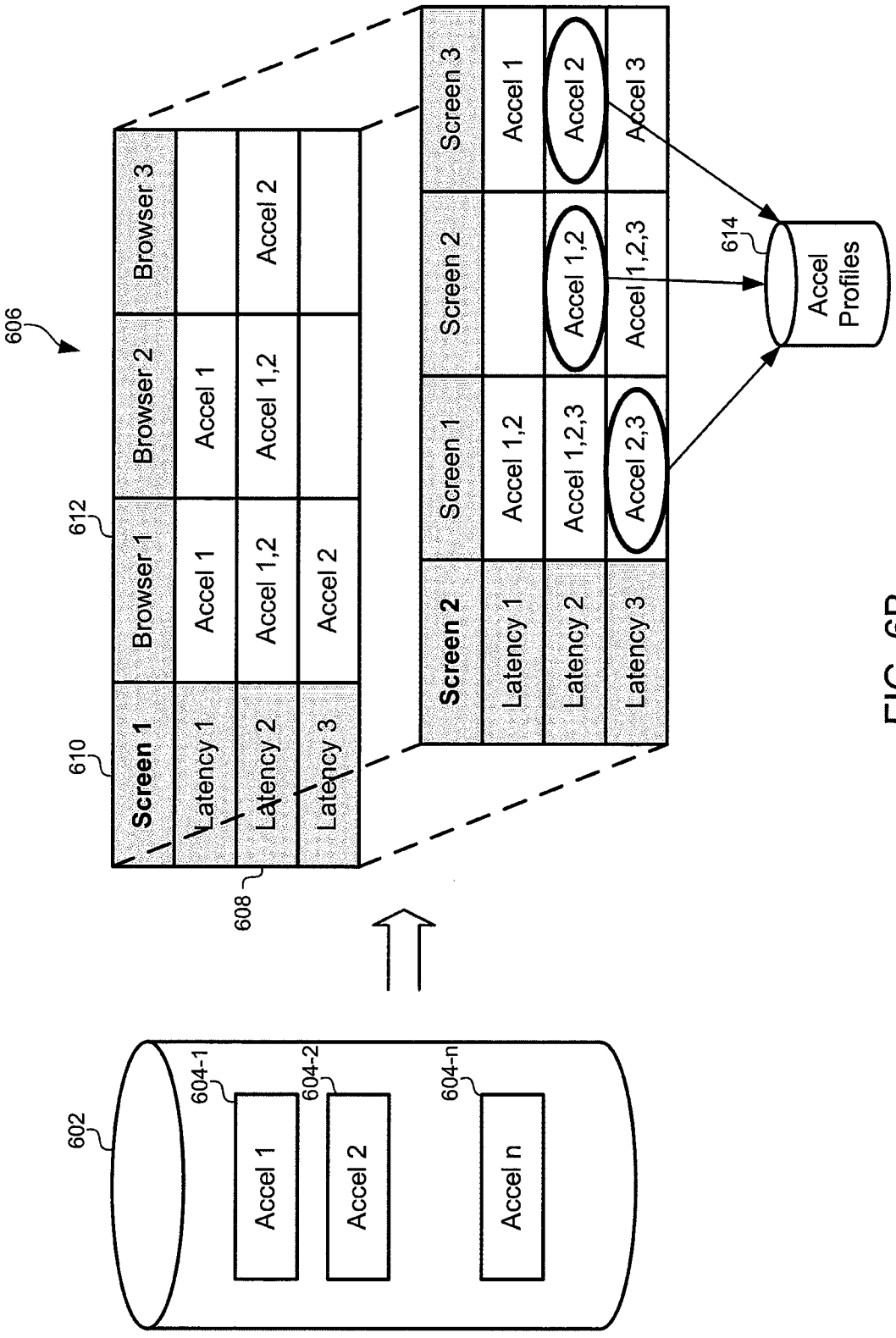

606
612
610
608
Screen 1
Browser 1
Browser 2
Browser 3
Latency 1
Latency 2
Latency 3
Accel 1
Accel 1,2
Accel 2
Accel 1
Accel 1,2
Accel 2
Screen 2
Screen 1
Screen 3
Latency 1
Latency 2
Latency 3
Accel 1,2
Accel 1,2,3
Accel 2,3
Accel 1
Accel 2
Accel 3
Accel 1,2
Accel 1,2,3
614
Accel Profiles
602
604-1
604-2
604-n
Accel 1
Accel 2
Accel n

FIG. 6B

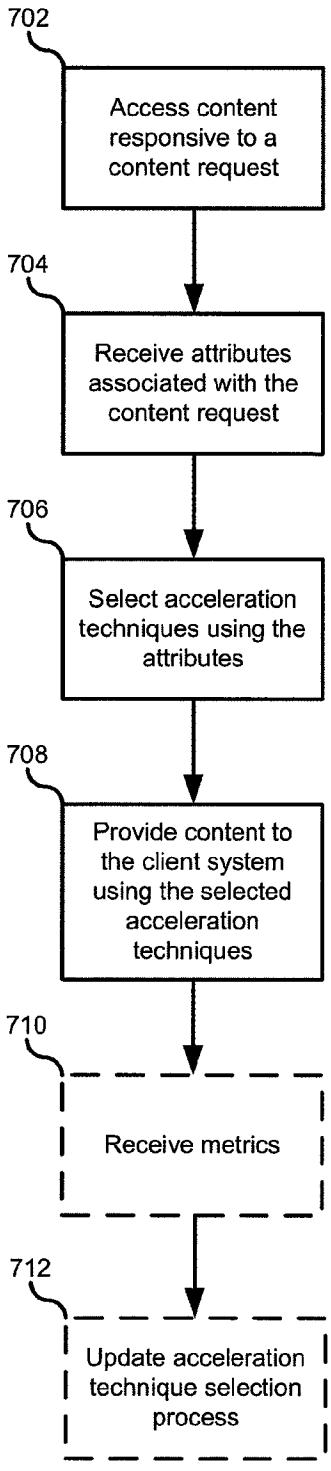

702
Access content responsive to a content request
704
Receive attributes associated with the content request
706
Select acceleration techniques using the attributes
708
Provide content to the client system using the selected acceleration techniques
710
Receive metrics
712
Update acceleration technique selection process

FIG. 7A

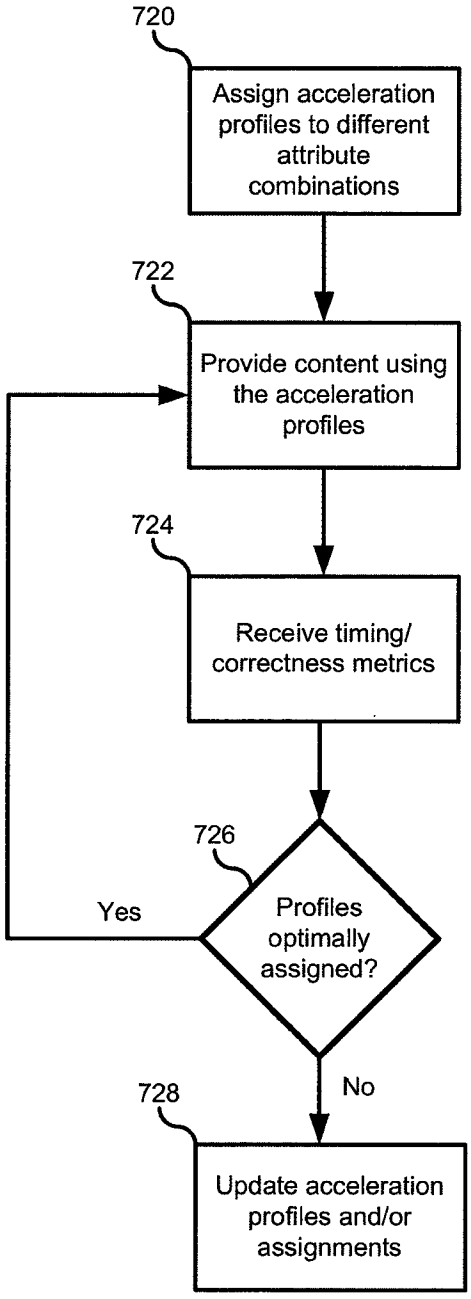

720
Assign acceleration profiles to different attribute combinations
722
Provide content using the acceleration profiles
724
Receive timing/ correctness metrics
726
Profiles optimally assigned?
Yes
No
728
Update acceleration profiles and/or assignments

FIG. 7B

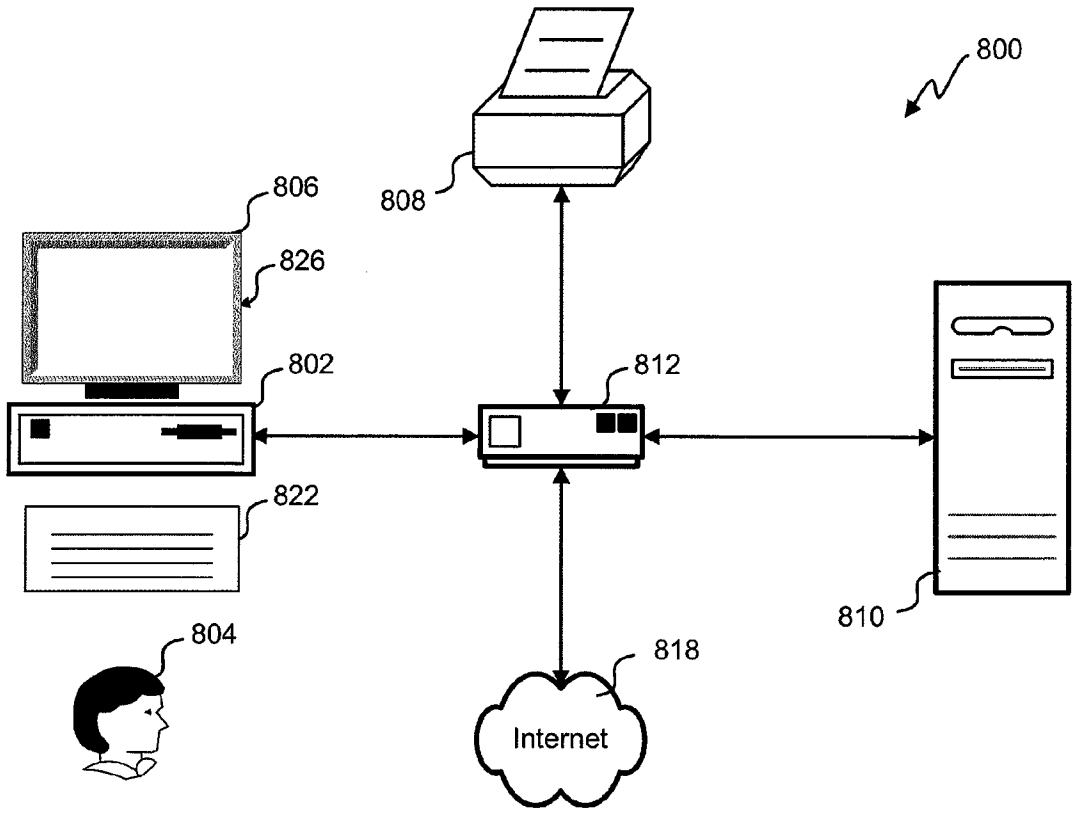


FIG. 8

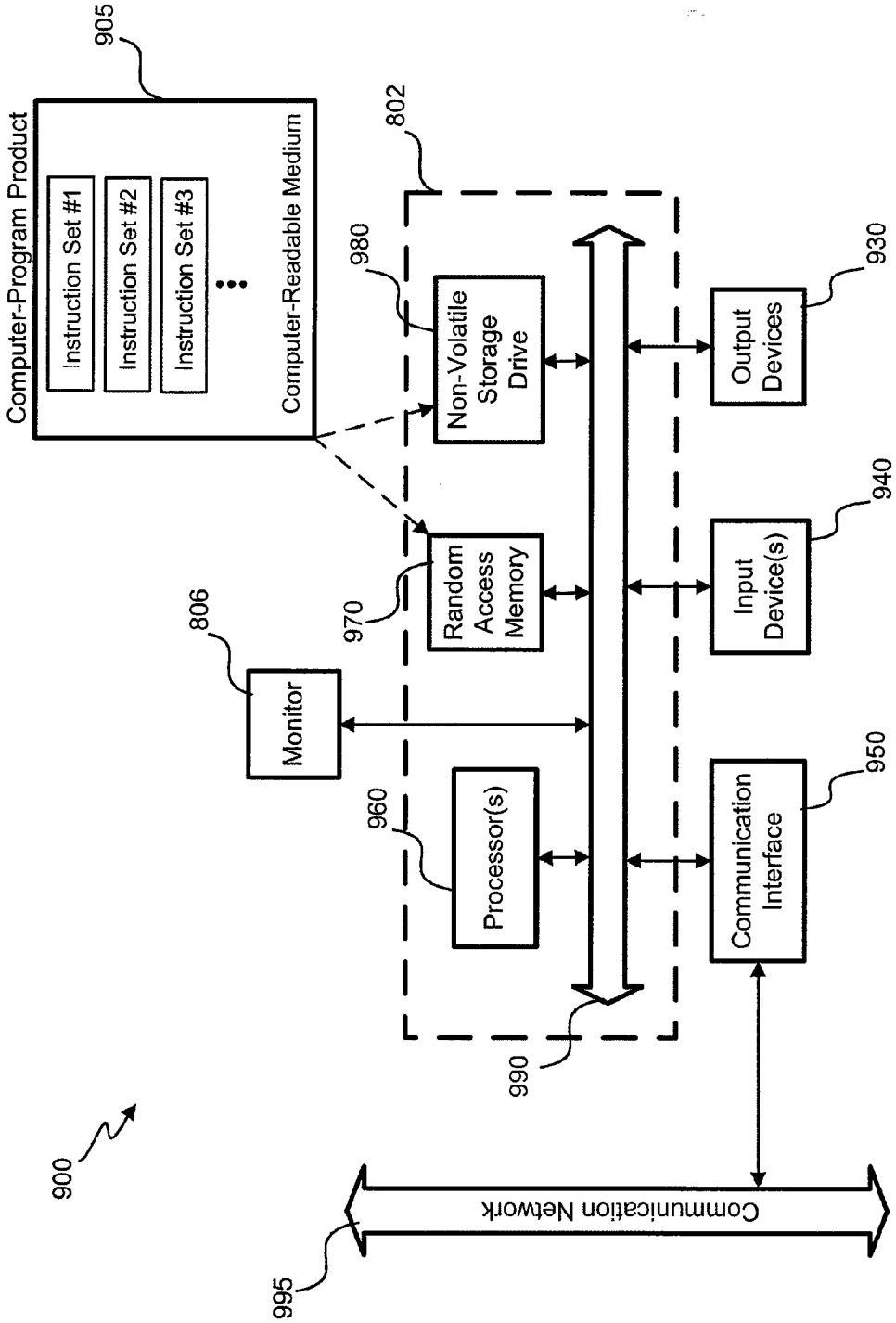

905
Computer-Program Product
Instruction Set #1
Instruction Set #2
Instruction Set #3
Computer-Readable Medium
802
980
Non-Volatile Storage Drive
930
Output Devices
940
Input Device(s)
970
Random Access Memory
806
Monitor
960
Processor(s)
950
Communication Interface
990
900
995
Communication Network

FIG. 9

DYNAMICALLY SELECTING BETWEEN ACCELERATION TECHNIQUES BASED ON CONTENT REQUEST ATTRIBUTES

FIELD OF THE INVENTION

This disclosure relates in general to a content delivery network (CDN) and, but not by way of limitation, more specifically to the acceleration of access to content provided by the CDN.

BACKGROUND OF THE INVENTION

A content delivery network (CDN) is a large distributed system of servers deployed in multiple data centers thoughout the Internet. The goal of a CDN is to serve content to end-users with high availability and high performance. Besides better performance and availability, CDNs also offload the traffic served directly from the content provider's origin infrastructure. CDNs can include geographically distributed points of presence (POPs) to locate edge servers close to end users. CDNs are capable of delivering content in high demand with higher quality of service (QoS). Content can be requested from a CDN using a universal resource locator (URL). Various techniques are used to route a URL request to a nearby POP, for example, in order to efficiently retrieve content.

The traffic over the Internet is growing rapidly as is the complexity and size of the information moved from sources of information to users of such information. Bottlenecks in the movement of data between CDN servers and client systems decrease the quality of the user experience. Traffic is expected to increase faster than the ability to resolve data transfers over the Internet. Therefore, improvements in the art are needed.

BRIEF SUMMARY OF THE INVENTION

In one embodiment, a system for dynamically selecting from among a plurality of acceleration techniques implemented in a Content Delivery Network (CDN) using attributes associated with content requests may be presented. The system may include a network interface that receives the content requests from a plurality of client systems for content stored in a plurality of edge servers distributed geographically throughout the CDN. In some embodiments, the content requests may include a first content request for first content, the first content request may originate from a first client system in the plurality of client systems, and/or the first content request may be associated with one or more attributes. The system may further include an intermediate server that accelerates access to the content stored in the plurality of edge servers. The intermediate server may include a first interface coupled to the network interface, a second interface configured to communicate with at least one of the plurality of edge servers of the CDN, and a memory device having stored thereon instructions for executing each of the plurality of acceleration techniques. The intermediate server may also include a processor configured to access the one or more attributes associated with the first content request, select one or more acceleration techniques from the plurality of acceleration techniques. The one or more acceleration techniques may be selected based on the one or more attributes. The processor may be further configured to use the one or more acceleration techniques to provide the content to the first client system.

In some embodiments, the one or more attributes may affect a performance of the one or more acceleration techniques in providing the first content to a user of the first client system. The one or more attributes may also be descriptive of the first client system or a network through which the first content request is received. The one or more attributes may include a geographic location of the first client system. The one or more attributes may include a measurement of latency associated with transmitting content from the CDN to the first client system. The one or more attributes may include a device type of the first client system. The one or more attributes may include a screen size of the first client system. The intermediate server may be physically combined with one of the plurality of edge servers of the CDN.

In some embodiments, the memory device may also store a plurality of acceleration profiles. Each of the plurality of acceleration profiles may include at least one of the plurality of acceleration profiles. The one or more acceleration techniques selected by the intermediate server may be selected as one of the plurality of acceleration profiles. The processor may be further configured to receive metrics from the first client system, where the metrics are associated with a performance in providing the first content to a user of the first client system. The processor may be further configured to dynamically update a process by which the one or more acceleration techniques are selected based on the metrics, and use the updated process to select acceleration techniques for subsequent requests associated with similar attributes. The first client system may include a testbed simulator that is configured to simulate a plurality of client system configurations and network connections.

In another embodiment, a method of dynamically selecting from among a plurality of acceleration techniques implemented in a CDN using attributes associated with content requests may be presented. The method may include receiving a first content request for first content through a network interface. In some embodiments, the network interface may be configured to receive the content requests from a plurality of client systems for content stored in a plurality of edge servers distributed geographically throughout the CDN. The first content request may originate from a first client system in the plurality of client systems, and the first content request may be associated with one or more attributes. The method may also include accessing, by an intermediate server, the one or more attributes associated with the first content request. In some embodiments, the intermediate server may be configured to accelerate access to the content stored in the plurality of edge servers. The method may additionally include selecting, by the intermediate server, one or more acceleration techniques from the plurality of acceleration techniques, where the one or more acceleration techniques are selected based on the one or more attributes. The method may further include using, by the intermediate server, the one or more acceleration techniques to provide the content to the first client system.

BRIEF DESCRIPTION OF THE DRAWINGS

The subject matter that is regarded as the invention is particularly pointed out and distinctly claimed in the claims at the conclusion of the specification. The foregoing and other objects, features, and advantages will be apparent from the following detailed description taken in conjunction with the accompanying drawings.

FIG. 1 illustrates a block diagram of an embodiment of a content distribution system.

FIG. **2** illustrates a block diagram of an embodiment of a point of presence (POP) that is part of a content delivery network (CDN).

FIG. **3**A depicts a POP arrangement with an intermediate server for accelerating content, according to some embodiments.

FIG. **3**B illustrates an alternate system for accelerating web-based content, according to some embodiments.

FIG. **4**A illustrates a flow diagram of content requests and responsive content processed by an intelligent acceleration system, according to some embodiments.

FIG. **4**B illustrates a second flow diagram of content requests and responsive content processed by an intelligent acceleration system, according to some embodiments.

FIG. **5** illustrates a system for simulating webpage load times, according to some embodiments.

FIG. **6**A illustrates a lookup file for determining the optimal set of acceleration techniques based on attributes associated with a content request, according to some embodiments.

FIG. **6**B illustrates a lookup file using acceleration profiles, according to some embodiments.

FIG. **7**A illustrates a flowchart of a process for selecting acceleration techniques based on attributes associated with the content request, according to some embodiments.

FIG. **7**B illustrates a method of optimizing acceleration profiles based on attributes and metrics, according to some embodiments.

FIG. **8** illustrates an exemplary environment in which some embodiments may be implemented.

FIG. **9** illustrates one example of a special-purpose computer system, according to some embodiments.

DETAILED DESCRIPTION OF THE INVENTION

The ensuing description provides descriptions of exemplary embodiments only, and is not intended to limit the scope, applicability, or configuration of the disclosure. Rather, the ensuing description of the exemplary embodiments will provide those skilled in the art with an enabling description for implementing the embodiments of the claims. It will be understood that various changes may be made in the function and arrangement of elements without departing from the spirit and scope as set forth in the appended claims.

Referring first to FIG. **1**, a block diagram of an embodiment of a content distribution system **100** is shown. The content originator **106** offloads delivery of the content objects to a content delivery network (CDN) **110** in this embodiment. The content originator **106** produces and/or distributes content objects and may include a content provider **108**, a content site **116**, and/or an origin server **112**. The CDN **110** can both cache and/or host content in various embodiments for third parties, such as the content originator **106**, to offload delivery and typically provide better quality of service (QoS) to a broad spectrum of end-user systems **102** distributed worldwide.

In this embodiment, the content distribution system **100** locates the content objects (or portions thereof) and distributes the content objects to one or more end-user systems **102**. The content objects can be dynamically cached and/or hosted within the CDN **110**. A content object may include any content file or content stream and could include, for example, video, pictures, data, audio, software, analytics, and/or text. The content object could be live, delayed, or stored. Throughout the specification, references may be made to a content object, content, content stream and/or content file, but it is to be understood that those terms could be used interchangeably wherever they may appear.

Many content providers **108** may use a CDN **110** or even multiple CDNs **110** to deliver the content objects over the Internet **104** to end users **128**. The CDN **110** may include a number of points of presence (POPs) **120**, which are geographically distributed through the content distribution system **100** to deliver content. Various embodiments may have any number of POPs **120** within the CDN **110** that are generally distributed in various locations around the Internet **104** so as to be proximate to end-user systems **102** in a network sense. Routing requests between the multiple POPs can be done during the DNS resolution and refined by assignment of an edge server. Other embodiments use routing, redirection, Anycast, DNS assignment and/or other techniques to locate the particular edge server that are able to provide content to the end users **128**. In addition to the Internet **104**, a wide area network (WAN), and/or a local area network (LAN) **114** or other backbone may couple the POPs **120** with each other and with other parts of the CDN **110**.

When an end user **128** requests content, such as a web page, through its respective end-user system **102** while browsing, the request for the web page can be passed either directly or indirectly via the Internet **104** to the content originator **106**. The content originator **106** may be defined as the source or re-distributor of content objects. The content site **116** may include an Internet web site accessible by the end-user system **102**. For example, the content site **116** could be a web site where the content is viewable using a web browser. In other embodiments, the content site **116** could be accessible with application software or customized hardware other than a web browser, for example, a set top box, a content player, video streaming appliance, a podcast player, an app running on a smart phone, etc. The content provider **108** can redirect such content requests to the CDN **110** after they are made, or alternatively can formulate the delivery path by embedding the delivery path into the universal resource indicators (URIs) for a web page. In either case, the request for content can be handed over to the CDN **110** in this embodiment by having the end-user system **102** perform a DNS look-up so as to choose which of the multiple POPs **120** should provide the requested content.

A particular edge server may retrieve the portion of the content object from the content provider **108**. Alternatively, the content provider **108** may directly provide the content object to the CDN **110** and its associated POPs **120** through prepopulation, i.e., in advance of the first request. The servers of the CDN **110** may include edge servers in each POP **120** that are configured to serve end user requests and/or store the actual content. The origin server **112** may continue to store a copy of each content object for the content originator **106**. Periodically, the content of the origin server **112** may be reconciled with the CDN **110** through a cache, hosting, and/or pre-population algorithms. Some content providers could use an origin server within the CDN **110** to host the content and thus avoid the need to maintain a separate copy.

Once the content object is retrieved from the origin server **112**, the content object may be stored within the particular POP **120** and may be served from that POP **120** to the end-user system **102**. The end-user system **102** may receive the content object and processes it for use by the end user **128**. The end-user system **102** could be a personal computer, media player, tablet computer, handheld computer, Internet appliance, phone, IPTV set top, video stream player, streaming radio, PDA, smart phone, digital music player, or any other device that can be configured to receive and process content objects. In some embodiments, a number of the end-user systems **102** could be networked together. Although this embodiment only shows a single content originator **106** and a

single CDN **110**, it will be understood that there could be many of each in various embodiments. Additionally, in some embodiments a content originator **106** could have a "captive" CDN **110** that is optionally used for its content when a third-party CDN is used to shed requests.

With reference to FIG. **2**, a block diagram of an embodiment of a POP **120** is shown that is part of the CDN **110**. Although only one POP **120** is depicted, there may be a number of POPs **120** similarly configured and geographically distributed throughout the CDN **110**. The POPs **120** can communicate through a WAN router **210** and/or an Internet router **220** for locating content objects. An interface to the Internet **104** from the POP **120** accepts requests for content objects from end-user systems **102**. The request comes from an Internet protocol (IP) address in the form of a URI.

Edge servers **230** may be implemented using general purpose computers loaded with software to perform various functions for the CDN **110**. The edge servers **230** could be rack mounted or arranged in clusters. Multiple hardware processors and storage media could be used to implement each edge server **230**. Each edge server **230** can load multiple instances of the same software and/or a variety of software to implement various functionalities. For example, software may be used on edge servers to implement switching fabric, routing, caching, hosting, DNS lookup, analytics, business rules, delivery assignment, and/or the like. The software instances can scale with the size of each POP **120**. Different edge servers **230** may have a different set of functionality as defined by the software instances that are programmed to run on each edge server **230**.

Switch fabric **240** assigns the request to one of the edge servers **230** according to a routing scheme such as round robin, load balancing, Cache Array Routing Protocol (CARP), random, and/or the like. In this embodiment, the switch fabric may be aware of which edge servers **230** have particular capabilities and may assign requests within the group having the particular capability to store and serve the particular content object referenced in a requested URI. A protocol such as CARP may be used in this embodiment to dispense the URIs between the edge servers **230**. Every time that a particular URI is requested from the group, it may be assigned to the same edge server **230**. For purposes of assigning a request, edge servers may be grouped together based on their ability to provide a requested content object, service a particular type of request, and/or the like.

In another embodiment, the switch fabric **240** assigns the request to one of the edge servers **230**, which can either service the request or reassign it to a neighboring edge server **230** with software to perform an assignment master function. The switch fabric **240** sends each packet flow or request to an edge server **230** listed in the configuration of the switch fabric **240**. The assignment can be performed by choosing the edge server **230** with the least amount of connections or the fastest response time. In some embodiments, the switch fabric **240** may assign the packet flow somewhat arbitrarily using round robin or random methodologies. When the chosen edge server **230** receives the packet flow, an algorithm may be used by the chosen edge server **230** to potentially reassign the packet flow between a group of edge servers to the one dictated by the algorithm. For example, the switch fabric **240** could choose a second edge server **230-2** being the next in the round robin rotation. The second edge server **230-2** could process the request and find that the first edge server **230-1** is being assigned this type of request. The request could then be reassigned to the first edge server **230-1** to fulfill.

As described above, the CDN **110** may be used to host content for others. Content providers **108** may upload content to an edge server **230** that hosts the content and functions as an origin server. After the content provider **108** places a content object in the CDN **110** it need not be hosted on the origin server **112** redundantly. Edge servers **230** can perform the hosting function within the CDN **110** with other edge servers **230** perhaps caching the same content that is hosted by another edge server **230**.

Requests from end-user systems **102** are assigned to an edge server **230** that may cache the requested content object. On occasion, the edge server **230** receiving a request does not have the content object stored and available for immediate serving. This so-called "cache miss" triggers a process within the CDN **110** to effectively find the content object (or portion thereof) while providing adequate Quality of Service (QoS). The content may be found in neighboring edge servers **230** in the same POP **120**, in another POP **120**, or even an external origin server **112**. The various edge servers **230** may be grouped for various URIs uniquely. In other words, one URI may look to one group of edge servers **230** on a cache miss while another URI will look to a different group of edge servers **230**. In various embodiments, a particular URI could be assigned to one or more edge servers **230** in a single POP, multiple POPs or even in every POP. Generally, more popular content is stored on more edge servers **230** and more POPs **120**.

When servicing requests from end-user systems **102**, some form of content processing may be performed on the requested content before it is delivered from an edge server **230**. In some cases, content processing may be performed by special software/hardware modules that are integrated with existing devices within the POP **120** or on the origin server itself **112**. If the content processing is performed on an edge server **230** or on an origin server **112**, the software/hardware performing the content processing may need to be distributed to each edge server **230** and/or each origin server **112**.

As modern CDNs proliferate and become more complex, content processing may become more specialized and programming intensive. Embodiments described herein may disclose an intermediate content processing server that can be used to process and deliver content from edge servers **230** to the client systems requesting the content. One particular type of content processing that may significantly improve the user experience is content acceleration. Acceleration is an accumulation of processing techniques to optimize the end user experience when viewing web pages. The most interesting content in the web page may be sped up using a number of techniques. These techniques can optimize different portions of the delivery chain. For example, acceleration can optimize gathering information from the origin server along with accelerating how that information is provided to the client system. Acceleration can also optimize gathering content from other POPs. In some embodiments, frontend and backend acceleration engines can have additional caching functionality that is separately indexed.

Embodiments herein may use acceleration techniques that can speed up the delivery of content to a client system. Client systems may include the end-user systems of FIG. **1**. While it is beyond the scope of this disclosure to exhaustively recite all possible acceleration techniques, a short overview of a number of different acceleration techniques may be provided herein to provide a sampling of different embodiments. It will be understood by one having skill in the art that these recited acceleration techniques are merely exemplary and not meant to be limiting. The embodiments described herein for selecting among a plurality of acceleration techniques based on attributes of the client system, delivery networks, and client

software should be interpreted as being compatible with all possible web acceleration techniques.

Some acceleration techniques selected by embodiments described herein may include caching, or pre-caching, content based on predictive algorithms. Frequently accessed documents may then be sent to the client system at a faster transfer rate than a more remote edge server could provide. Similarly, cache refreshing techniques can ensure that frequently accessed content is available to client systems when it is most likely to be requested.

Many CDNs may resolve URIs in order to locate content on an edge server. Some acceleration techniques may preemptively resolve URIs that are identified in a webpage in order to reduce transfer time. The webpage may be prescanned or pre-processed to identify URIs in the HTML or JavaScript. In addition to preemptively resolving URIs, predictive algorithms may determine which content items are likely to be accessed in the near future and prefetch those documents to be delivered to the client system.

Some acceleration techniques may also use data compression in order to facilitate rapid data transfer. By preemptively compressing data packages, they can be delivered to client systems more efficiently. Images and other types of media files may be compressed such that their quality is reduced if they are not immediately viewable on a device's screen. In other cases, reduced quality images may be preemptively transmitted to a client device, and additional image information may be transmitted upon an actual request to re-create a high-quality image.

Many acceleration techniques used herein may edit or otherwise optimize the actual HTML or JavaScript used to define a webpage or access content. For example, content requests may be grouped together such that repeated requests are minimized. Requests for content stored in common locations in edge servers may also be grouped together. JavaScript may be edited such that the default conditions are immediately processed and available, while alternate conditional statements can be evaluated after a page is loaded. TCP acceleration techniques may also be used. For example, an edge server may maintain a persistent TCP connection between the client device and the edge server.

FIG. **3**A illustrates a system for accelerating web-based content, according to some embodiments. The system may be part of a CDN, and may be located within one POP **120** of the CDN. The pop may be connected to other networks, including the Internet, through an Internet router **220**. The Internet router **220** may have, or may also be referred to as, a network interface **320**. The network interface **320** may be configured to receive requests from client systems for content, and to provide content from edge servers of the CDN to the client systems in response to such requests. The network interface **320** may generically refer to any gateway, router, interface, system, and/or terminal through which such requests are passed between client systems and any device on the CDN.

Within the POP **120** the switching fabric **240** may be communicatively coupled to one or more intermediate servers **302**. The intermediate servers may also be referred to as a web page access accelerator (WPA), and acceleration server, a web accelerator, and/or the like. Instead of providing webpage content directly between the edge servers **230** and a client system, traffic may instead be routed through one or more of the intermediate servers **302** when the content can be configured for accelerated access. Accordingly, a request for webpage content may be directed through a particular one of the intermediate servers **302** that is equipped with various acceleration mechanisms as described herein. As the switching fabric **240** routes content requests and content deliveries through the POP **120**, appropriate acceleration techniques may be identified by the intermediate servers **302**, and these acceleration techniques may be applied to the content as it is routed to the client systems.

The intermediate servers **302** may include one or more acceleration engines configured to apply acceleration techniques. The intermediate servers may also include storage devices, such as hard disks, flash memory, databases, and/or the like that are configured to store instructions for carrying out the various acceleration techniques. The intermediate servers may also include communication interfaces configured to communicate with the edge servers **230** as well as the network interface **320** and/or Internet router **220**.

Although many embodiments described herein make particular reference to edge servers within a CDN, the invention should not be limited to only CDN environments. Other embodiments may also include server farms of which the intermediate servers **302** are a part. Some embodiments may be implemented using cloud computing environments that deliver both content and application services. Some embodiments may be simple web servers equipped with web acceleration software.

Although FIG. **3**A illustrates intermediate servers **302** that are separate from the edge servers **230**, other embodiments may combine the intermediate servers **302** with the edge servers **230**. FIG. **3**B illustrates an alternate system for accelerating web-based content, according to some embodiments. The system of FIG. **3**B may be very similar to that of FIG. **3**A; however, one or more of the edge servers **230** may be physically or logically integrated together with an intermediate server. For example, edge server **230-1** could be physically or logically integrated with intermediate server **306**. The two "servers" may share storage space, processing and computing resources, and communication ports. The two servers may be physically integrated together in the same server rack, or even in the same mainframe computing device. In some embodiments, the intermediate server **306** may be implemented in software and referred to as a web accelerator.

In this configuration, some edge servers may be integrated with intermediate servers while others are not. Content provided by edge server **230-1** may be accelerated by the intermediate server **306**, as well as any of the other intermediate servers **302**. For example, certain acceleration techniques may be implemented by intermediate server **306** that may apply only to content stored on edge server **230-1**. After being processed by intermediate server **306**, the content provided by edge server **230-1** may also be processed by intermediate server **302** according to different acceleration techniques. In other cases, edge servers that are not coupled with intermediate servers may send content to intermediate server **306** for processing. In other words, intermediate server **306** may accelerate content from edge server **230-1** as well as any of the other edge servers **230**, depending upon the particular embodiment.

POP configurations according to FIG. **3**A and/or FIG. **3**B may be geographically distributed throughout the CDN. In some cases, web acceleration techniques may be distributed throughout the CDN. For example, web acceleration techniques that take place on edge servers or origin servers that are further away from the client system may be referred to as "back-end" acceleration techniques, while acceleration techniques that take place on edge servers that are closer to the client systems or that directly deliver content to client systems may be referred to as "front-end" acceleration techniques. Bifurcating the acceleration process between front-end and back-end systems is described fully in the commonly-assigned U.S. patent application Ser. No. 13/571,320 entitled

"Inter Point of Presence Split Architecture" filed on Aug. 9, 2012, which is hereby incorporated by reference for all purposes.

Some acceleration techniques have been particularly designed to operate using the POP configurations according to FIG. **3**A and/or FIG. **3**B. Some of these acceleration techniques are described fully in the commonly-assigned U.S. patent application Ser. No. 12/848,611 entitled "Systems and Methods Thereto for Acceleration of Web Pages Access Using Next Page Optimization, Caching and Prefetching Techniques" filed on Aug. 2, 2010, which is hereby incorporated by reference for all purposes. For example, one acceleration technique may include prefetching resources of subsequent or additional webpage requests. Many of the resources are common to several or all the pages that may be fetched, and therefore pre-fetching such resources is likely to save fetching them for other pages. In some cases, the intermediate servers **302** can intercept webpage code and parse it prior to sending it out to client systems.

In some cases, after browsing a site for a while, newly-requested pages load very quickly because most of their resources are already in the browser's cache. The only non-cached items are the HTML itself and a few resources which have not been seen on previous pages yet. In such cases, loading the HTML from the network represents a large percentage of loading the entire web page, even when the HTML is loaded very quickly. Therefore, a script may be added that detects the mouse presence over a link, a button, or any other means of selecting a URL. This script may be enabled to detect whether the focus is on the link, button, or other control. After the detection, the script, might or might not wait a while to reduce the number of false positives, after which it pre-fetches the relevant page. If, during this time, the mouse moved from the location of a URL, or has otherwise lost its focus of the web page, the pre-fetch is canceled. If the page pointed to by the link is small and the server is fast, many times it is possible to bring the page to the browser's cache before the link is actually clicked, thus substantially reducing the load time of that page as it appears to the user.

Another acceleration technique may leverage the fact that on the same web site, many pages have common resources. A web page may be processed differently for a case where it was reached from within the web site compared to a case where it was reached from outside of the web site. The conclusion about how the page was reached may be determined according to the HTTP headers of the web page, a special cookie, the existence of a "referrer" header, a configuration, or any other technique. The intermediate servers **302** may intercept the webpage and send the client system to a different version thereof.

According to another acceleration technique, caching of dynamic data may include separating the static parts of the HTML page from the dynamic parts. This can be performed by, for example, "marking" parts of the HTML as static or dynamic, or automatically, by studying instances of the same page and deducing which parts are common. Once the static and dynamic parts of the page have been marked, this information can be used to accelerate page loading. First, the static part can be processed to achieve optimizations using techniques discussed in this document or otherwise, and the processed data may be kept in cache. Once a request for the page is accepted, the original page may be requested, and then the dynamic parts of it can be "applied" to the processed static parts. The resulting page can then sent to a client system. The static part, processed or not, can be sent to a client system from the cache, without requiring a request to the web server to obtain the original page. However, a code may be injected into the page which directs the browser, without any need of additional support, to asynchronously send additional requests to retrieve the dynamic data. When the addition data is retrieved, it can be injected into the DOM in the correct locations.

In another acceleration technique, resources may be intelligently cached. Every resource may be equipped with a version indicator, and this version may be increased every time the resource is changed. Every whole web page, i.e., a web page along with all its resources, may also have a version indicator. The version of the page, or an identifier that stands for this version, may be sent to the client system along with the web page every time it is requested. If the client system has already accessed that page, the version of the page received by the client system, or an identifier which represents the current version, may be sent with the request. In one embodiment, this may be achieved by using cookies. All the resources may be associated with a very long expiration time. Thus, when the browser encounters these resources it will be able to retrieve them from the browser cache.

In another acceleration technique, when a page is served by a web server or proxy, the pages that are candidates for being the next pages to be served for the same client system may also be processed. The candidate web pages can be deduced either statically from analyzing the current page, for example, by looking at all the links, or by collecting statistics and choosing accordingly. A configuration or a combination may be used, for example, by determining whether there are many links and ordering their processing according to calculated statistics. This can be done recursively to any depth and process may be located on all the selected pages.

Typically, pages of a particular web site will contain common data. In another acceleration technique, the common data between pages is sent only once. Only the non-common data of a page is sent to the user every time that the user requests a new page, while the common data is sent only once. The intermediate servers **302** may compute a difference between the requested pages and send only such differences to the client system.

As stated above, these acceleration techniques are merely exemplary, and not meant to be limiting. Many additional acceleration techniques may be used by embodiments described herein.

Embodiments described herein may focus on dynamically selecting a set of acceleration techniques from a plurality of possible acceleration techniques based on attributes associated with a content request. These attributes may indicate a particular connection type, a hardware description such as screen size and/or device type, a software characteristic such as browser type and/or browser version, a data carrier or ISP, a latency metric, a geolocation, and/or the like. Each of these attributes may cause particular acceleration techniques to be more or less effective, depending on how they are combined and evolving over time. Some embodiments may heuristically determine the optimal set of acceleration techniques for any client system configuration. The client system configuration may be determined by evaluating the received attributes and a set of acceleration techniques may then be applied dynamically as the request for content is serviced.

Currently, individual websites are analyzed extensively to determine which acceleration techniques should be applied when providing associated content in response to a client system request. The HTML code may be analyzed, the content objects may be analyzed, business rules may be applied to website content, JavaScript functions may be optimized, and/or the like. However, while the content and web code may be extensively analyzed and optimized on the server side of the

transaction, current acceleration techniques are applied in a one-size-fits-all fashion to the many different client system configurations that may request content.

One problem with this approach is that many client system configurations may particularly benefit from some acceleration techniques that may not be universally applied on the server side. Other client system configurations may interfere with some acceleration techniques and thereby reduce their effectiveness, or even slow down the transaction.

Another problem is that it takes a tremendous amount of time and analysis to tailor a website to effectively use the currently available acceleration techniques. This process may take multiple months to complete. However, by the time a fully optimized website is made available using tailored acceleration techniques, client system configurations may have evolved to the point where the website acceleration scheme is now obsolete. Network characteristics and bottlenecks may have changed, new operating system versions may have been released, new network topologies and technologies may have been developed or improved, latency may have changed over time, browsers may have been upgraded, and/or the like. Each of these client system configuration changes may affect the optimization of the acceleration techniques that was painstakingly developed only a few weeks earlier.

For example, modern web browsers are now being equipped with acceleration techniques of their own to increase the speed at which web content is rendered in the browser display. Browser updates may be distributed to client systems rapidly and frequently. Some acceleration techniques used by browsers may interfere with acceleration techniques implemented by a CDN. For example, browser software may apply an acceleration technique whereby it stops downloading resources until it resolves a line of JavaScript. On the other end, the server-side HTML code may have been optimized to continuously execute JavaScript while content is downloaded. These two conflicting paradigms may act to cancel out the benefits of each to the detriment of the user experience.

In another example, certain acceleration techniques may be particularly useful in long-latency situations. Geolocations of the client systems that are far away from the corresponding edge server may be used as a proxy for latency in some situations. However, when a client system is located near the corresponding edge server in either a network latency or geographic sense, acceleration techniques used to reduce latency may simply add unnecessary overhead to the content transmission. A fast network that is very close to the data origin edge server may actually be slowed down by using acceleration techniques, such as image bundling. Counterintuitively, the best acceleration technique may be to do nothing for the limited subset of customers who are close to the data origin.

In another example, device characteristics of a client system may make some acceleration techniques more useful than others. Depending on the screen size, processing power, and/or operating system, content can be optimized accordingly.

Currently, acceleration techniques require developers to bet on the right techniques to use given the commonly used browsers, networks, and devices at the time of development. However, once websites are rolled out that implement the selected acceleration techniques, the website may be only two to three months from being obsolete given the rapid change in client system configurations and transmission networks.

Therefore, the embodiments described herein may analyze each request individually as they are received. The content request may include or may be otherwise associated with attributes. These attributes may be associated with the transmission network, a request type, device characteristics, and/or the like. For each particular request, the attributes may be analyzed, and a a set of acceleration techniques may be selected from a group of possible acceleration techniques and applied on a per-request basis.

By analyzing each request individually, the best set of acceleration techniques may be applied and the user experience can be improved. For example, a user may request a webpage using a desktop computer connected through a T1 connection with a 1024×2048 screen size. These attributes may dictate that a first set of acceleration techniques may be used. The same user may next request the same website using a smart phone connected through a mobile 4G network with approximately a 2"×4" screen size. Even though the user may be sitting in the same room for both requests, the second request to the smart phone for the same website may utilize a completely different set of acceleration techniques that are applied in real time as the request is serviced by the content delivery network.

FIG. **4**A illustrates a flow diagram of content requests and responsive content processed by an intelligent acceleration system, according to some embodiments. A client system **402** may send a content request to a website. For example, a user may enter a URL into a navigation field of a web browser or may click on a link within a web browser. In some cases, the web browser may be directed to a website at an origin server, and the origin server may redirect the request or portions of the request to locations within the CDN. Alternatively, elements of the URL itself may direct the web browser to first contact the CDN.

The CDN may receive the request for content and may intelligently determine the best edge server(s) to provide the content. A particular edge server **230** may be identified and receive the content request (**410**). Note that FIG. **4**A illustrates the content request originating from the client system **402** and eventually reaching the edge server **230**. The intermediate transactions that may involve an origin server and other server/devices within the CDN have been omitted for clarity.

As the content is sourced from the edge server **230**, the CDN may pass the content through an intermediate server **302** as described above. Note that in some embodiments (not shown) the function of the intermediate server **302** may be implemented on the same physical hardware as the edge server **230**. The intermediate server **302** may intercept the content (**414**) and apply one or more acceleration techniques (**416**).

The intermediate server **302** may select the acceleration techniques to apply from a plurality of available acceleration techniques. The selection process may be based on attributes that are associated with the request. In one embodiment, the attributes may be transmitted from the client system **402** to the intermediate server **302** directly (**412**). The intermediate server **302** may intercept the request and extract attributes therefrom. The intermediate server **302** may also directly query the client system **402**. In some embodiments, JavaScript generating a URL for the client system **402** may also be configured to provide information to the intermediate server **302**. In some embodiments, the attributes may be automatically determined based on the content request. For example, an IP address associated with the content request may be analyzed to determine a geolocation. Some embodiments may query a third-party service to obtain information associated with the IP address, such as a business, a connection type, and/or the like. Network timing information may be ascertained to determine a latency metric. "Pings" may also be sent

between the CDN and the client system to determine the latency metric. Tags provided in the request may identify a browser type and/or client system configuration. Other techniques for ascertaining attributes not specifically listed may also be used. The acceleration technique selection process and the particular attributes that may be used will be described further herein below.

The intermediate server **302** can then apply the selected acceleration techniques to the content and provide the content to the client system **402** using the selected acceleration techniques (**418**). In this configuration, neither the edge server **230** nor the client system **402** need be aware that the intermediate server **302** is applying acceleration techniques to the content during delivery. Therefore, different acceleration techniques and intermediate servers **302** may be swapped in and out of the CDN as they are developed or become obsolete.

In some embodiments, there may be a feedback loop associated with the content delivery to client systems **402**. The feedback loop may collect metrics describing the speed and correctness with which the content was provided and displayed on the client system **402**. Generally, two different types of metrics may be provided. First, metrics related to latency may be collected. Second, metrics related to how correctly the website/content was displayed on the client system may be collected.

Webpage metrics may be gathered by a number of different means. In one embodiment, the load sequence of a typical web page may include many different resources. Some of the resources may be visible while others may not be visible. Therefore, some of the visible resources may be part of the viewport while others are not, depending upon the screen size and/or the device type. Additionally, the speed at which a web page loads may be measured as perceived by a user once the viewport is complete. It is therefore very difficult to deduce when the page has finished loading from the user's point of view based merely upon network analysis. Current measurement techniques either calculate this time once all the components of the page have finished loading, including those that are invisible, or check the "onload" event of the HTML document, which also has only a small correlation to the actual user perceived load time.

In most websites, the last item to be loaded in the viewport is a graphic item, such as an image or a Flash object. This happens because the size of graphic items is relatively large and takes more time to load than the smaller-sized components. Therefore, graphic items are often loaded later than the textual, i.e., HTML, JavaScript, CSS, etc., elements. Using this assumption, the actual perceived load time of the page may be estimated.

For every background image, the server may create invisible dummy images that are marked as loaded when the background image finishes loading. The time the page starts loading is saved in memory. Once every predefined time interval (which can be set to different values depending on the desires granularity of the result) all of the graphical elements of the web page can be checked, for example, by scouting the DOM of the web page, by lists exported by the browser such as document images, or by any other means. The elements can be images, Flash objects, or any other type of element. For every element found, its position may be calculated, for example, by using all the elements starting from the selected element and finishing at the root of the DOM tree. If the element's position is in the viewport, it is added to the known viewport element list.

A graphical snapshot of the screen can be stored that may contain only the browser, or any other part of the screen, along with the elapsed time passed since the start load time. These snapshots can be later analyzed to determine the exact time the viewport has finished rendering. This process can be repeated for all known elements in the lists previously created. The load time can then be estimated, stored, and later retrieved by the intermediate server for analysis.

Another method of retrieving metrics related to site loading times and correctness may involve fingerprinting sites. A fingerprint for each website can be generated based on a number of images, types of JavaScript, length of HTML code, number of linked sites and/or content, advertising sources, and/or the like, in order to generate a unique signature for each webpage. The fingerprint generation algorithm could be invoked at regular time intervals as the webpage is loading and compared to the known fingerprint. When the current fingerprint of the webpage sufficiently matches the known fingerprint, the webpage may be considered loaded from the user's perspective.

Another method of retrieving metrics related to site loading times and correctness may involve using a browser API that implements the standard Navigation Timing Specification provided by W3C. The Navigation Timing Specification is a Java API detailing the timing information of the page load. Available in most newer browsers, it helps developers test user experiences remotely. The Navigation Timing Specification can be used to retrieve metrics such as the time when the user begins a navigation to a new page, a time when the first requested page returns to the browser, the time when the page is then parsed into a DOM, the time when the page has completed loading, and/or other timing-related information.

Newer browsers may also use the W3C Resource Timing Specification. While the Navigation Timing Specification addresses timing information associated with a navigation event, the Resource Timing Specification interface may allow JavaScript mechanisms to collect complete timing information related to resources within a webpage. The specification may be particularly useful for resources within a single webpage that are stored as content objects at various servers within the CDN. The Resource Timing Specification may be used to determine when each image or other content object on a webpage is loaded by retrieving the XY coordinates and the load times of every object on a webpage. An intermediate server could then perform an in-memory simulation of the page rendering and determine overall load times and metrics.

Another method for measuring metrics related to site loading times and correctness may involve taking screenshots and comparing RGB histograms of the webpage as it is loaded to known values. By comparing the incremental screenshots to a known RGB histogram value of the webpage when it is completely loaded, a numerical score can be derived representing the difference between the first and last RGB histogram to determine how far along the webpage is in the download and rendering process. In some embodiments, this value may be between 0.0 and 1.0. These values can be plotted over time to form a visual curve, the area above which can be calculated as a metric of how quickly the the webpage, or a part of the webpage, has been loaded that is visible to the user.

Using RGB values may be difficult to receive through a beacon-type data response because it may require screenshots rather than numerical values. However, the RGB values have been found to be related to numbers available through the W3C timing specifications. The numbers available through the timing specifications are easier to receive through a beacon-type data response. These numbers may then be used as a proxy to determine approximately what the RGB values would be for the visual portion of the webpage.

As used herein, a "beacon" may include any software routine configured to send information from the client system to

a content source, CDN, edge server, and/or intermediate server where the information includes metrics on webpage load times and/or correctness. For example, a beacon may include a small snippet of JavaScript at the bottom of a source code page of HTML for a website. The JavaScript may post values to the intermediate server or may store values for later retrieval. The beacon may wait for a time interval after the page is finished loading so as not to compete for resources and affect the loading of the webpage. In some embodiments, the browser may make an AJAX call to send metric information using XML back to the intermediate servers. This information may also include coordinates of each image on the page, which can then be analyzed to determine whether they were correctly loaded.

Many of the methods described in the preceding paragraphs for retrieving timing-related metrics from a webpage may also be used to provide metrics related to the correctness with which a webpage is rendered on the client system. For example, final RGB values of the webpage may be transmitted and compared to an expected RGB value. The beacon may transmit XY coordinates for each of the images and/or text segment on webpage, and these values may be compared to expected values. It should also be noted that the Resource Timing Specification may be used to retrieve the XY coordinates of every object on a webpage in addition to retrieving the load times. In some embodiments, the beacon may acquire periodic screen captures or snapshots of the webpage as it is loading on the client device. The screen captures can be transmitted back to the intermediate server, or alternatively, values derived from the screen captures can be transmitted back to the intermediate server. For example, the above-described fingerprint may be derived from a screenshot and send back to the intermediate server to be compared to an expected value. Other methods may also be used to retrieve metrics related to correctness.

FIG. **4**B illustrates a second flow diagram of content requests and responsive content processed by an intelligent acceleration system, according to some embodiments. This flow diagram may be very similar to the flow diagram of FIG. **4**A. In this case, the attributes may be sent as part of the original content request (**422**). When servicing the content request, the edge server **230** may send the content along with the attributes to the intermediate server **302** (**424**).

FIG. **5** illustrates a system for simulating webpage load times, according to some embodiments. In the systems of FIG. **4**A and FIG. **4**B, the metrics in the feedback loop are supplied by actual client systems making requests for content from the CDN. In contrast, the system of FIG. **5** receives metrics in the feedback loop from a testbed simulator **506**. In one embodiment, a customer's website may be analyzed according to the traditional methods of identifying acceleration techniques on the server side. The acceleration techniques may be used as a default or baseline set of techniques before beginning the runtime analysis.

The simulation testbed **506** may include a simulation engine **502** configured to mimic a client system making requests for content such as a webpage from the CDN. The simulation engine **502** may receive as inputs a plurality of different configurations **504** that may be used by real-world client systems. Each of the configurations **504** may be run by the simulation engine **502** and may provide a request to the CDN for content. From the perspective of the CDN, the request may be indistinguishable from real-world client system requests. In other words, the request may be routed to edge servers **230** and content may be routed through one or more intermediate servers **302** where acceleration techniques may be selected based on the attributes of the particular configuration under simulation.

A configuration may include the attributes that are used by the intermediate servers **302** to select from among the available acceleration techniques. For example, a configuration may include attributes that indicate a particular browser type or version. A configuration may also include attributes that indicate a particular type of computer system. A configuration may also include attributes that include network latency, network type, ISP, or mobile provider. A configuration may also include hardware specifications, such as a screen size, memory size, processing power, and/or the like. A configuration may also include an attribute indicating whether the client system is using mobile cellular technology or a fixed landline device. A configuration may also include an attribute indicating client-side acceleration techniques that may be used by the client system. In short, configurations may include any or all of the attributes described herein that may be used by the intermediate server **302** to select acceleration techniques.

When the simulation testbed **506** receives the content from the CDN, the simulation engine **502** may download the content and render the content as it would be rendered on a corresponding real-world client system. One advantage of the simulation testbed **506** over using metrics collected from real world client systems is that a more complete set of metrics may be available. For example, the simulation testbed **506** may collect precise loading times, as well as precise image locations and granular timing data for each content object loaded in a webpage.

In many cases, customers of the CDN may supply their website and content objects to the CDN to be heuristically optimized using the simulation testbed **506**. Each intermediate server **302** may be loaded with a lookup file that is populated using the results of the simulation testbed **506**. The lookup file may be indexed using an identifier for the particular website or content object and the attributes associated with the content request. The lookup file may then return a particular set of acceleration techniques that have been identified as optimal for the particular client system configuration.

FIG. **6**A illustrates a lookup file for determining the optimal set of acceleration techniques based on attributes associated with a content request, according to some embodiments. A memory element, such as a database **602**, may be configured to store instructions that cause the processor to execute one or more acceleration functions **604**. Each acceleration function may be independent of the others such that multiple acceleration functions may be applied to a single content object. Additionally, some acceleration functions may be dependent on others such that certain acceleration function should be paired or executed in a certain order.

A look up table **606** may be comprised of a multidimensional data structure, such as a data cube or multidimensional array. Lookup table **606** may include a dimension corresponding to each attribute use by the intermediate server(s) to select acceleration functions. In this example, the lookup table **606** includes at least three dimensions. The first dimension may correspond to to a browser type **612**. The second dimension may correspond to a latency measurement **608**. The third dimension may correspond to a screen size **610**. It will be understood that these three dimensions are merely exemplary and not meant to be limiting. In practice, many other dimensions may also be used that correspond to attributes associated with the content request.

In some embodiments, one dimension of the lookup table **606** may represent detected proxy configurations associated with the client system. For example, some embodiments may

detect whether a client system is behind a transparent proxy and use this information to select acceleration techniques. Proxies may apply their own acceleration techniques or filter unrecognized or unexpected data from their data streams. Therefore, understanding proxy information related to a client system can be beneficial in selecting acceleration techniques. In some cases, proxies may control access to certain geographic locations such as countries like China. Acceleration techniques may be selected based on the behavior of these control proxies, and take into account any access controls or data filtering operations conducted by the proxies.

In some cases, dimensions may be subdivided into discrete values. For example, the browser type **612** may be divided into discrete values, each of which corresponds to a particular browser type, such as Microsoft™ Internet Explorer, Google™ Chrome, Mozilla™ Firefox, and/or the like. In other cases, dimensions representing continuous or near continuous values may be quantized into discrete buckets. For example, the latency measurement **608** may be received as near-continuous time values. Depending on the number of subdivisions within this dimension, the time domain may be quantized into a series of buckets. For example, Latency 1 may represent 0 to 100 ms, Latency 2 may represent 100 ms to 200 ms. Latency 3 may represent more than 200 ms, and so forth.

Each entry in the lookup table **606** may include an indication of which acceleration functions should be used for the corresponding set of attributes. When content responsive to a request is received by the intermediate server, the attributes associated with the request may be used to index a set of acceleration functions in the lookup table **606**. For example, a request associated with Browser 2, Latency 3, and Screen 2 may correspond to acceleration functions 1, 2, and 3. Each entry in the lookup table **606** may include a designation of acceleration functions, as well as an order in which they should be applied.

FIG. **6**B illustrates a lookup file using acceleration profiles, according to some embodiments. Such acceleration techniques selected from the plurality of acceleration techniques may be grouped together to form acceleration profiles. Instead of listing acceleration functions and the order and manner in which they are applied, each entry in the lookup table **606** may instead reference acceleration profile. In some embodiments, the acceleration profiles may be stored in an acceleration profile database **614**.

When a content object is initially stored in the CDN. A lookup table **606** may be constructed using attributes that are expected to affect the transmission of the content object. The lookup table **606** may be populated using default acceleration profiles that are believed to be the most effective for the particular configurations based on past experience or the predictive intuition of the designer. The default acceleration profiles may initially be used to provide the content object to client systems. Alternatively, the default profiles may be used as a starting point for operating within a simulation testbed.

As feedback metrics describing the load time and correctness of the content object are received, the lookup table **606** may be dynamically adjusted to optimize the acceleration profiles assigned to each entry in the lookup table **606**. If the received timing and/or correctness metrics indicate that the default acceleration profile is performing worse than expected, a new acceleration profile may be generated that combines acceleration functions **604** in ways that remedy the performance shortfall.

As timing and/or correctness metrics are received over time, the entries in the lookup table **606** may be continuously adjusted. Therefore, a content provider, such as a website owner, may make continuous changes to their website content and the acceleration system of the CDN will automatically adjust itself to continually optimize the acceleration techniques applied for each client system or request type. The content provider need not inform the CDN when changes are made to the website, nor does the CDN need to detect when changes are made to the website. After a website change, the received metrics may indicate that some of the acceleration profiles in the lookup table **606** are no longer optimal, and adjustments can be made on the fly.

In addition to updating acceleration profiles, the lookup table **606** itself may also be restructured. Attribute categories may become irrelevant as technology evolves and may be substituted, added, or removed from the lookup table. For example, screen size may may play a smaller role in acceleration technique performance in some embodiments. The screen size entry **610** in the lookup table **606** may be replaced with another attribute type, such as geolocation, network type, and/or the like.

In some embodiments, different versions of a lookup table may be available for particular customers. For example a customer account may be associated with a level of service that the user has chosen to pay for. More expensive service levels may offer a more expansive set of acceleration techniques or acceleration profiles that combine acceleration techniques in unique ways or allow for more acceleration techniques to be applied.

FIG. **7**A illustrates a flowchart of a process for selecting acceleration techniques based on attributes associated with the content request, according to some embodiments. This process may be carried out by an intermediate server, or acceleration server. Alternatively, this process may be carried out by an edge server that is configured to apply acceleration functions to content as it is provided to a client system. The method may include accessing content responsive to a content request (**702**). If the content is provided by an edge server to an intermediate server, the intermediate server may access the content as it is provided by the edge server. Alternatively, if the edge server is performing this function, then the content may be accessed as it is stored at the edge server and provided to the client system.

The method may also include receiving attributes associated with the content request (**704**). The attributes may be provided as part of the content request. The attribute may also be provided directly by the client system in a separate transaction from the content request. The attributes may also be stored locally on the intermediate server or the edge server. For example, a content request may be identified using an IP address. The IP address may then be used to look up a set of attributes associated with content requests from the requesting client system. The stored attributes may be been provided in the past during a registration procedure, or in association with prior content requests. Some attributes may also be received from third parties, such as entities that provide business information, connection information, and/or latency associated with a particular IP address. As described above, the attributes may include information descriptive of a screen size, a network type, a device type, a latency measure, a geolocation, and/or any other type of information that may affect how the selected acceleration techniques perform.

The method may also include selecting acceleration techniques based on the attributes (**706**). In some embodiments, the attributes may be used to select a set of acceleration techniques stored as an acceleration profile. In some embodiments, the selection acceleration techniques may be based on information in addition to the attributes. For example, network load statistics, load balancing requirements, storage

space availability, and/or other characteristics of the CDN may be used to select acceleration techniques.

The method may further include providing content to the client system using the selected acceleration techniques (**708**). The edge server and/or intermediate server may process the content according to acceleration techniques. In some cases, this may involve creating a new version of the content object, such as an HTML source file, that is optimized according to one or more of the selected acceleration techniques. Therefore, multiple versions of the content object may be stored, each of which corresponds to one or more acceleration profiles associated with the original content object.

Where a feedback loop is operational, the method may additionally include receiving metrics that are descriptive of the timing and/or correctness with which the content object was received and loaded by the client system (**710**). Any of the techniques described above herein may be used to receive these metrics. The method may further include updating the acceleration technique selection process based on the metrics (**712**). This step may include adding or replacing acceleration profiles in a lookup table. This step may also include changing the attributes that are used in the selection process. In some cases, this step may include creating new pre-cached content objects that correspond to new acceleration profiles.

FIG. **7**B illustrates a method of optimizing acceleration profiles based on attributes and metrics, according to some embodiments. The method may include assigning acceleration profiles to different attribute combinations (**720**). These acceleration profiles may be comprised of one or more acceleration functions, or acceleration techniques. Each acceleration profile may also include instructions describing how the acceleration technique should be implemented, including an order in which they should be implemented. The acceleration profiles may populate a lookup table that is indexed by the various attribute combinations. These initial acceleration profiles may be default profiles, and may be assigned based on historical information stored and analyzed within the CDN. The initial acceleration profiles may also be assigned based on administrator preferences, client system preferences, or user subscription account levels.

In some embodiments, the method may also include inserting new acceleration techniques into one or more of the acceleration profiles or entries in a lookup table. This method may be useful in testing the effectiveness of new acceleration techniques in various client system configurations. For example, a new acceleration technique may be inserted into each of the lookup table entries in order to analyze its performance in each system configuration type. In some cases, testing new acceleration techniques can be limited to a testing environment and restricted from live use with real client systems. Therefore, the method may include detecting whether a testing environment is operational, and selectively activating new acceleration techniques while the testing environment is active. The method may also include detecting when real client systems are being serviced by the intermediate server, and selectively deactivating new acceleration techniques accordingly.

The method may also include providing content using the acceleration profile (**722**). The content may be provided to actual requesting client systems. Alternatively, the content may be provided to a simulation test bed that simulates various client system configurations and network connections for the purpose of optimizing the process for selecting among the various acceleration techniques. In some cases, simulation data may be combined with real-world data in the optimization process.

The method may further include receiving timing and/or correctness metrics (**724**). As described above, these metrics may include any information that indicates timing events associated with loading the content object on the client system. These metrics may also include information describing how correctly the content object was rendered, displayed, or made available by the client system. These metrics may be received around the time when the content is provided to the client system or testbed simulator. Alternatively, these metrics may also be received later after the client system or testbed simulator has finished processing the requested content.

The method may additionally include determining whether the profiles are optimally assigned (**726**). This determination may be made by comparing the receiving metrics to threshold values. These threshold values may be expected values determined statistically for each combination of acceleration techniques. The threshold values may also be based on historical data collected and analyzed within the CDN. If the metrics fall within an acceptable range in relation to the threshold values, the method may again provide content using the acceleration profiles in response to different attribute combinations provided by the simulation testbed or client system. Alternatively, if it is determined that the acceleration profiles are not optimally assigned, the acceleration profile under test may be updated or reassigned in order to provide a better combination of acceleration techniques (**728**). The acceleration profile under test may be edited according to a sequence of possible acceleration techniques determined experimentally using historical and/or statistical data from within the CDN.

Referring next to FIG. **8**, an exemplary environment with which embodiments may be implemented is shown with a computer system **800** that can be used by a user **804** to program, design or otherwise interact with the computer system **800**. The computer system **800** can include a computer **802**, keyboard **822**, a network router **812**, a printer **808**, and a monitor **806**. The monitor **806**, processor **802** and keyboard **822** are part of a computer system **826**, which can be a laptop computer, desktop computer, handheld computer, mainframe computer, etc. The monitor **806** can be a CRT, flat screen, etc.

A user **804** can input commands into the computer **802** using various input devices, such as a mouse, keyboard **822**, track ball, touch screen, etc. If the computer system **800** comprises a mainframe, a user **804** can access the computer **802** using, for example, a terminal or terminal interface. Additionally, the computer system **826** may be connected to a printer **808** and a server **810** using a network router **812**, which may connect to the Internet **818** or a WAN.

The server **810** may, for example, be used to store additional software programs and data. In one embodiment, software implementing the systems and methods described herein can be stored on a storage medium in the server **810**. Thus, the software can be run from the storage medium in the server **810**. In another embodiment, software implementing the systems and methods described herein can be stored on a storage medium in the computer **802**. Thus, the software can be run from the storage medium in the computer system **826**. Therefore, in this embodiment, the software can be used whether or not computer **802** is connected to network router **812**. Printer **808** may be connected directly to computer **802**, in which case, the computer system **826** can print whether or not it is connected to network router **812**.

With reference to FIG. **9**, an embodiment of a special-purpose computer system **900** is shown. The above methods may be implemented by computer-program products that direct a computer system to perform the actions of the above-described methods and components. Each such computer-

program product may comprise sets of instructions (codes) embodied on a computer-readable medium that directs the processor of a computer system to perform corresponding actions. The instructions may be configured to run in sequential order, or in parallel (such as under different processing threads), or in a combination thereof. After loading the computer-program products on a general purpose computer system **926**, it is transformed into the special-purpose computer system **900**.

Special-purpose computer system **900** comprises a computer **802**, a monitor **806** coupled to computer **802**, one or more additional user output devices **930** (optional) coupled to computer **802**, one or more user input devices **940** (e.g., keyboard, mouse, track ball, touch screen) coupled to computer **802**, an optional communications interface **950** coupled to computer **802**, a computer-program product **905** stored in a tangible computer-readable memory in computer **802**. Computer-program product **905** directs system **900** to perform the above-described methods. Computer **802** may include one or more processors **960** that communicate with a number of peripheral devices via a bus subsystem **990**. These peripheral devices may include user output device(s) **930**, user input device(s) **940**, communications interface **950**, and a storage subsystem, such as random access memory (RAM) **970** and non-volatile storage drive **980** (e.g., disk drive, optical drive, solid state drive), which are forms of tangible computer-readable memory.

Computer-program product **905** may be stored in non-volatile storage drive **980** or another computer-readable medium accessible to computer **802** and loaded into memory **970**. Each processor **960** may comprise a microprocessor, such as a microprocessor from Intel® or Advanced Micro Devices, Inc.®, or the like. To support computer-program product **905**, the computer **802** runs an operating system that handles the communications of product **905** with the above-noted components, as well as the communications between the above-noted components in support of the computer-program product **905**. Exemplary operating systems include Windows® or the like from Microsoft® Corporation, Solaris® from Oracle®, LINUX, UNIX, and the like.

User input devices **940** include all possible types of devices and mechanisms to input information to computer system **802**. These may include a keyboard, a keypad, a mouse, a scanner, a digital drawing pad, a touch screen incorporated into the display, audio input devices such as voice recognition systems, microphones, and other types of input devices. In various embodiments, user input devices **940** are typically embodied as a computer mouse, a trackball, a track pad, a joystick, wireless remote, a drawing tablet, a voice command system. User input devices **940** typically allow a user to select objects, icons, text and the like that appear on the monitor **906** via a command such as a click of a button or the like. User output devices **930** include all possible types of devices and mechanisms to output information from computer **802**. These may include a display (e.g., monitor **906**), printers, non-visual displays such as audio output devices, etc.

Communications interface **950** provides an interface to other communication networks and devices and may serve as an interface to receive data from and transmit data to other systems, WANs and/or the Internet **918**. Embodiments of communications interface **950** typically include an Ethernet card, a modem (telephone, satellite, cable, ISDN), a (asynchronous) digital subscriber line (DSL) unit, a FireWire® interface, a USB® interface, a wireless network adapter, and the like. For example, communications interface **950** may be coupled to a computer network, to a FireWire® bus, or the like. In other embodiments, communications interface **950** may be physically integrated on the motherboard of computer **802**, and/or may be a software program, or the like.

RAM **970** and non-volatile storage drive **980** are examples of tangible computer-readable media configured to store data such as computer-program product embodiments of the present invention, including executable computer code, human-readable code, or the like. Other types of tangible computer-readable media include floppy disks, removable hard disks, optical storage media such as CD-ROMs, DVDs, bar codes, semiconductor memories such as flash memories, read-only-memories (ROMs), battery-backed volatile memories, networked storage devices, and the like. RAM **970** and non-volatile storage drive **980** may be configured to store the basic programming and data constructs that provide the functionality of various embodiments of the present invention, as described above.

Software instruction sets that provide the functionality of the present invention may be stored in RAM **970** and non-volatile storage drive **980**. These instruction sets or code may be executed by the processor(s) **960**. RAM **970** and non-volatile storage drive **980** may also provide a repository to store data and data structures used in accordance with the present invention. RAM **970** and non-volatile storage drive **980** may include a number of memories including a main random access memory (RAM) to store of instructions and data during program execution and a read-only memory (ROM) in which fixed instructions are stored. RAM **970** and non-volatile storage drive **980** may include a file storage subsystem providing persistent (non-volatile) storage of program and/or data files. RAM **970** and non-volatile storage drive **980** may also include removable storage systems, such as removable flash memory.

Bus subsystem **990** provides a mechanism to allow the various components and subsystems of computer **802** communicate with each other as intended. Although bus subsystem **990** is shown schematically as a single bus, alternative embodiments of the bus subsystem may utilize multiple busses or communication paths within the computer **802**.

A number of variations and modifications of the disclosed embodiments can also be used. For example, embodiments show the DNS function being resident within a POP with edge servers, but other embodiments could place the DNS function geographically separate from any content serving functions. Other embodiments could place multiple DNS functions in a POP to divide the work load for those DNS requests received by the POP.

Specific details are given in the above description to provide a thorough understanding of the embodiments. However, it is understood that the embodiments may be practiced without these specific details. For example, circuits may be shown in block diagrams in order not to obscure the embodiments in unnecessary detail. In other instances, well-known circuits, processes, algorithms, structures, and techniques may be shown without unnecessary detail in order to avoid obscuring the embodiments.

Implementation of the techniques, blocks, steps and means described above may be done in various ways. For example, these techniques, blocks, steps and means may be implemented in hardware, software, or a combination thereof. For a hardware implementation, the processing units may be implemented within one or more application specific integrated circuits (ASICs), digital signal processors (DSPs), digital signal processing devices (DSPDs), programmable logic devices (PLDs), field programmable gate arrays (FPGAs), processors, controllers, micro-controllers, microprocessors, other electronic units designed to perform the functions described above, and/or a combination thereof.

Also, it is noted that the embodiments may be described as a process which is depicted as a flowchart, a flow diagram, a swim diagram, a data flow diagram, a structure diagram, or a block diagram. Although a depiction may describe the operations as a sequential process, many of the operations can be performed in parallel or concurrently. In addition, the order of the operations may be re-arranged. A process is terminated when its operations are completed, but could have additional steps not included in the figure. A process may correspond to a method, a function, a procedure, a subroutine, a subprogram, etc. When a process corresponds to a function, its termination corresponds to a return of the function to the calling function or the main function.

Furthermore, embodiments may be implemented by hardware, software, scripting languages, firmware, middleware, microcode, hardware description languages, and/or any combination thereof. When implemented in software, firmware, middleware, scripting language, and/or microcode, the program code or code segments to perform the necessary tasks may be stored in a machine readable medium such as a storage medium. A code segment or machine-executable instruction may represent a procedure, a function, a subprogram, a program, a routine, a subroutine, a module, a software package, a script, a class, or any combination of instructions, data structures, and/or program statements. A code segment may be coupled to another code segment or a hardware circuit by passing and/or receiving information, data, arguments, parameters, and/or memory contents. Information, arguments, parameters, data, etc. may be passed, forwarded, or transmitted via any suitable means including memory sharing, message passing, token passing, network transmission, etc.

For a firmware and/or software implementation, the methodologies may be implemented with modules (e.g., procedures, functions, and so on) that perform the functions described herein. Any machine-readable medium tangibly embodying instructions may be used in implementing the methodologies described herein. For example, software codes may be stored in a memory. Memory may be implemented within the processor or external to the processor. As used herein the term "memory" refers to any type of long term, short term, volatile, nonvolatile, or other storage medium and is not to be limited to any particular type of memory or number of memories, or type of media upon which memory is stored.

Moreover, as disclosed herein, the term "storage medium" may represent one or more memories for storing data, including read only memory (ROM), random access memory (RAM), magnetic RAM, core memory, magnetic disk storage mediums, optical storage mediums, flash memory devices and/or other machine readable mediums for storing information. The term "machine-readable medium" includes, but is not limited to portable or fixed storage devices, optical storage devices, and/or various other storage mediums capable of storing that contain or carry instruction(s) and/or data.

While the principles of the disclosure have been described above in connection with specific apparatuses and methods, it is to be clearly understood that this description is made only by way of example and not as limitation on the scope of the disclosure.

What is claimed is:

1. A system for dynamically selecting from among a plurality of acceleration techniques implemented in a Content Delivery Network (CDN) using attributes associated with content requests, comprising:

a network interface that receives the content requests from a plurality of client systems for content stored in a plurality of edge servers distributed geographically throughout the CDN, wherein:

the content requests comprise a first content request for first content;

the first content request originates from a first client system in the plurality of client systems; and

the first content request is associated with one or more attributes; and

an intermediate server that accelerates access to the content stored in the plurality of edge servers, the intermediate server comprising:

a first interface coupled to the network interface;

a second interface configured to communicate with at least one of the plurality of edge servers of the CDN;

one or more memory devices having stored thereon:

instructions for executing each of the plurality of acceleration techniques; and

a plurality of acceleration profiles, wherein each of the plurality of acceleration profiles specifies at least one of the plurality of acceleration techniques; and

a processor configured to:

access the one or more attributes associated with the first content request;

select one or more acceleration techniques from the plurality of acceleration techniques, wherein:

the one or more acceleration techniques are selected based on the one or more attributes; and

the one or more acceleration techniques modify the content;

use the one or more acceleration techniques to provide the content to the first client system;

receive metrics from the first client system, wherein the metrics are associated with a performance in providing the first content to a user of the first client system;

dynamically update a process by which the one or more acceleration techniques are selected based on the metrics; and

use the updated process to select acceleration techniques for subsequent requests associated with similar attributes.

2. The system for dynamically selecting from among the plurality of acceleration techniques implemented in the CDN using the attributes associated with the content requests of claim **1**, wherein the one or more attributes comprises a geographic location of the first client system.

3. The system for dynamically selecting from among the plurality of acceleration techniques implemented in the CDN using the attributes associated with the content requests of claim **1**, wherein the intermediate server is physically combined with one of the plurality of edge servers of the CDN.

4. The system for dynamically selecting from among the plurality of acceleration techniques implemented in the CDN using the attributes associated with the content requests of claim **1**, wherein the one or more acceleration techniques selected by the intermediate server are selected as one of the plurality of acceleration profiles.

5. The system for dynamically selecting from among the plurality of acceleration techniques implemented in the CDN using the attributes associated with the content requests of claim **1**, wherein the first client system comprises a testbed simulator that is configured to simulate a plurality of client system configurations and network connections.

6. The system for dynamically selecting from among the plurality of acceleration techniques implemented in the CDN using the attributes associated with the content requests of

claim 1, wherein the one or more attributes are received by the intermediate server separately from the first content request.

7. The system for dynamically selecting from among the plurality of acceleration techniques implemented in the CDN using the attributes associated with the content requests of claim 1, wherein the one or more attributes are received by the intermediate server together with the first content request.

8. The system for dynamically selecting from among the plurality of acceleration techniques implemented in the CDN using the attributes associated with the content requests of claim 1, wherein a first acceleration technique in the plurality of acceleration techniques is dependent on a second acceleration techniques in the plurality of acceleration techniques such that the first acceleration technique and the second acceleration technique are executed in a predetermined order.

9. The system for dynamically selecting from among the plurality of acceleration techniques implemented in the CDN using the attributes associated with the content requests of claim 1, wherein each of the plurality of acceleration profiles comprises a multidimensional data structure, wherein each dimension of the multidimensional data structure is associated with a corresponding one of the one or more attributes.

10. The system for dynamically selecting from among the plurality of acceleration techniques implemented in the CDN using the attributes associated with the content requests of claim 1, wherein the plurality of acceleration techniques comprises preemptively compressing media files to be transmitted to the client device.

11. The system for dynamically selecting from among the plurality of acceleration techniques implemented in the CDN using the attributes associated with the content requests of claim 1, wherein the plurality of acceleration techniques comprises optimizing code associated with the first content.

12. The system for dynamically selecting from among the plurality of acceleration techniques implemented in the CDN using the attributes associated with the content requests of claim 1, wherein the plurality of acceleration techniques comprises identifying portions of the content that are dynamic and caching portions of the content that are static.

13. The system for dynamically selecting from among the plurality of acceleration techniques implemented in the CDN using the attributes associated with the content requests of claim 1, wherein the processor is further configured to dynamically select one or more acceleration techniques from the plurality of acceleration techniques based attributes for each request received by the intermediate server.

14. A method of dynamically selecting from among a plurality of acceleration techniques implemented in a Content Delivery Network (CDN) using attributes associated with content requests, the method comprising:

receiving a first content request for first content through a network interface, wherein:

the network interface is configured to receive the content requests from a plurality of client systems for content stored in a plurality of edge servers distributed geographically throughout the CDN;

the first content request originates from a first client system in the plurality of client systems;

the first content request is associated with one or more attributes;

accessing, by an intermediate server, the one or more attributes associated with the first content request, wherein:

the intermediate server is configured to accelerate access to the content stored in the plurality of edge servers;

selecting, by the intermediate server, one or more acceleration techniques from the plurality of acceleration techniques, wherein:

the one or more acceleration techniques are selected based on the one or more attributes;

the one or more acceleration techniques modify the content; and

the one or more acceleration techniques form a first acceleration profile in a plurality of acceleration profiles;

using, by the intermediate server, the one or more acceleration techniques to provide the content to the first client system

receiving, by the intermediate server, metrics from the first client system, wherein the metrics are associated with a performance in providing the first content to a user of the first client system;

dynamically updating, by the intermediate server, a process by which the one or more acceleration techniques are selected based on the metrics; and

using, by the intermediate server, the updated process to select acceleration techniques for subsequent requests associated with similar attributes.

15. The method of dynamically selecting from among the plurality of acceleration techniques implemented in the CDN using the attributes associated with the content requests of claim 14, wherein the one or more attributes affect a performance of the one or more acceleration techniques in providing the first content to a user of the first client system.

16. The method of dynamically selecting from among the plurality of acceleration techniques implemented in the CDN using the attributes associated with the content requests of claim 14, wherein the one or more attributes are descriptive of the first client system or a network through which the first content request is received.

17. The method of dynamically selecting from among the plurality of acceleration techniques implemented in the CDN using the attributes associated with the content requests of claim 14, wherein the one or more attributes comprises a measurement of latency associated with transmitting content from the CDN to the first client system.

18. The method of dynamically selecting from among the plurality of acceleration techniques implemented in the CDN using the attributes associated with the content requests of claim 14, wherein the one or more attributes comprises a device type of the first client system.

19. The method of dynamically selecting from among the plurality of acceleration techniques implemented in the CDN using the attributes associated with the content requests of claim 14, wherein the first client system comprises a testbed simulator that is configured to simulate a plurality of client system configurations and network connections.

* * * * *

Exhibit F

US008615577B2

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

(10) **Patent No.: US 8,615,577 B2**
(45) **Date of Patent: Dec. 24, 2013**

(54) **POLICY BASED PROCESSING OF CONTENT OBJECTS IN A CONTENT DELIVERY NETWORK USING MUTATORS**

(75) Inventors: **Bryan Black**, Tempe, AZ (US); **Jacob S. Roersma**, Grand Rapids, MI (US); **Jared Boelens**, Tempe, AZ (US); **Luke Knol**, Tempe, AZ (US); **Neil Dunbar**, Bristol (GB); **Sig Lange**, Tempe, AZ (US); **Wylie Swanson**, Tempe, AZ (US)

(73) Assignee: **Limelight Networks, Inc.**, Tempe, AZ (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 57 days.

(21) Appl. No.: **13/336,915**

(22) Filed: **Dec. 23, 2011**

(65) **Prior Publication Data**

US 2012/0198041 A1 Aug. 2, 2012

Related U.S. Application Data

(63) Continuation of application No. PCT/US2011/023410, filed on Feb. 1, 2011.

(51) **Int. Cl.**
G06F 15/173 (2006.01)

(52) **U.S. Cl.**
USPC **709/223**; 709/201; 709/225; 709/232

(58) **Field of Classification Search**
USPC 709/201–203, 217–219, 223–226, 709/231–232
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,064,968 A 5/2000 Schanz
6,658,467 B1 12/2003 Rice et al.
6,757,740 B1 6/2004 Parekh et al.
6,792,399 B1 9/2004 Phillips et al.
6,918,120 B2 7/2005 Riedel
6,952,737 B1 10/2005 Coates et al.
7,072,863 B1 7/2006 Phillips et al.
(Continued)

FOREIGN PATENT DOCUMENTS

KR 10-2002-0076028 A 10/2002
WO WO 2009/061829 A1 5/2009

OTHER PUBLICATIONS

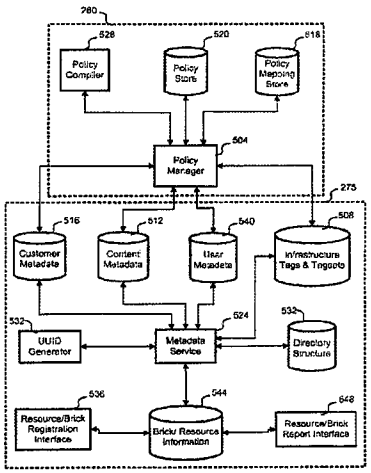
U.S. Appl. No. 13/336,831, Final Office Action mailed Feb. 11, 2013, 12 pages.
(Continued)

Primary Examiner — Bharat N Barot
(74) *Attorney, Agent, or Firm* — Kilpatrick Townsend & Stockton LLP

(57) **ABSTRACT**

A method for processing content objects with resources associated with a content delivery network (CDN) having a plurality of geographically distributed points of presence (POPs) is disclosed. The resources are enrolled to be accessible from the CDN. Each resource is categorized using tags that categorize the resources. Selection of a policy from a plurality of policies is received, where the plurality of policies define processes to perform on content objects. The selected policy includes an applicability criteria and a call to the resource. Metadata is received at the CDN, the metadata being related to a content object, a requester of the content object and/or a provider of the content object. It is determined that the policy is applicable through analysis of the metadata and/or applicability criteria. The resource is called according to the call in the policy to cause the resource to perform specified processing on the content object.

20 Claims, 19 Drawing Sheets



(56) **References Cited**

U.S. PATENT DOCUMENTS

7,136,922	B2	11/2006	Sundaram et al.	
7,266,556	B1	9/2007	Coates	
7,272,654	B1	9/2007	Brendel	
7,376,727	B2	5/2008	Weller et al.	
7,376,736	B2	5/2008	Sundaram et al.	
7,627,391	B2	12/2009	Key	
7,664,839	B1	2/2010	Karr et al.	
7,840,667	B2	11/2010	Weller et al.	
7,860,964	B2	12/2010	Brady et al.	
7,904,541	B2	3/2011	Swildens et al.	
8,023,429	B2 *	9/2011	Briscoe et al.	709/226
8,041,809	B2	10/2011	Sundaram et al.	
8,069,182	B2	11/2011	Pieper	
8,108,507	B2	1/2012	Weller et al.	
8,151,317	B2	4/2012	Hinton et al.	
8,156,243	B2	4/2012	Richardson et al.	
8,165,122	B2	4/2012	Kotalwar et al.	
8,255,557	B2 *	8/2012	Raciborski et al.	709/219
8,291,083	B2 *	10/2012	Black et al.	709/226
2001/0034795	A1	10/2001	Moulton et al.	
2002/0091801	A1	7/2002	Lewin et al.	
2002/0133491	A1	9/2002	Sim et al.	
2002/0138652	A1	9/2002	Taylor	
2002/0147774	A1	10/2002	Lisiecki et al.	
2003/0079027	A1	4/2003	Slocombe et al.	
2003/0126233	A1	7/2003	Bryers et al.	
2003/0149581	A1	8/2003	Chaudhri et al.	
2003/0217365	A1	11/2003	Caputo	
2005/0005025	A1	1/2005	Harville et al.	
2007/0156845	A1	7/2007	Devanneaux et al.	
2007/0250468	A1	10/2007	Pieper	
2008/0065724	A1	3/2008	Seed et al.	
2008/0072264	A1	3/2008	Crayford	
2008/0077682	A1	3/2008	Nair et al.	
2008/0155086	A1	6/2008	Jensen	
2008/0155386	A1	6/2008	Jensen	
2008/0195664	A1	8/2008	Maharajh et al.	
2008/0222291	A1	9/2008	Weller et al.	
2009/0150518	A1	6/2009	Lewin et al.	
2010/0023693	A1	1/2010	Dilley et al.	
2010/0250710	A1 *	9/2010	Cadwell et al.	709/219
2010/0293185	A1	11/2010	Rosado et al.	
2010/0325264	A1	12/2010	Crowder et al.	
2011/0138394	A1	6/2011	Ravishankar et al.	
2011/0218946	A1	9/2011	Stern et al.	
2011/0252082	A1 *	10/2011	Cobb et al.	709/203
2011/0283304	A1	11/2011	Roberts et al.	
2011/0320715	A1	12/2011	Ickman et al.	
2012/0030170	A1	2/2012	Bernbo et al.	
2012/0054303	A1	3/2012	Priyadarshan et al.	
2012/0066352	A1 *	3/2012	Cadwell et al.	709/219
2012/0198070	A1	8/2012	Black et al.	
2012/0246472	A1 *	9/2012	Berengoltz et al.	713/165
2012/0303735	A1 *	11/2012	Raciborski et al.	709/212
2013/0132366	A1	5/2013	Pieper	

OTHER PUBLICATIONS

"Content delivery network", retrieved from the internet at web address: http://web.archive.org/web/20101229031436/http://en.wikipedia.org/wiki/Content_delivery_network, 6 pages.

International Search Report dated Oct. 21, 2011 for International PCT Application No. PCT/US2011/023410, 11 pages.

U.S. Appl. No. 13/336,743, Office Action mailed Oct. 15, 2012, 15 pages.

U.S. Appl. No. 13/344,263, Office Action mailed Jan. 11, 2013, 8 pages.

U.S. Appl. No. 13/341,724, Office Action mailed Jan. 4, 2013, 23 pages.

U.S. Appl. No. 13/336,831 Office Action mailed Aug. 14, 2012, 15 pages.

U.S. Appl. No. 13/336,893 Office Action mailed Aug. 13, 2012, 9 pages.

U.S. Appl. No. 13/336,831, Notice of Allowance mailed Jul. 8, 2013, 36 pages.

* cited by examiner

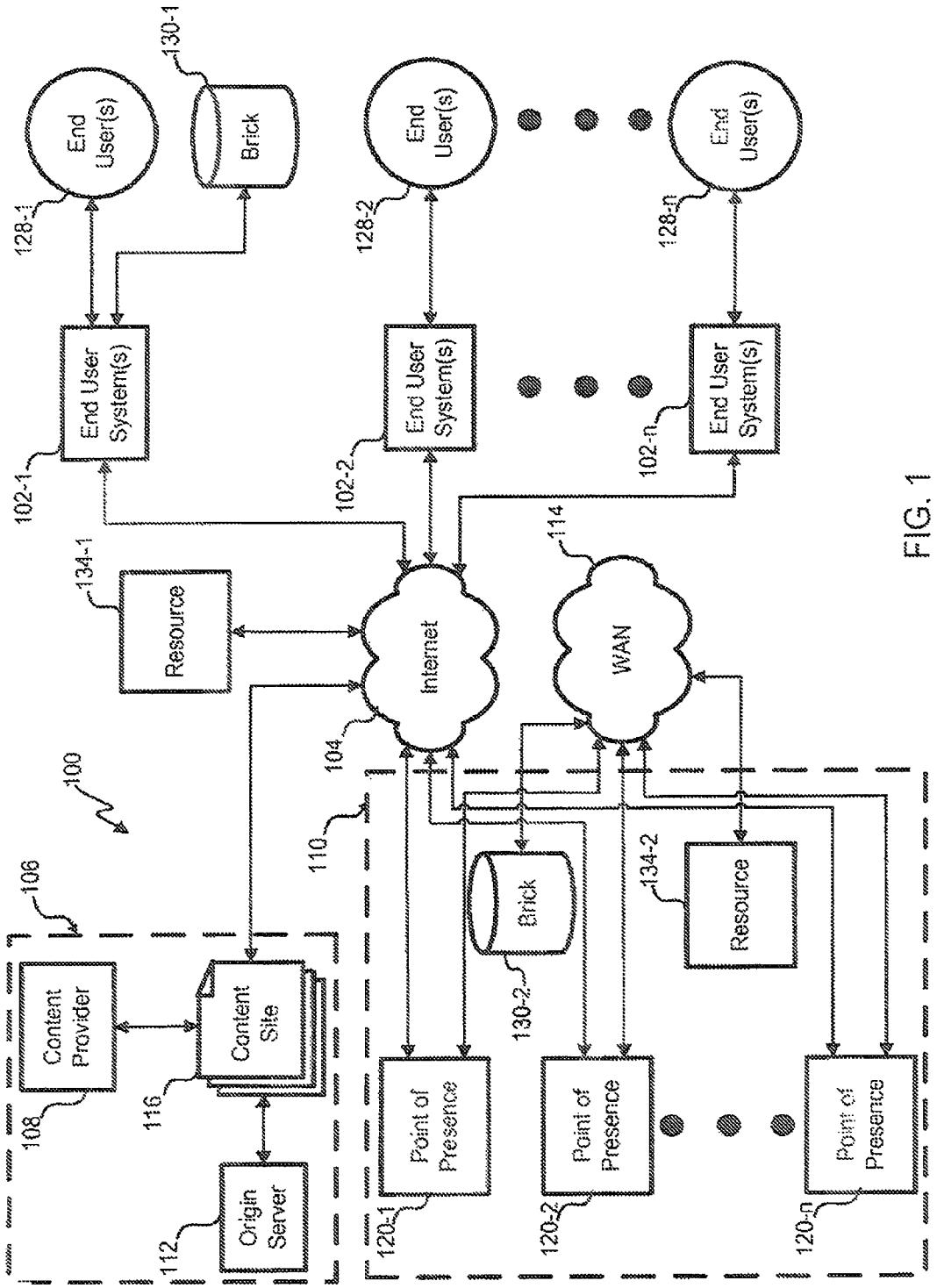

End User(s)
128-1
Brick
130-1
End User(s)
128-2
End User(s)
128-n
End User System(s)
102-1
End User System(s)
102-2
End User System(s)
102-n
Resource
134-1
Internet
104
WAN
114
100
110
106
Content Provider
108
Content Site
116
Origin Server
112
Brick
130-2
Resource
134-2
Point of Presence
120-1
Point of Presence
120-2
Point of Presence
120-n

FIG. 1

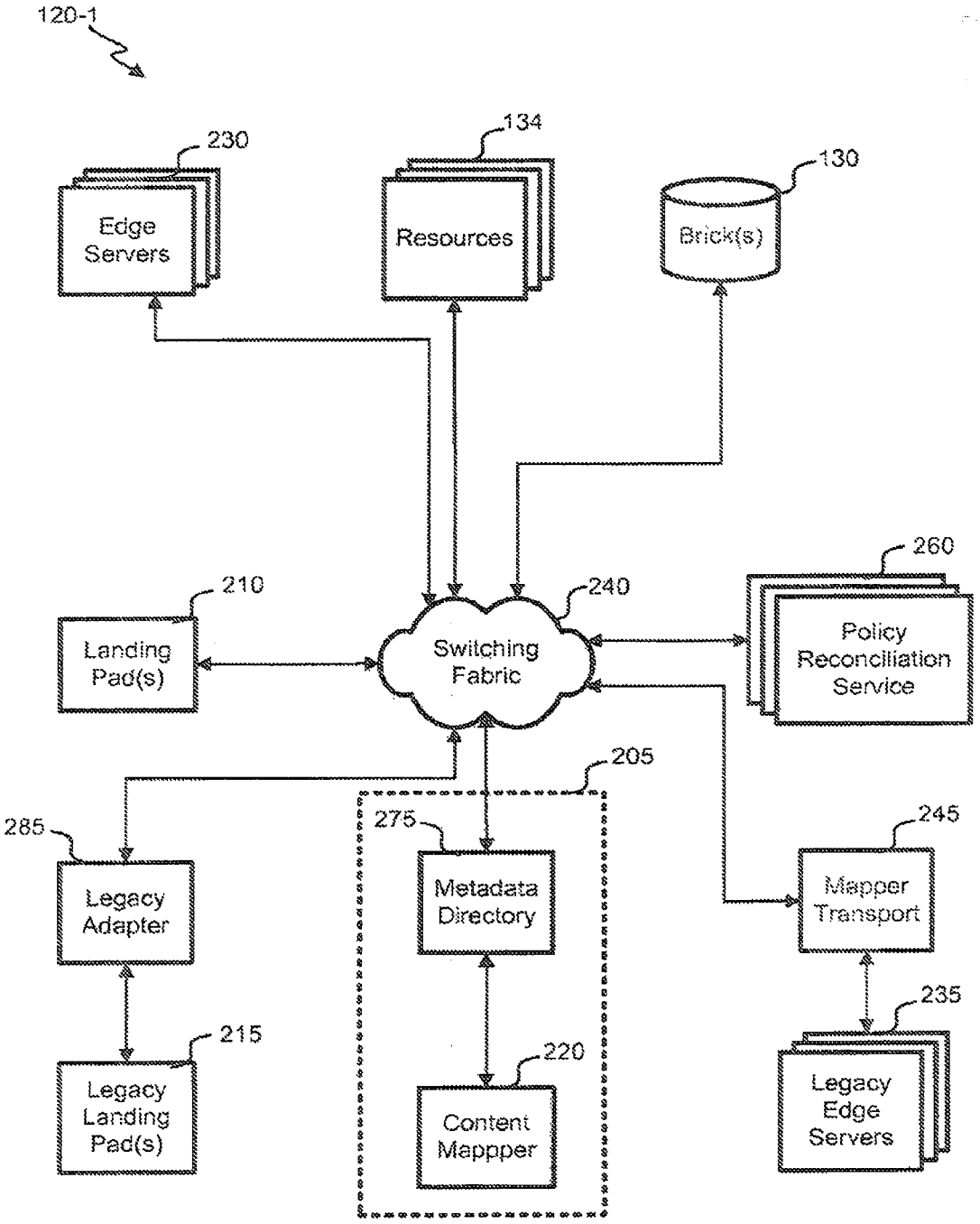

120-1
230
Edge Servers
134
Resources
130
Brick(s)
260
Policy Reconciliation Service
210
Landing Pad(s)
240
Switching Fabric
205
285
Legacy Adapter
275
Metadata Directory
245
Mapper Transport
215
Legacy Landing Pad(s)
220
Content Mappper
235
Legacy Edge Servers

FIG. 2A

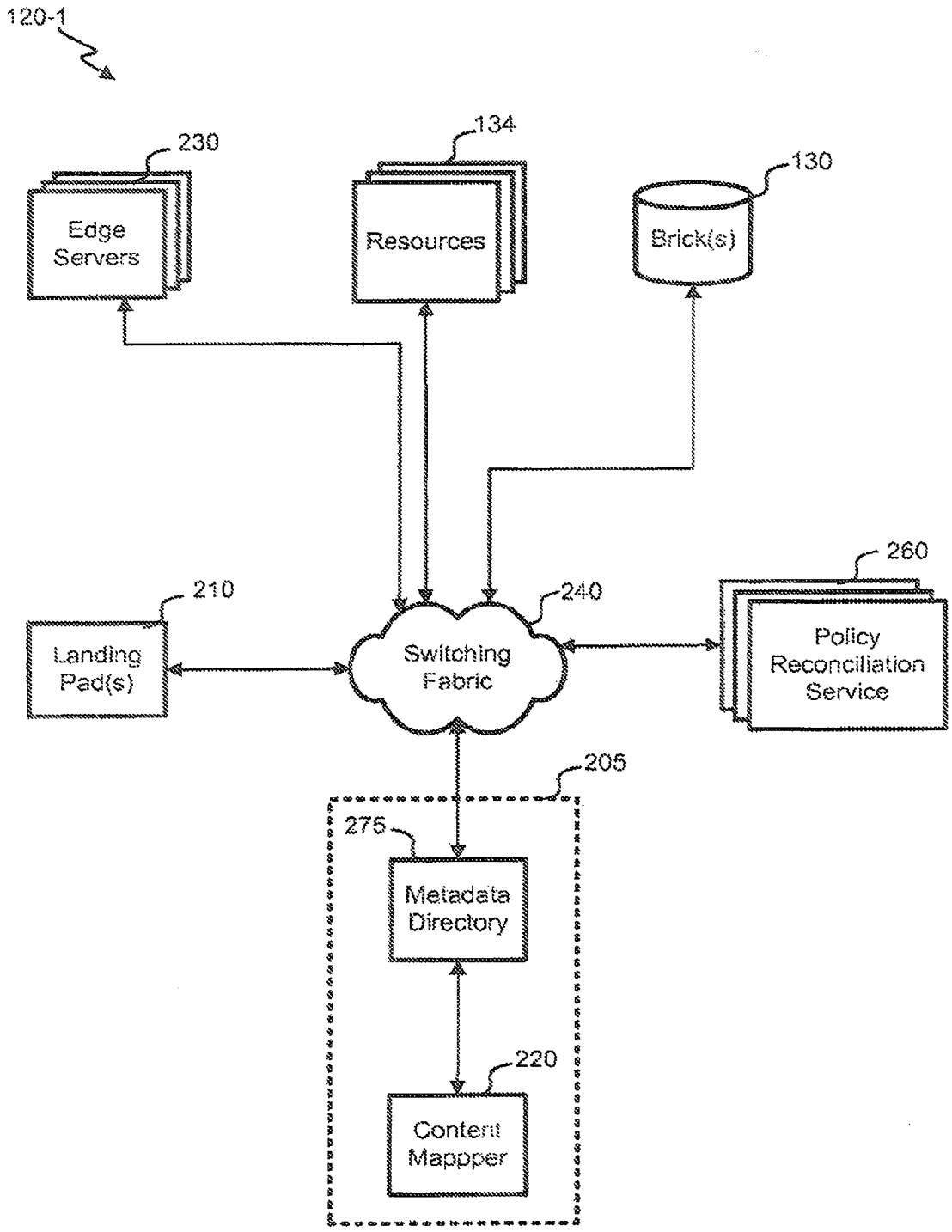

120-1
230
Edge Servers
134
Resources
130
Brick(s)
210
Landing Pad(s)
240
Switching Fabric
260
Policy Reconciliation Service
205
275
Metadata Directory
220
Content Mappper

FIG. 2B

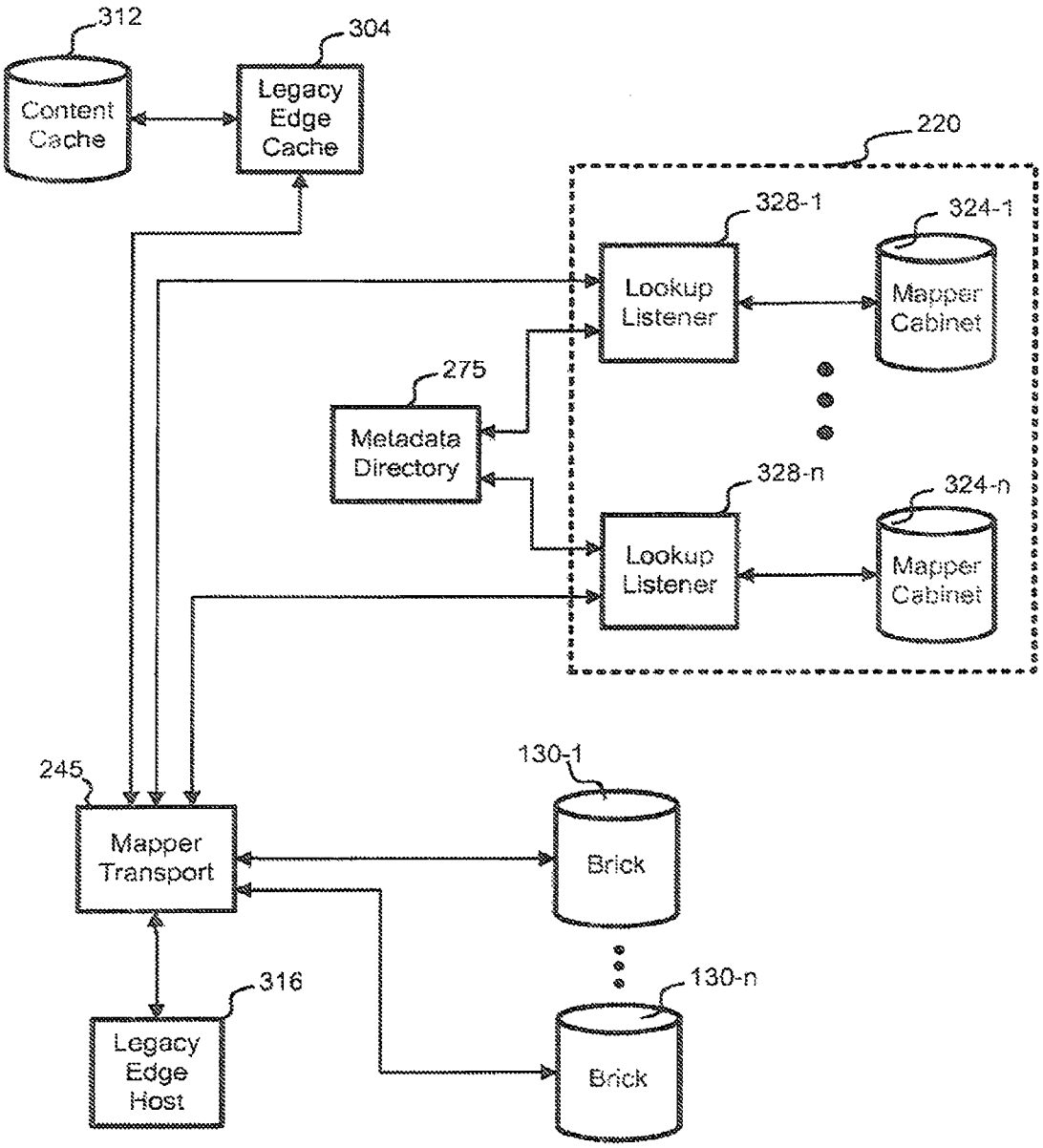

312
304
Content Cache
Legacy Edge Cache
220
328-1
324-1
Lookup Listener
Mapper Cabinet
275
Metadata Directory
328-n
324-n
Lookup Listener
Mapper Cabinet
245
130-1
Mapper Transport
Brick
316
130-n
Legacy Edge Host
Brick

FIG. 3A

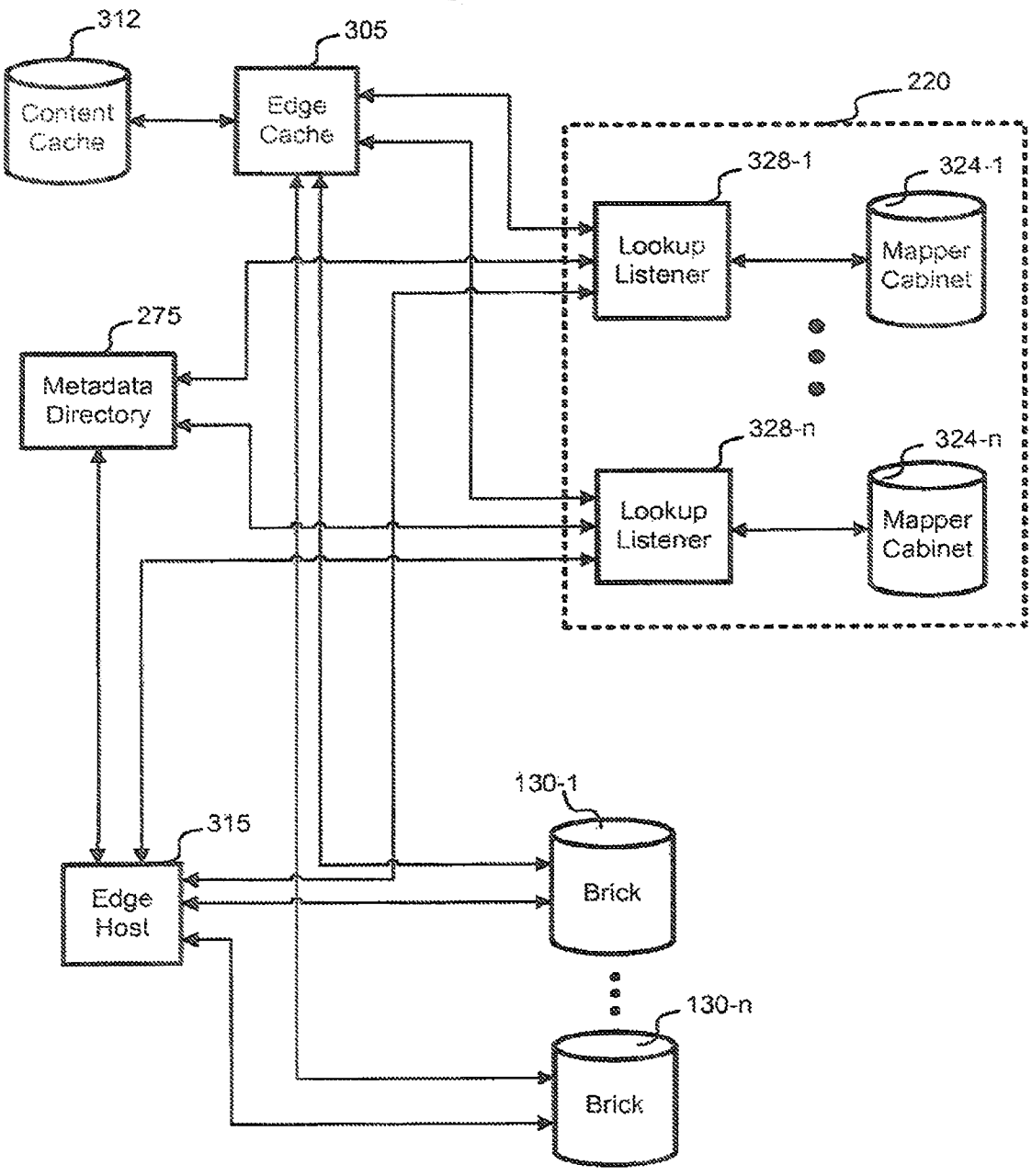

312
Content Cache
305
Edge Cache
220
328-1
Lookup Listener
324-1
Mapper Cabinet
275
Metadata Directory
328-n
Lookup Listener
324-n
Mapper Cabinet
130-1
315
Edge Host
Brick
130-n
Brick

FIG. 3B

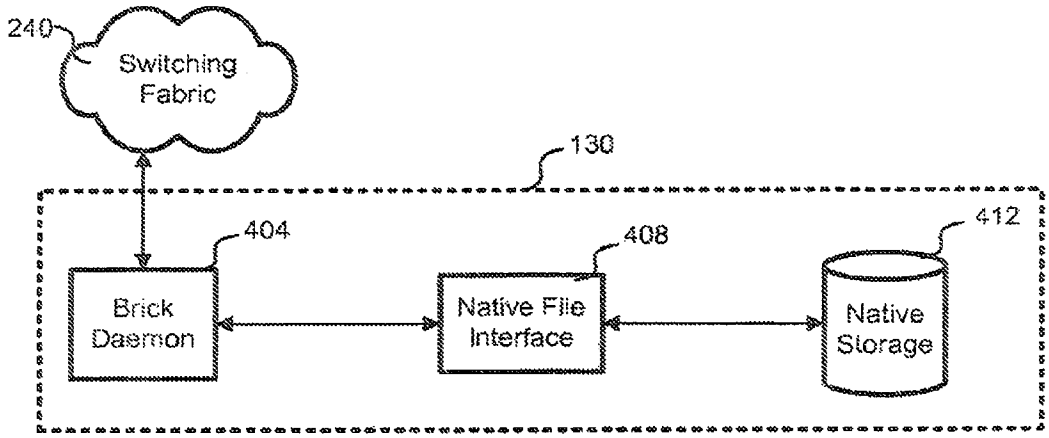


FIG. 4A

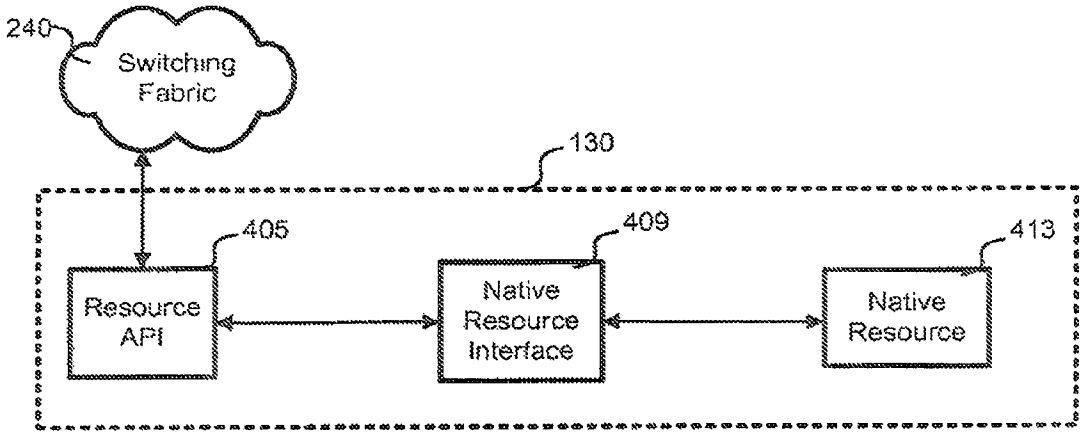


FIG. 4B

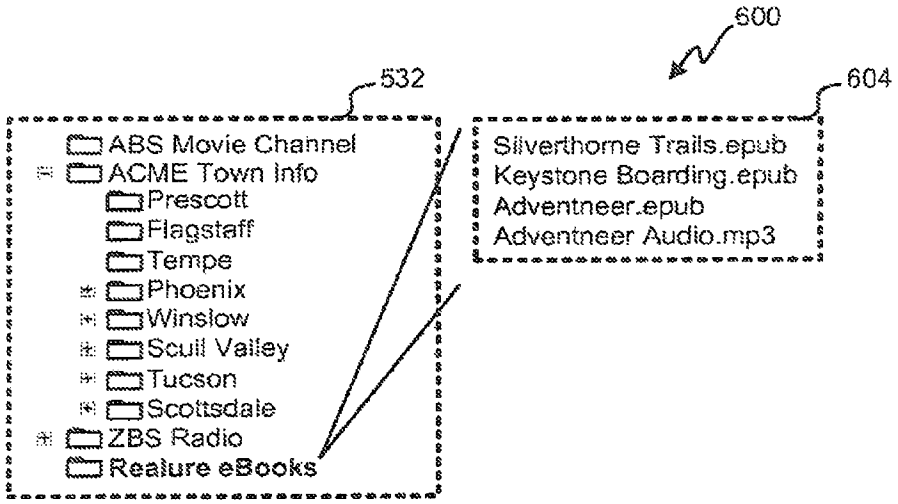


FIG. 6

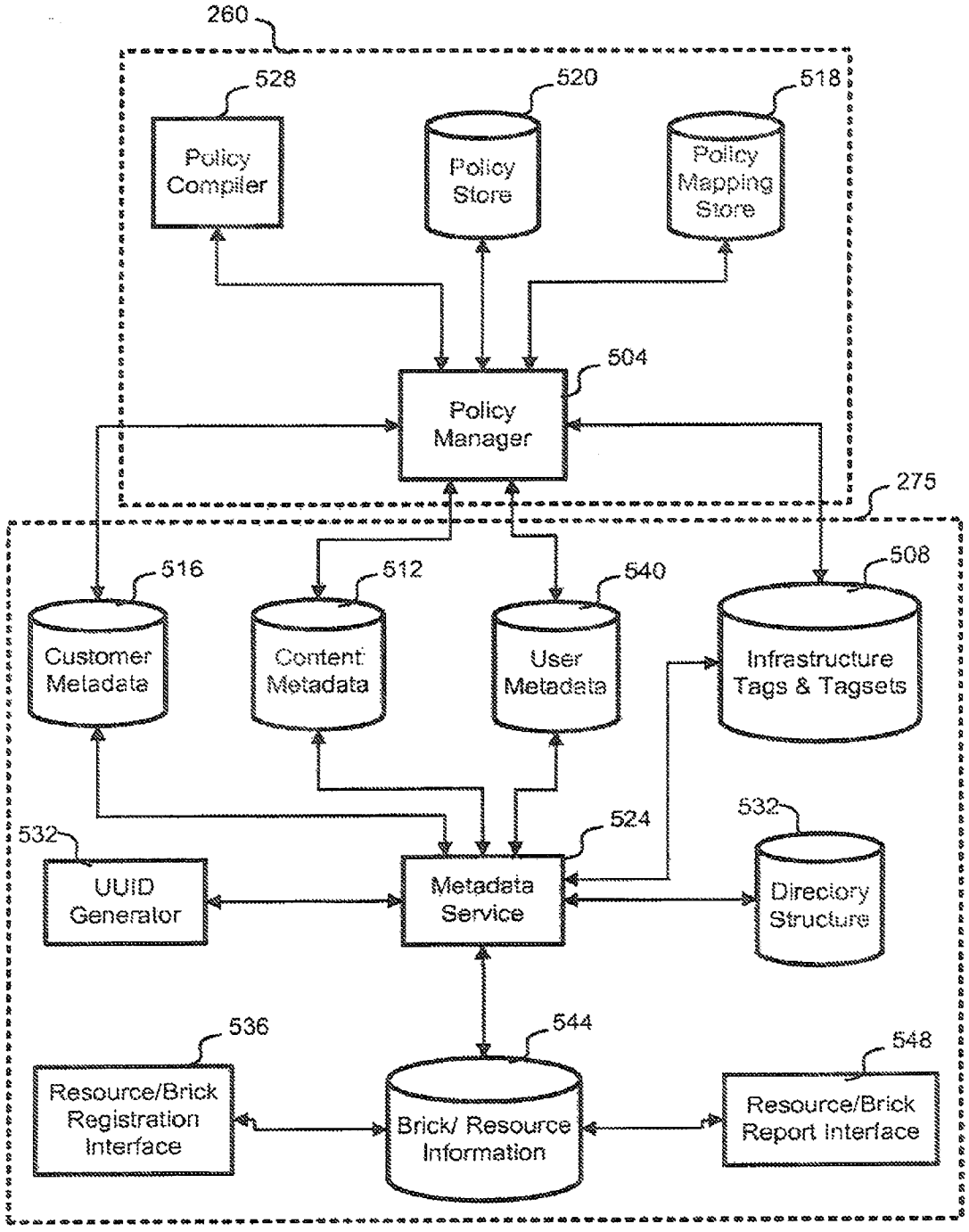

260
528
Policy Compiler
520
Policy Store
518
Policy Mapping Store
504
Policy Manager
275
516
Customer Metadata
512
Content Metadata
540
User Metadata
508
Infrastructure Tags & Tagsets
532
UUID Generator
524
Metadata Service
532
Directory Structure
536
Resource/Brick Registration Interface
544
Brick/ Resource Information
548
Resource/Brick Report Interface

FIG. 5

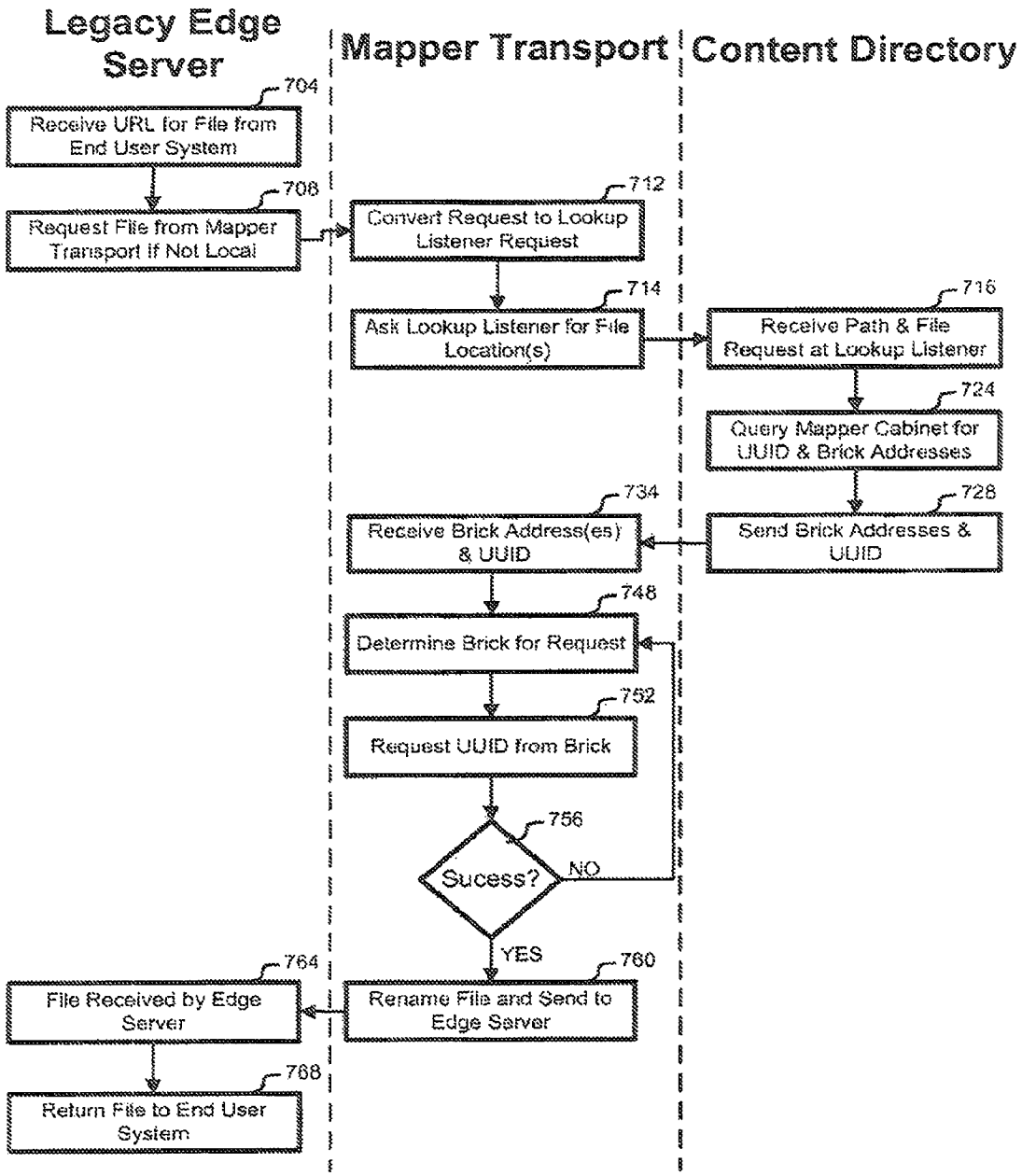

Legacy Edge Server
Mapper Transport
Content Directory
704
Receive URL for File from End User System
708
Request File from Mapper Transport If Not Local
712
Convert Request to Lookup Listener Request
714
Ask Lookup Listener for File Location(s)
716
Receive Path & File Request at Lookup Listener
724
Query Mapper Cabinet for UUID & Brick Addresses
728
Send Brick Addresses & UUID
734
Receive Brick Address(es) & UUID
748
Determine Brick for Request
752
Request UUID from Brick
756
Sucess?
NO
YES
760
Rename File and Send to Edge Server
764
File Received by Edge Server
768
Return File to End User System

FIG. 7

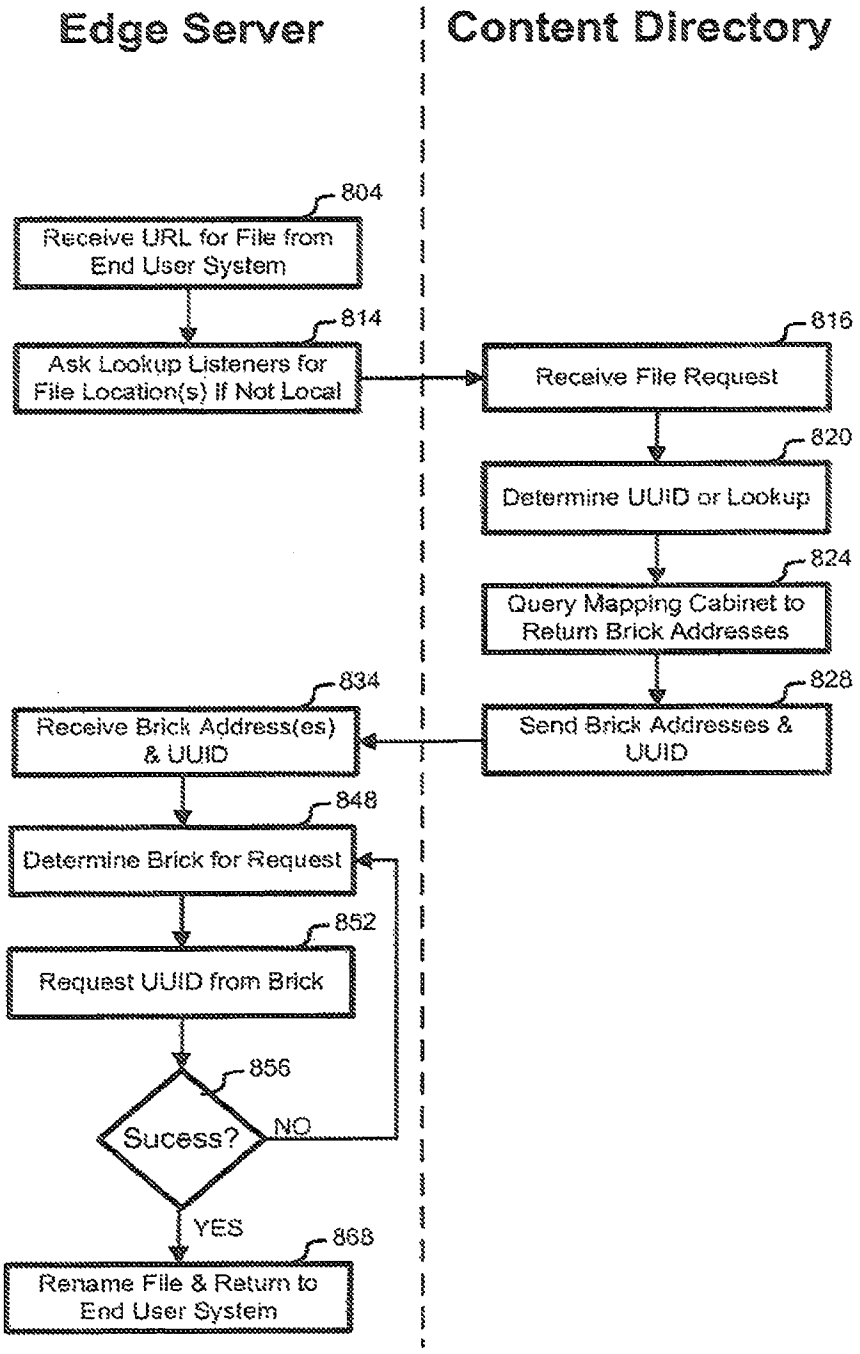

Edge Server
Content Directory
804
Receive URL for File from End User System
814
Ask Lookup Listeners for File Location(s) If Not Local
816
Receive File Request
820
Determine UUID or Lookup
824
Query Mapping Cabinet to Return Brick Addresses
828
Send Brick Addresses & UUID
834
Receive Brick Address(es) & UUID
848
Determine Brick for Request
852
Request UUID from Brick
856
Sucess?
NO
YES
868
Rename File & Return to End User System

FIG. 8

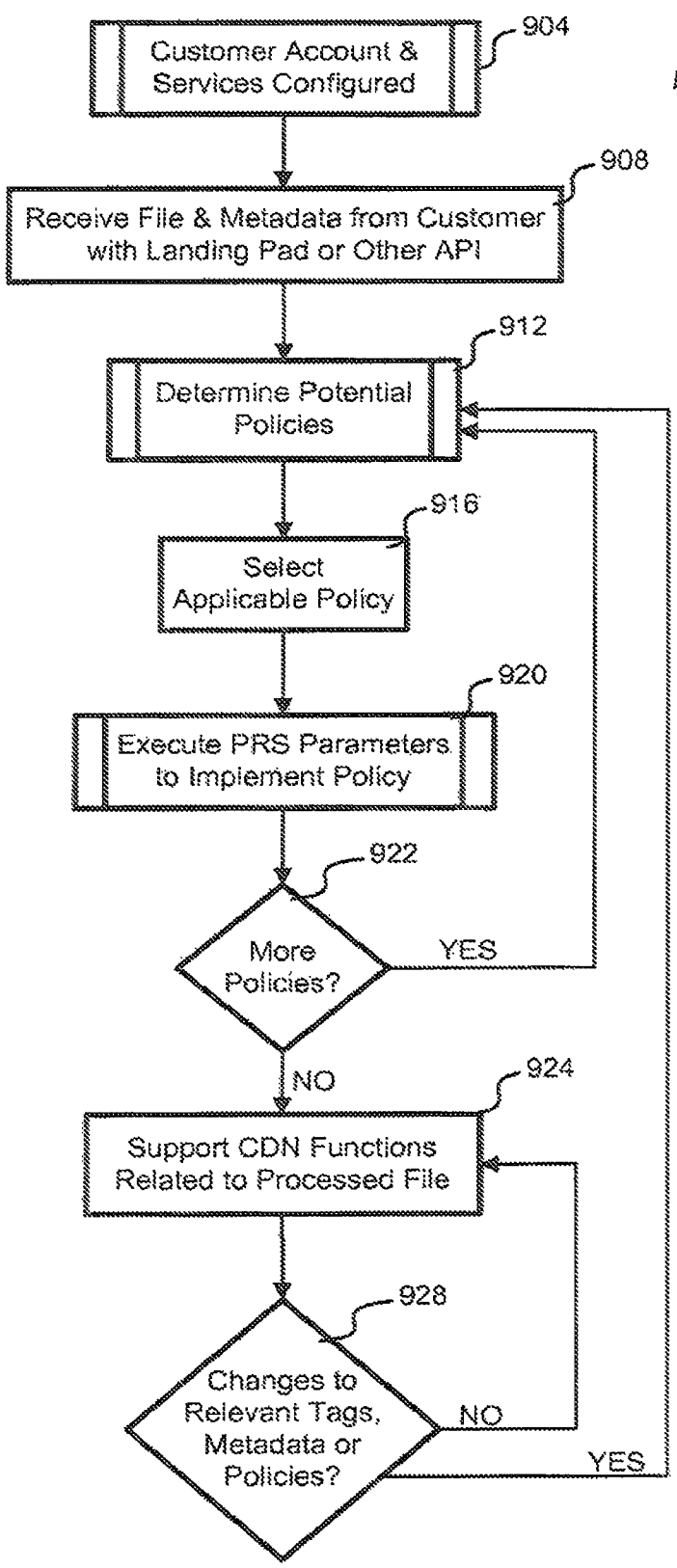

900
904
Customer Account & Services Configured
908
Receive File & Metadata from Customer with Landing Pad or Other API
912
Determine Potential Policies
916
Select Applicable Policy
920
Execute PRS Parameters to Implement Policy
922
More Policies?
YES
NO
924
Support CDN Functions Related to Processed File
928
Changes to Relevant Tags, Metadata or Policies?
NO
YES

FIG. 9

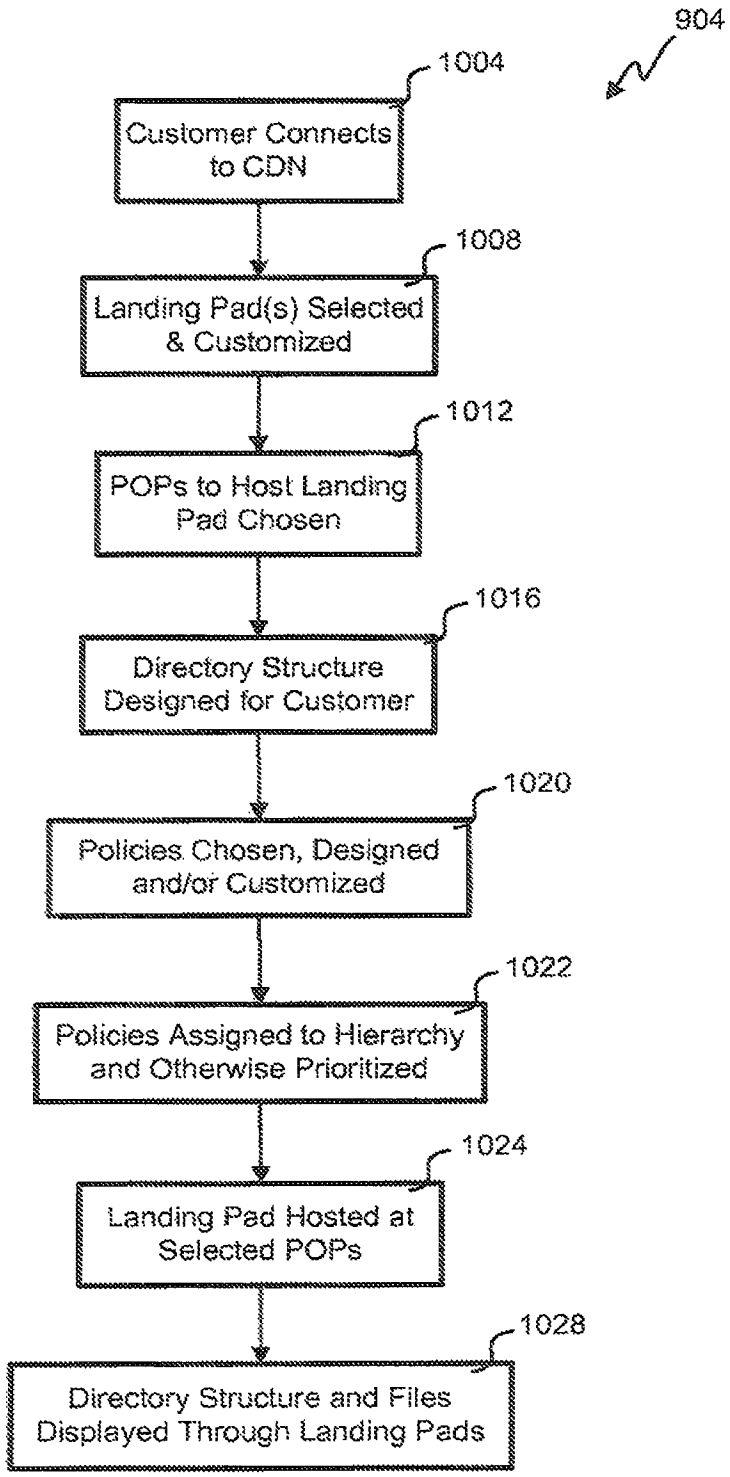

904
1004
Customer Connects to CDN
1008
Landing Pad(s) Selected & Customized
1012
POPs to Host Landing Pad Chosen
1016
Directory Structure Designed for Customer
1020
Policies Chosen, Designed and/or Customized
1022
Policies Assigned to Hierarchy and Otherwise Prioritized
1024
Landing Pad Hosted at Selected POPs
1028
Directory Structure and Files Displayed Through Landing Pads

FIG. 10

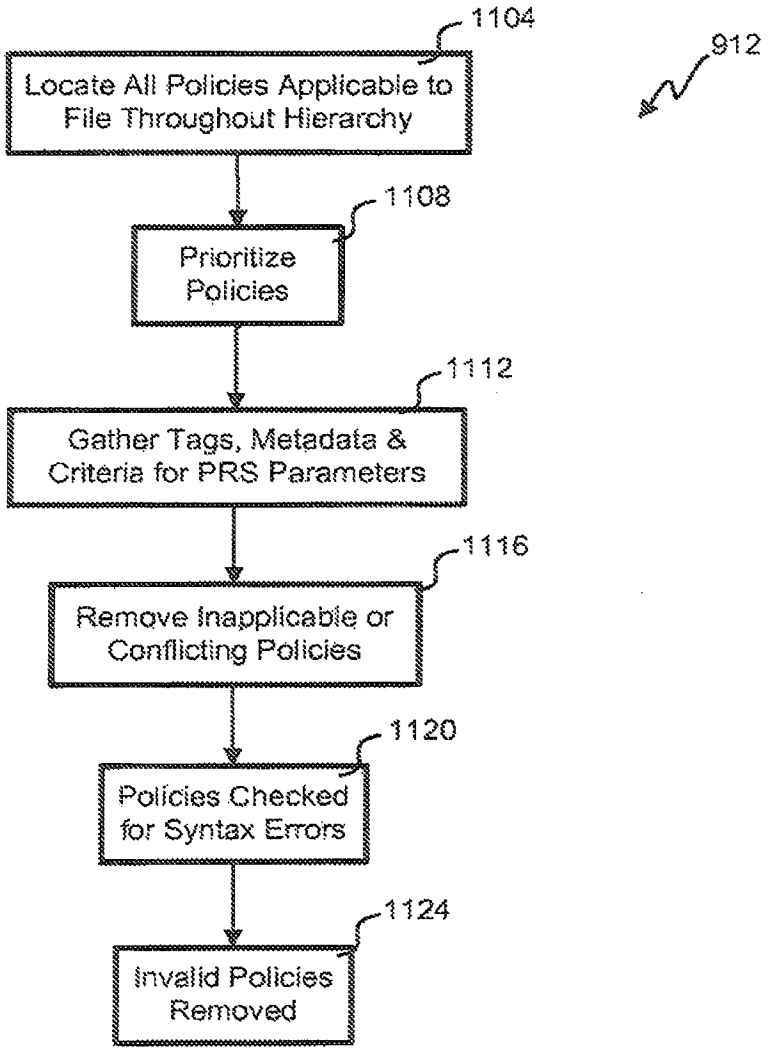

1104
912
Locate All Policies Applicable to File Throughout Hierarchy
1108
Prioritize Policies
1112
Gather Tags, Metadata & Criteria for PRS Parameters
1116
Remove Inapplicable or Conflicting Policies
1120
Policies Checked for Syntax Errors
1124
Invalid Policies Removed

FIG. 11

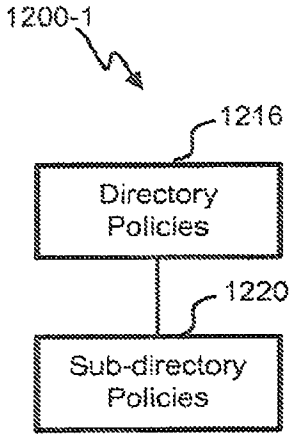


FIG. 12A

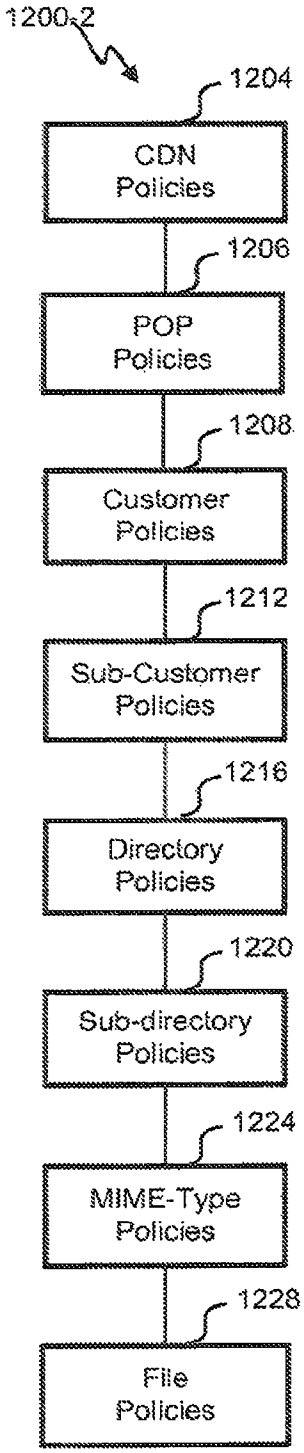


FIG. 12B

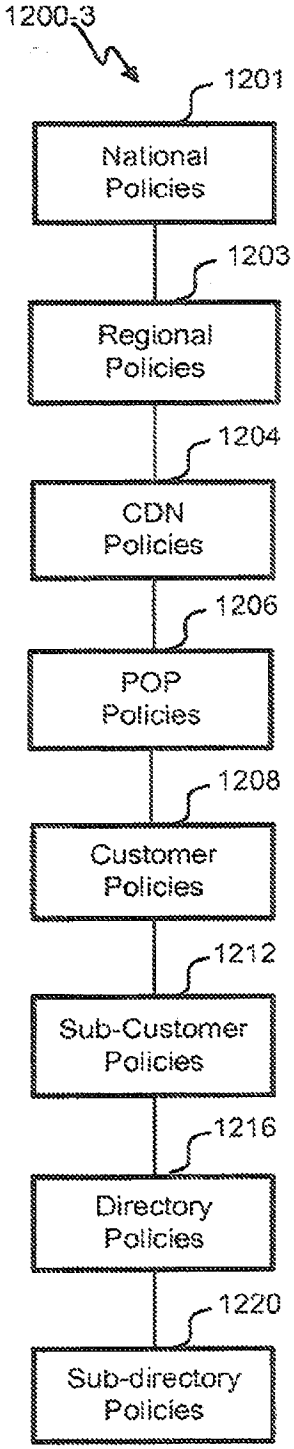


FIG. 12C

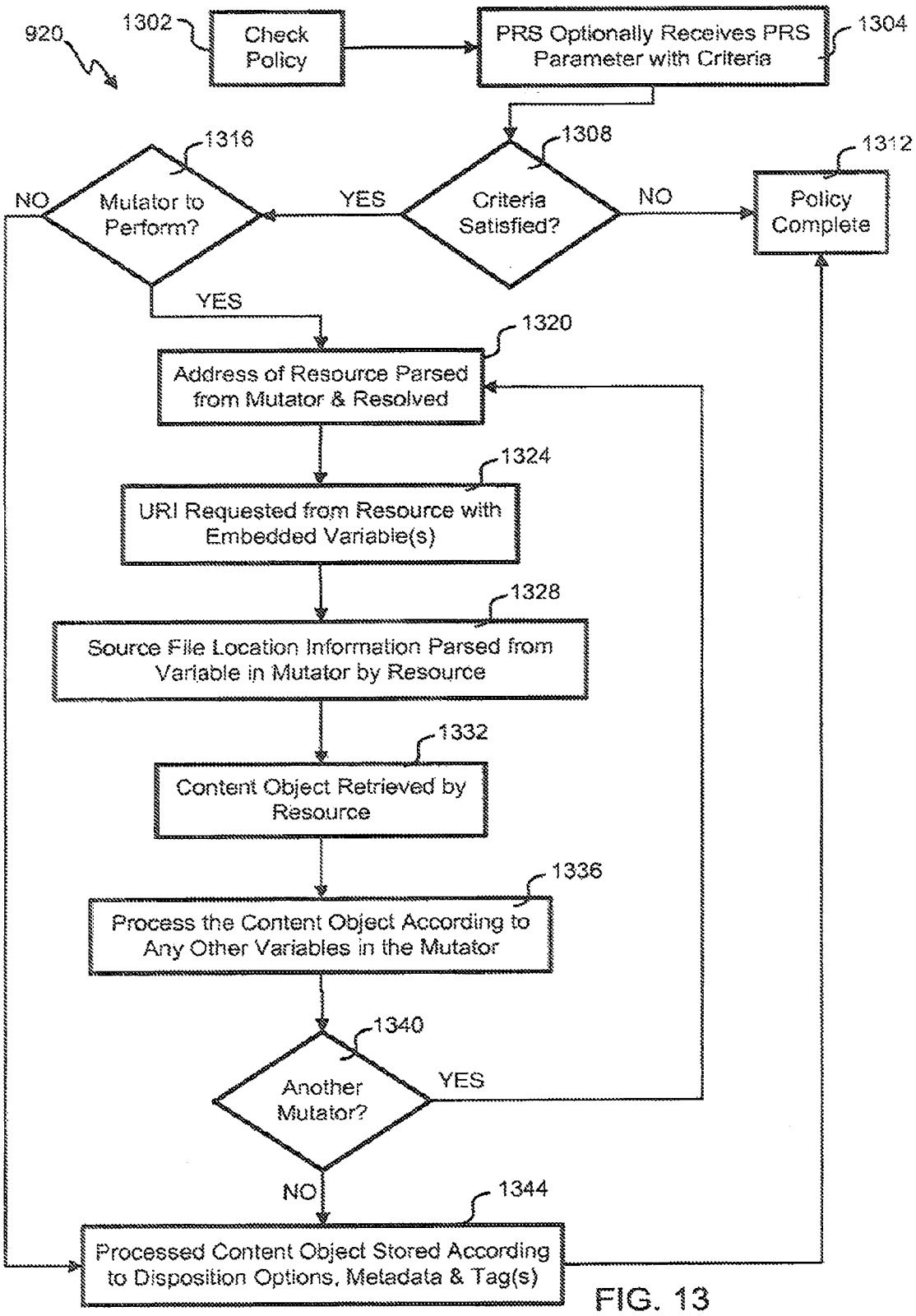

920
1302
Check Policy
PRS Optionally Receives PRS Parameter with Criteria
1304
1316
1308
1312
NO
Mutator to Perform?
YES
Criteria Satisfied?
NO
Policy Complete
YES
1320
Address of Resource Parsed from Mutator & Resolved
1324
URI Requested from Resource with Embedded Variable(s)
1328
Source File Location Information Parsed from Variable in Mutator by Resource
1332
Content Object Retrieved by Resource
1336
Process the Content Object According to Any Other Variables in the Mutator
1340
Another Mutator?
YES
NO
1344
Processed Content Object Stored According to Disposition Options, Metadata & Tag(s)

FIG. 13

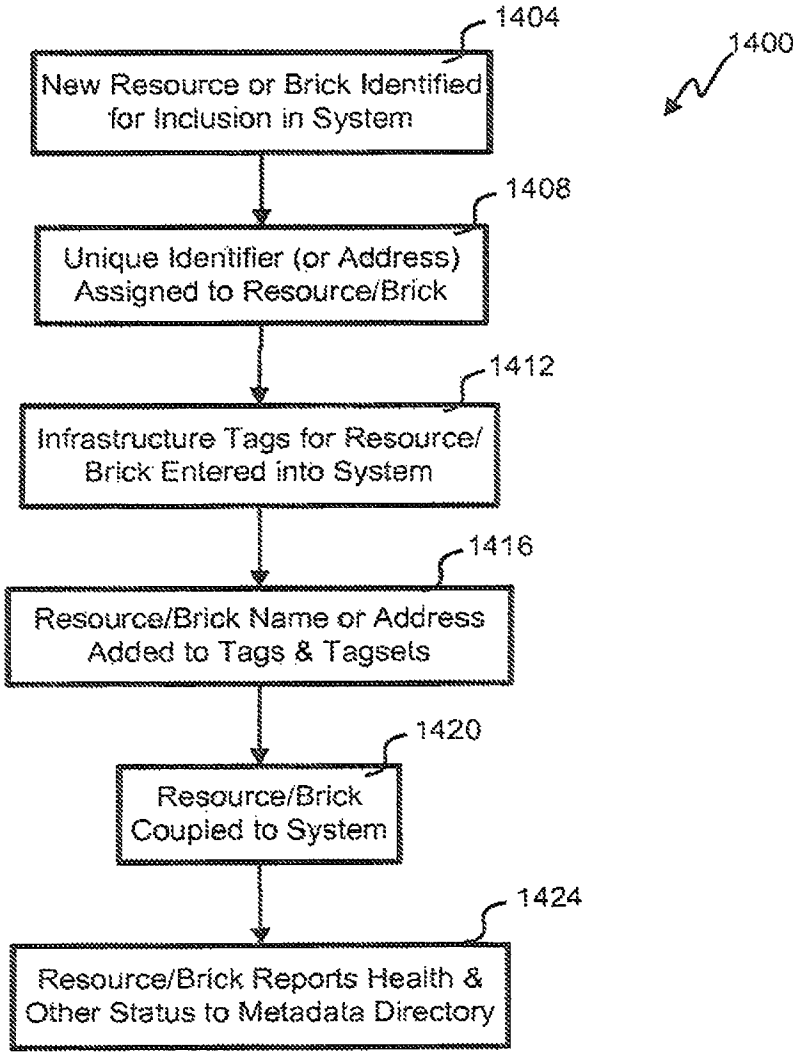

1404
New Resource or Brick Identified for Inclusion in System
1400
1408
Unique Identifier (or Address) Assigned to Resource/Brick
1412
Infrastructure Tags for Resource/ Brick Entered into System
1416
Resource/Brick Name or Address Added to Tags & Tagsets
1420
Resource/Brick Coupled to System
1424
Resource/Brick Reports Health & Other Status to Metadata Directory

FIG. 14

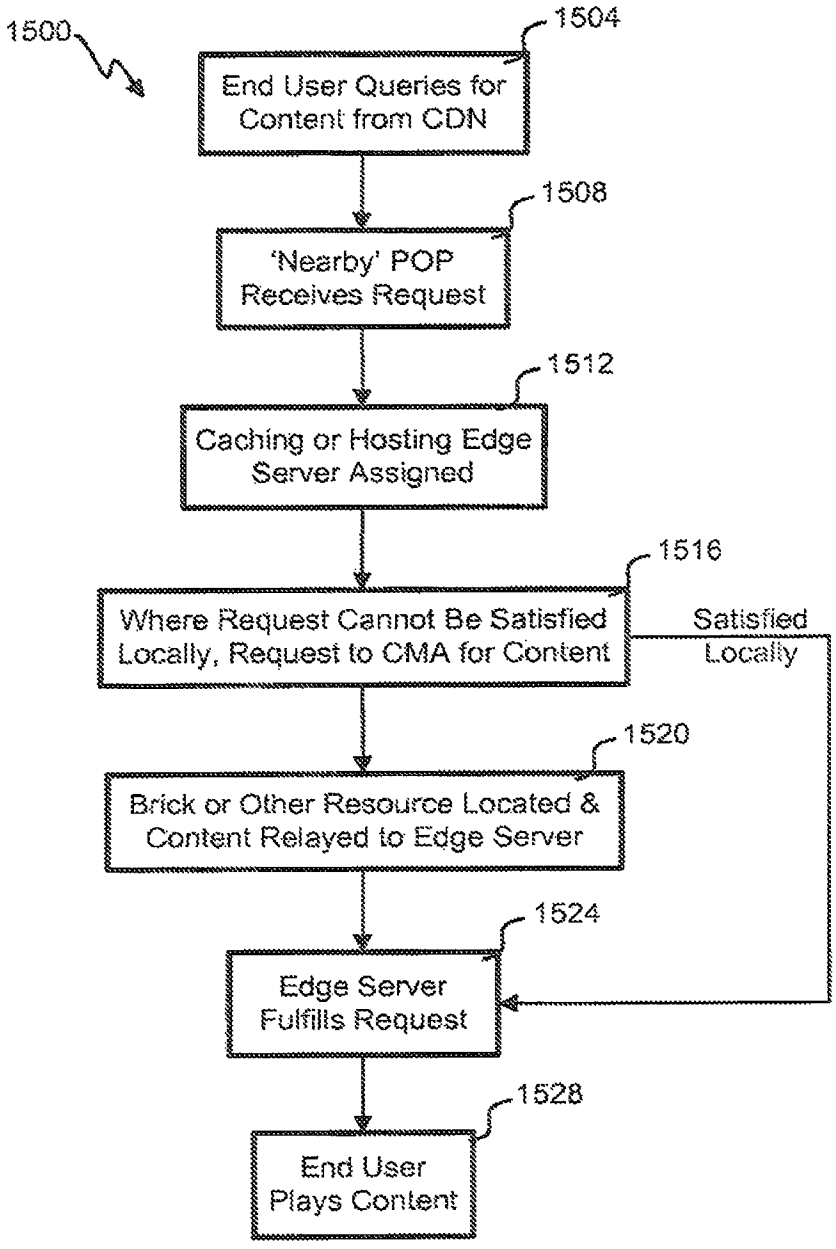

1500
1504
End User Queries for Content from CDN
1508
'Nearby' POP Receives Request
1512
Caching or Hosting Edge Server Assigned
1516
Where Request Cannot Be Satisfied Locally, Request to CMA for Content
Satisfied Locally
1520
Brick or Other Resource Located & Content Relayed to Edge Server
1524
Edge Server Fulfills Request
1528
End User Plays Content

FIG. 15

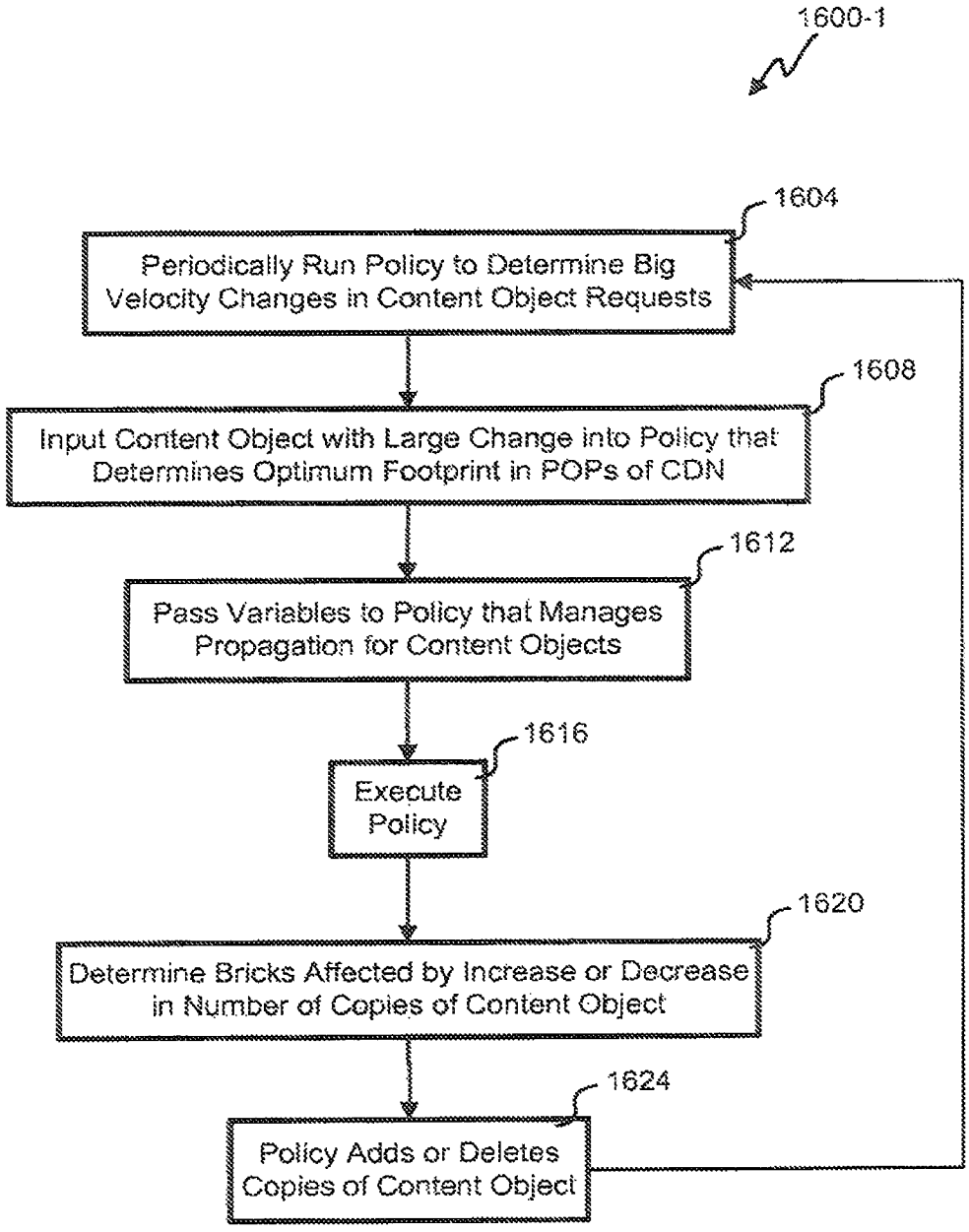

1600-1
1604
Periodically Run Policy to Determine Big Velocity Changes in Content Object Requests
1608
Input Content Object with Large Change into Policy that Determines Optimum Footprint in POPs of CDN
1612
Pass Variables to Policy that Manages Propagation for Content Objects
1616
Execute Policy
1620
Determine Bricks Affected by Increase or Decrease in Number of Copies of Content Object
1624
Policy Adds or Deletes Copies of Content Object

FIG. 16A

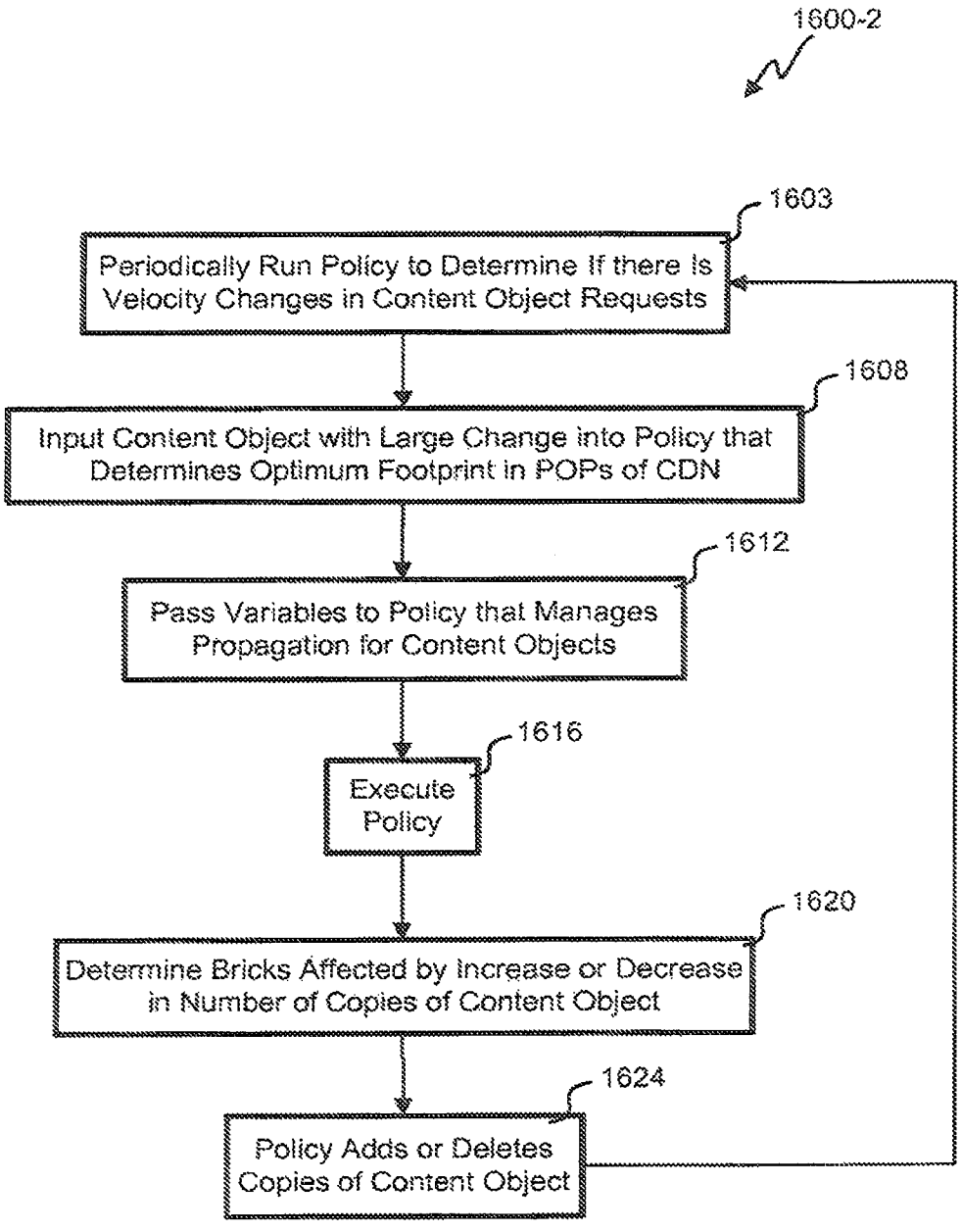

1600-2
1603
Periodically Run Policy to Determine If there Is Velocity Changes in Content Object Requests
1608
Input Content Object with Large Change into Policy that Determines Optimum Footprint in POPs of CDN
1612
Pass Variables to Policy that Manages Propagation for Content Objects
1616
Execute Policy
1620
Determine Bricks Affected by Increase or Decrease in Number of Copies of Content Object
1624
Policy Adds or Deletes Copies of Content Object

FIG. 16B

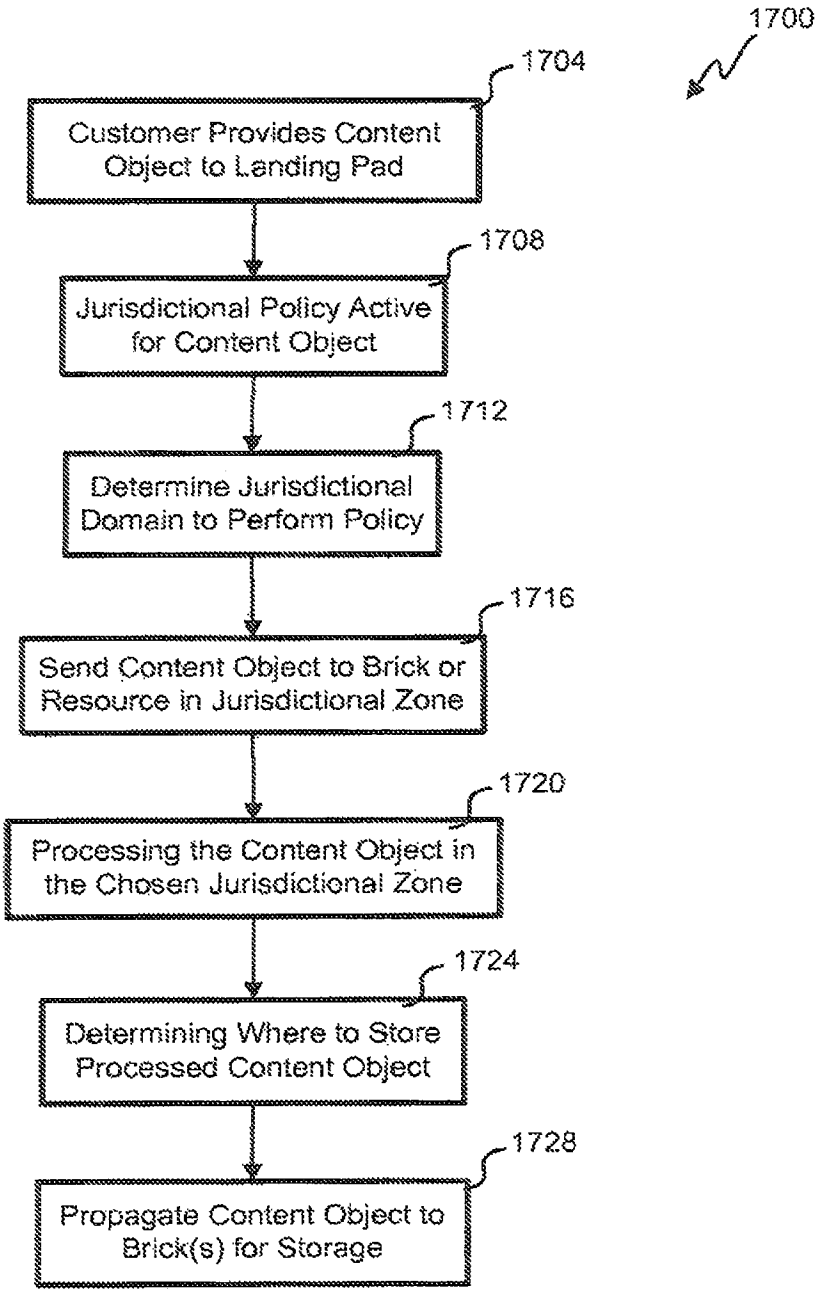

1700
1704
Customer Provides Content Object to Landing Pad
1708
Jurisdictional Policy Active for Content Object
1712
Determine Jurisdictional Domain to Perform Policy
1716
Send Content Object to Brick or Resource in Jurisdictional Zone
1720
Processing the Content Object in the Chosen Jurisdictional Zone
1724
Determining Where to Store Processed Content Object
1728
Propagate Content Object to Brick(s) for Storage

FIG. 17

POLICY BASED PROCESSING OF CONTENT OBJECTS IN A CONTENT DELIVERY NETWORK USING MUTATORS

BACKGROUND

This disclosure relates in general to content delivery networks (CDNs) and, but not by way of limitation, to managing assets associated with the CDN.

Content providers (i.e., customers) offload their content delivery to CDNs for any number of reasons. CDNs specialize in a range of content delivery and hosting options optimized for performance so that end users get a desired quality of service (QoS). Different CDNs have different topologies relying on more or less points of presence (POPs) distributed geographically across the Internet. CDNs with more POPs tend to have less resources in each POP, while those with less POPs tend to have more resources in each one. The topology and asset allocation in a particular CDN is inflexible.

Customers are using CDNs and cloud services in more creative ways. Applications, storage and other services are being provided remotely. CDNs have not provided the flexibility to adapt to all the needs of customers yet have an excellent topology of distributed POPs with fast interconnectivity between those POPs. Currently, limited interfaces to the CDN with little or no customization results in lost opportunity.

SUMMARY

In one embodiment, the present disclosure provides a method and system for flexibly processing content objects. The processing is performed with a content delivery network (CDN) having a number of geographically distributed points of presence (POPs). Content objects are ingested through landing pads and stored or otherwise processed in a flexible way where storage bricks and other resources are chosen flexibly by characterization tags. Policies are used to describe which content objects are processed by which categories of resources. A group of resources characterized by the tag are chosen when the processing is performed. When retrieving content, the content object can be stored on any storage brick found through the tag analysis process. A query is translated to addresses for the chosen storage brick(s).

In another embodiment, the present disclosure provides a method for processing content objects with a plurality of resources associated with a content delivery network (CDN) having a plurality of geographically distributed points of presence (POPs). The plurality of resources are enrolled to be accessible from the CDN. Each of the plurality of resources is categorized using a plurality of tags that categorize the plurality of resources, and the plurality of resources includes a resource. Selection of a policy from a plurality of policies is received, and the plurality of policies define processes to perform on content objects stored at the CDN. The selected policy includes an applicability criteria and a call to the resource. Metadata is received at the CDN related to the content object, a requester of the content object and/or a provider of the content object. The content object is received for storage at the CDN. It is determined, through analysis of the metadata and/or the applicability criteria, that the policy is applicable and that other policies are not applicable to the received content object. The resource is called according to the call in the policy to cause the resource to perform specified processing on the received content object.

In another embodiment, the present disclosure provides a content delivery network (CDN) having a plurality of geographically distributed points of presence (POPs) for processing content objects with a plurality of resources. The CDN includes a landing pad to receive a content object from a client, one or more databases comprising a list of the plurality of resources, each of the plurality of resources being associated with one or more tags, and each tag indicating a characteristic of the associated resource. The CDN also includes a policy reconciliation service (PRS) for maintaining and processing policies, the PRS being coupled to the one or more databases. The PRS includes a policy store comprising a plurality of policies, each of the plurality of policies defining specific processing of content objects, the plurality of policies including a first policy and a second policy. Each of the first policy and the second policy includes an applicability parameter indicating criteria that a content object must satisfy in order for the content object to be processed in accordance with the respective first or second policy, the criteria indicated in the first policy's applicability parameter being different from the criteria indicated in the second policy's applicability parameter. The first policy comprises a disposition parameter indicating criteria that a resource must satisfy in order for the resource to effect the first policy, and the first policy comprises one or more mutators, each mutator comprising a template for inclusion of an address of a resource and/or a location of a received content object. The CDN also includes a policy manager configured to determine that the first policy is applicable to the received content object and that the second policy is not applicable to the received content object, the determination being based on the first policy's criteria and metadata related to the received content object, a requester of the received content object and/or a provider of the received content object at the CDN. The policy manager is further configured to identify a resource of the plurality of resources for effecting the first policy based on the disposition parameter and a tag associated with the first policy.

In another embodiment, the present disclosure provides a content delivery network (CDN) having a plurality of geographically distributed points of presence (POPs) for processing content objects with a plurality of resources. The CDN includes two or more hardware servers programmed for enrolling the plurality of resources to be accessible from the CDN; categorizing each of the plurality of resources using a plurality of tags that categorize the plurality of resources, where the plurality of resources includes a resource; receiving selection of a policy from a plurality of policies, where the plurality of policies define processes to perform on content objects stored at the CDN, wherein the selected policy includes an applicability criteria and a call to the resource; receiving, at the CDN, metadata related to a content object, a requester of the content object and/or a provider of the content object; receiving the content object for storage at the CDN; determining that the policy is applicable and that other policies are not applicable to the received content object through analysis of the metadata and/or the applicability criteria; and calling the resource according to the call in the policy to cause the resource to perform specified processing on the received content object.

Further areas of applicability of the present disclosure will become apparent from the detailed description provided hereinafter. It should be understood that the detailed description and specific examples, while indicating various embodiments, are intended for purposes of illustration only and are not intended to necessarily limit the scope of the disclosure.

BRIEF DESCRIPTION OF THE DRAWINGS

The present disclosure is described in conjunction with the appended figures:

FIG. 1 depicts a block diagram of an embodiment of a content distribution system;

FIGS. 2A and 2B depict block diagrams of embodiments of a point of presence (POP);

FIGS. 3A and 3B depict block diagrams of embodiments of a content management architecture;

FIGS. 4A and 4B depict block diagrams of embodiment of a content brick or resource;

FIG. 5 depicts a block diagram of an embodiment of the policy reconciliation service interacting with a metadata directory;

FIG. 6 depicts a diagram of an embodiment of a directory structure;

FIGS. 7 and 8 illustrate swim diagrams of embodiments of a process for using the content management architecture to retrieve a content object;

FIG. 9 illustrates a flowchart of an embodiment of a process for applying policies to a content object;

FIG. 10 illustrates a flowchart of an embodiment of a process for configuring a customer account;

FIG. 11 illustrates a flowchart of an embodiment of a process for disambiguation of policies;

FIGS. 12A, 12B and 12C depict block diagrams of embodiments of policy prioritization hierarchies;

FIG. 13 illustrates a flowchart of an embodiment of a process for performing a policy;

FIG. 14 illustrates a flowchart of an embodiment of a process for enrolling a resource or brick into the content distribution system;

FIG. 15 illustrates a flowchart of an embodiment of a process for delivering a content object using the content management architecture;

FIGS. 16A and 16B illustrate flowcharts of embodiments of a process for elastically managing propagation of content objects; and

FIG. 17 illustrates a flowchart of an embodiment of a process for using policies to change the jurisdiction used to process a content object.

In the appended figures, similar components and/or features may have the same reference label. Further, various components of the same type may be distinguished by following the reference label by a dash and a second label that distinguishes among the similar components. If only the first reference label is used in the specification, the description is applicable to any one of the similar components having the same first reference label irrespective of the second reference label.

DETAILED DESCRIPTION

The ensuing description provides preferred exemplary embodiment(s) only, and is not intended to limit the scope, applicability or configuration of the disclosure. Rather, the ensuing description of the preferred exemplary embodiment(s) will provide those skilled in the art with an enabling description for implementing a preferred exemplary embodiment. It is understood that various changes may be made in the function and arrangement of elements without departing from the spirit and scope as set forth in the appended claims.

Referring first to FIG. 1, a block diagram of an embodiment of a content distribution system **100** is shown. The content originator **106** offloads delivery of the content objects to a content delivery network (CDN) **110** in this embodiment. The content originator **106** produces and/or distributes content objects and includes a content provider **108**, a content site **116**, and an origin server **112**. The CDN **110** can cache, redistribute and/or host content in various embodiments for third parties such as the content originator **106** to offload delivery and typically provide better quality of service (QoS).

In this embodiment, the content distribution system **100** locates the content objects (or portions thereof) and distributes the content objects to end user systems **102**. A content object is any content file or content stream and could include, for example, video, pictures, data, audio, software, and/or text. The content object could be live, delayed or stored. Throughout the specification, references may be made to a content object, content, content stream and/or content file, but it is to be understood that those terms could be used interchangeably wherever they may appear.

Many content providers **108** use a CDN **110** to deliver the content objects over the Internet **104** to end users **128**. The CDN **110** includes a number of points of presence (POPs) **120**, which are geographically distributed through the content distribution system **100** to deliver content. Various embodiments may have any number of POPs **120** within the CDN **110** that are generally distributed in various locations around the Internet **104** to be proximate to end user systems **102**. Multiple POPs **120** use the same IP address such that an Anycast routing scheme is used to find a POP **120** likely to be close, in network terms, to the end user for each request. In addition to the Internet **104**, a wide area network (WAN) **114** or other backbone may couple the POPs **120** with each other and also couple the POPs **120** with other parts of the CDN **110**.

When an end user **128** requests a web page through its respective end user system **102**, the request for the web page is passed either directly or indirectly via the Internet **104** to the content originator **106**. The content originator **106** is the source or re-distributor of content objects. The content site **116** is a web site accessible by the end user system **102**. In one embodiment, the content site **116** could be a web site where the content is viewable with a web browser. In other embodiments, the content site **116** could be accessible with application software other than a web browser. In this embodiment, the content provider **108** directs content requests to a CDN **110** after they are made or formulates the delivery path to the CDN **110** by embedding the delivery path into the URLs for a web page. In any event, the request for content is handed over to the CDN **110** by using an Anycast IP address corresponding to one, two or more POPs **120**.

Once the request for a content object is passed to the CDN **110**, the request is associated with a particular POP **120** within the CDN **110** using the Anycast routing scheme. The particular POP **120** may retrieve content object from the content provider **108** if not already within the CDN **110**. Alternatively, the content provider **108** may directly provide the content object to the CDN **110** and its associated POPs **120** through pre-population or hosting in advance of the first request. The CDN servers include edge servers that actually serve end user requests. The origin server **112** holds a copy of each content object for the content originator **106**. Periodically, the content of the origin server **112** may be reconciled with the CDN **110** through a cache, hosting and/or pre-population algorithm. Some content providers **108** could use an origin server within the CDN **110** to host the content and avoid the need to maintain an accessible copy of the content object at the origin server **112** of the content originator **106**.

Once the content object is retrieved from the origin server **112** by the CDN **110**, the content object is stored in a manner accessible to the CDN to allow processing by that POP **120** to service the end user systems **102**. For example, the content object could be stored on a brick **130**. Streamed content objects can have real time or near real time information or can be previously stored. The end user system **102** receives the

content object and processes it for use by the end user **128**. The end user system **102** could be a personal computer, media player, handheld computer, Internet appliance, phone, IPTV set top, web server, processing system, streaming radio or any other device that receives and/or plays content objects. In some embodiments, a number of the end user systems **102** could be networked together. Although this embodiment only shows a single content originator **106** and a single CDN **110**, it is to be understood that there could be many of each in various embodiments.

Storage accessible to the CDN **110** includes bricks **130** in this embodiment. A brick **130** is any storage medium inside or outside the CDN **110** that is part of a content management architecture (CMA). The storage medium includes a layer of software to accommodate commands for the brick. Any storage array, network attached storage, drive, flash media, or other non-volatile memory could act as a brick **130** with the proper layer of software. In this embodiment, one of the end user systems **102-1** has a brick **130-1** coupled to it. The CDN **110** could store content on any brick **130** to implement a policy, regardless of whether the brick is internal or external to the CDN **110**.

Other resources **134** are available to the CDN **110** to process content. Resources **134** can be internal or external to the CDN **110**. A brick **130** is just a resource, but it is broken out separately since the processing it performs is largely limited to storage. Generally, a resource **134** is any hardware or software function that can store or process a content object. Examples include, transcoders, cryptographic processors, compression engines, content processors (e.g., image processors, video processors or audio processors), thumbnail generators, media syndication services, video/audio ad insertion engines, video processing services, metadata insertion engines, or anything that can process content objects and can be interfaced with an API from the CDN **110**. In this example, there is a first resource **134-1** available to the CDN **110** over the Internet **104** and a second resource **134-2** within the CDN **110**, but it is to be understood there could be many more resources available to the CDN **110**.

With reference to FIG. **2**A, a block diagram of an embodiment of a POP **120-1** is shown. There are both legacy edge servers **235** that don't natively support the CMA and edge servers **230** that do in this embodiment. Legacy edge servers **235** use a mapper transport **245** that supports the CMA to gather any content requested from CDN **110** and present the content like an origin server. The mapper transport **245** makes the calls necessary to locate the content and pass it to the legacy edge server **235**. Requests are made to bricks **130** by the mapper transport **245** that acts as a reverse proxy to return the requested content. Typically, the software on the legacy edge server **235** does not require any rewriting to allow integration with the CMA because of the mapper transport **245**.

The various edge servers **230**, **235** are coupled to each other and the Internet **104** and WAN **114** using switching fabric **240**. Edge servers **230**, **235** number in the thousands in a given POP **120**. The edge servers **230**, **235** could be divided by function and/or customer. Loading algorithms can be used to divide load among the edge servers **230**, **235** in any number of ways. The edge servers **230** perform caching, streaming, hosting, storage, and/or other functions within the POP **120**. An edge server **230**, **235** is typically a rack-mounted computer that could have varying levels of processing power, memory and storage. Software running on the edge server **230**, **235** includes, for example, HTTP proxy caching, media servers, Flash™ servers, Java™ application servers, Silverlight™ servers, etc.

The switching fabric **240** is used for several functions. Incoming requests for content objects are routed to the edge servers **230**, **235** using the switching fabric. This could be done using routing, redirection or domain name service (DNS). Load balancing, round-robin, and/or other techniques could be used by the switching fabric **240** to route requests to edge servers **230**, **235**. Communication within the POP **120** also uses the switching fabric **240**. Edge servers **230**, **235** could have multiple software applications running that communicate with other edge servers **230**, **235**.

There are legacy landing pads **215** and landing pads **210** supporting CMA. The legacy landing pads **215** use a legacy adapter **285** to integrate with the CMA. The legacy adapter **285** includes a portable operating system interface for Unix ("POSIX") adapter to allow backward compatibility to legacy landing pads **215**. Many applications are designed to directly interface with the AMA without requiring the POSIX functionality of the legacy adapter **285**. A universal namespace and directory space is provided by the legacy adapter **285** for the CMA to abstract the legacy storage interface from the native storage. There can be multiple landing pads **210**, **215** in multiple POPs **120** for a given customer to provide an ingest point for content objects.

A content directory **205** is provided for the CMA to allow locating, processing and storing content. The content directory **205** includes a metadata directory **275** and a content mapper **220**. The metadata directory **275** manages through selection of resources and bricks that are members of tag and tagset groups, which resources and bricks are selected for particular processing task. The content mapper **220** is just a database storing UUID and corresponding path and filename along with the brick addresses that store the file referenced by a particular UUID. The health of bricks **130** and resources **134**, metadata, tags, and other information is maintained by the metadata directory **275**. All bricks **130** and resources **134** have various tags associate with them. For each tag or tagset, the bricks **130** or resources **134** that have that tag or tagset are known to the metadata directory **275** to allow selection of a brick **130** or resource **134** for a particular processing task performed on a content object.

The content mapper **220** is a distributed database or data structure that translates path and filename to a universal unique identifier (UUID). In this embodiment, the UUID is a 256-bit number that is randomly, pseudorandomly, sequentially, or unpredictably assigned to each content object file stored in the CDN **110**. It is extremely unlikely that two files would have the same UUID and a check could be performed prior to assignment to be sure the UUID generated hasn't already been used within the CDN **110**.

A policy reconciliation service (PRS) **260** maintains and processes policies. Each policy defines processing to perform on one or more content objects. The operation of a policy is affected by criteria based upon metadata and tags/tagsets. Where there are multiple policies applicable to content, the PRS disambiguates the situation based upon a hierarchy or by picking the lowest or highest common denominator for the applicable policies.

Within each POP **120** or elsewhere in the CDN, there are a number of bricks **130** that store content objects and resources **134** that process the content objects. A policy can define the classes of bricks suitable for storage and the processing that a resource **134** would perform on a content object. Parameters are passed to a resource **134** using a mutator that is part of a policy.

With reference to FIG. **2**B, a block diagram of an embodiment of a POP **120-2** is shown. The edge servers **230** and landing pads **210** in this embodiment natively support the

CMA without any translation or interfaces required. Calls are made to the content directory **205** to find the UUID for a content object and the brick names or identifiers that hold the content object. When storing content objects, the content directory **205** uses the tags and metadata to choose one or more bricks **130** that would store a particular content object.

With reference to FIG. **3**A, a block diagram of an embodiment that shows portions of a content management architecture (CMA) using legacy edge servers **235**. This embodiment has only legacy edge hosts **316** and legacy edge caches **304** instantiated on the legacy edge servers **235**. Other embodiments could additionally include edge caches and/or edge servers that natively support the CMA.

When a content object or portion thereof is not found on the legacy edge server **235**, reference to a mapper transport **245** is made. The mapper transport **245** acts as an origin server for all the content in the CMA. The mapper transport **245** interacts with the lookup listeners **328** to get names or addresses of the bricks **130** that hold the content object along with its UUID. The mapper transport **245** then proxies the content object back to the requesting legacy edge cache **304** or legacy edge host **316**. The protocol and handshaking expected by the legacy edge cache **304** or legacy edge host **316** is performed by the mapper transport.

The metadata directory **275** and content mapper **220** collectively form the content directory **205**. The metadata directory **275** translates a path and filename to a UUID when originally storing a content object. To find out a UUID or brick addresses, the path and filename is sent to the content mapper **220**, by multicasting using multiple unicast channels to some or all the lookup listeners **328**. The namespace is divided between the lookup listeners **328** in addition to having multiple alternative lookup listeners **328**. Multiple lookup listeners **328** that receive the request will respond, but the requester only uses the first lookup listener **328** to respond. Where there are multiple lookup listeners distributed around the CDN, a distributed database protocol is used to keep all of them reconciled.

The mapper cabinet **324** stores the UUID and brick names or addresses for each path and filename combination. The lookup listener **328** queries the mapper cabinet **324** with the path and filename, to get the UUID for that path and filename that is returned with all the brick names that hold the content file. The lookup listener **328** with the answer passes the UUID and brick information back to the mapper transport **245**. Where there are multiple bricks **130** with the UUID, the mapper transport **245** chooses one and confirms it is there. Additional, bricks **130** could be queried if unsuccessful. The mapper transport **245** proxies the content object back to the requesting legacy edge cache **304** or legacy edge host **316**, but other embodiments could redirect the request directly to the brick **130**.

With reference to FIG. **3**B, a block diagram of another embodiment that shows portions of a CMA. This embodiment uses edge caches **305** and edge hosts **315** that support the CMA. Where the content object is not found locally, the edge server **230** will request the path and file from the content mapper **220**. Through multicast to the lookup listeners **328**, one with the answer returns it after a query to its respective mapper cabinet **324**. The edge server **230** can make an educated guess on what lookup listeners **328** are likely to respond first instead of querying all of them within the CDN each time. The guess could be based upon which returned answers quickly in the past or based upon an estimate of the closest ones in a network sense.

The UUID and brick names or addresses are returned by the content mapper **220** to the edge server **230**. The edge cache **305** or edge host **315** can directly request the content object from a brick address by providing it the UUID. Where there are multiple bricks with the content object, they could be queried according to different schemes, for example, querying in parallel or sequentially. The bricks **130** may be inside or outside the CDN **110**. Where a name of a brick is returned instead of an address, a domain name lookup service could be used to find the address.

Referring next to FIG. **4**A, a block diagram of an embodiment of a content brick **130** is shown. A brick **130** is connected to the switching fabric **240** in some way to be managed the CMA. The brick daemon **404** is a software layer that is between the switching fabric **240** and the native file interface **408** to translate communication with the CMA to allow storage on the native storage **412**. Since there are many native file interfaces and host platforms, the brick daemon **404** is customized for the host platform. This embodiment of the brick daemon **404** only does translation, but other embodiments could perform authentication and/or encryption. Files are stored on the native storage **412** with the UUID as the file name.

Referring next to FIG. **4**B, a block diagram of an embodiment of a resource **134** is shown. The resource **134** could be any hardware or software that processes a content object. A resource API **405** receives mutators and other commands. The resource API **405** interfaces with a native resource interface **409** to command a native resource **413** to perform processing on a content object. In some cases, the resource **134** has a native API the is suitable for integration with the CMA without the need for a resource API layer.

With reference to FIG. **5**, a block diagram of an embodiment of the PRS **260** interacting with a metadata directory **275** is shown. The PRS **260** includes a policy manager **504** that controls a policy compiler, a policy store **520**, and a policy mapping store **518**. The policy compiler **528** performs disambiguation to resolve conflicts when multiple policies apply to the same content object. Conflicts can be resolved by a hierarchical scheme where policies higher in the hierarchy take precedence. In another embodiment, the policy compiler chooses the most or least stringent of the conflicting policies. For example, a policy that requires all JPEG files be deleted after two weeks and another policy that requires all files to be deleted when not requested for a day could be resolved either most stringently to delete the JPEG file after not being used for a day or least stringently to be deleted after two weeks. Additionally, any syntax errors in the policies are found and identified by the policy compiler **528**.

The policy store **520** holds all the policies in the CMA. The policies are applicable to many customers and each have various levels of alterations for a particular customer. There are policies for ingest, replication, hosting, transcoding, encryption, compression, thumbnailing, various workflows, aging content in/out of system, and other processing for a content object. Each policy is a function with defined with PRS parameters that include criteria, variables, storage disposition and optional mutators. Table I below shows examples of some policies with the PRS parameters that implement the policy and the variables used. For example, a transcode policy retrieves a source URL and places it in an intake subdirectory for the transcoder. The transcoder performs any number of different transcodes on the source files as specified and stores the resulting files as the specified transcode URLs.

TABLE I

Example Ingest and Hosting Policies

Policy	PRS Parameters	Variable(s)
Ingest	Ingest API	Origin URL, Content Tags,
	Information	Transcode Options,
		Storage Options, Purge Date
	Transcoder	Transcode Options, URL,
	Format(s)	Content Tags
	Store File(s)	File Name, Storage Options
	Automatic Purging	Purge Date
Replication	File Copy	Number of Copies, Content
		Tags, Infrastructure Tags
Transcode	Retrieve Source	Source URL
	n Transcodes	Transcode Options, Different
		Transcodes, Source URL, Content
		Tags
	Store Results	Transcode URLs
Host	Hosting API	Origin URL, Content Metadata,
	Information	Customer Metadata,
		Storage Options, Purge Date
	Store File(s)	Stored URL, Storage Options
	File Aging	Purge Date

Policies are preformulated or custom designed by content providers or content receivers. The policy could be in any format (e.g., XML, text, etc.) and could be command line instructions, run-time executed source code or compiled into object code. Policies can age into or out of system. A PRS parameter acts as an instruction in a CDN-specific programming language. A policy can be assigned to an end user who receives content, a customer who provides content, a content object, a class of content objects, a directory, and/or any other tag or metadata demarcation.

Criteria for a policy define its applicability to the content objects in the CMA. Criteria allow size-based processing, MIME type workflows, or any metadata or tag qualifier before performing the policy. For example, a compression policy could be applied to a particular MIME type stored in a particular POP that has not been requested for some period of time.

Each policy has PRS parameter that defines a disposition for the content object to be performed after any processing. The disposition can say what type bricks **130** or resources **134** to use. The number of copies of the content object to have and what geographic spread to place on those copies can also be defined. A deletion date can be defined in the PRS parameter.

A mutator indicates a resource **134** that will process the content object. The API to the resource typically includes the source path and filename for content file and any number of variable that affect processing by the resource **134**. The mutators are in the form of a URL in this embodiment, but other embodiments could use any format. The mutator URL identifies the address of the resource **134**, a source content object location that is being operated upon and a number of variables. The mutator URL can perform conditional actions based upon prior mutators and/or variables.

The functionality of a policy is demonstrated with an example thumbnailing policy that uses a thumbnailing resource to create thumbnail images for an image content object. In this example, the policy would store any files which end in the file name extension is 'jpg' in three different locations. One of the locations is in the European Union and two are stored in the United States. Once all copies have been made, a call to the thumbnailer resource **134** is made, which generates a small thumbnail image of the source JPEG file that is stored in a predetermined location. The thumbnailer resource **134** uses the pathname of the source JPEG file as well as its size in bytes passed as variables in the mutator URL. The thumbnailer resource **134** is an HTTP-based API which is called with this URI: http://www.imagetransform.org/thumbnailer?path=<full path to source image>&size=<size of image>. The resource is located at the address of the imagetransform.org domain, which may or may not be within the CDN **110**.

In this example, all known bricks **130** have infrastructure tags **508** for their geographical locations (e.g., city, metropolitan area, country, supranational entity). For example, a brick **130** in London would be tagged with the tags LONDON, UK, EU, and EMEA. A brick in **130** Paris would be tagged with PARIS, FRANCE, EU, and EMEA. A brick **130** in Chicago would be tagged CHICAGO, IL, USA, NA, and AMERICAS, and so on.

The one or more databases or data structures hold the infrastructure tags and tagsets **508** and addresses of all bricks **130** or resources **134** that comply with each tag or tagset. The tagset could be named to be the same as the tag, by convention, i.e., LONDON, USA, etc. Tagsets could be conjunctions of two or more tags. For example, a tagset called LONDHPERF could contain both the LONDON and HIGH-PERFORMANCE tags. A query to the metadata service **524** for a given tag or tagset would return all bricks **130** and resources **134** that have the tag(s).

All known bricks **130** or other resources **134** are arbitrarily grouped with a tagset having any number of tags. For geographic tags, a brick **130** cannot be in London and somewhere else at the same time, so generally a geographic tag are not conjoined with other geographic tags in a tagset. Not all the tags which exist need to be in a tagset—some might be reserved for future use. Similarly, not all tagsets need be utilized in any policy.

This example policy is expressed in a pseudo language below as four PRS parameters. The first PRS parameter is the policy name. For the second PRS parameter, one or more criteria can be specified as positive or negative logic to test for a condition before applying the policy. In this example, the criteria defines applicability to content files of the JPG MIME type. The disposition in the third PRS parameter is the storage conditions specifying tags and the minimum number of copies.

Policy: "ExamplePol"
Criteria: [{name='.*.jpg$'}]
Disposition: [{Tagset="USA", MinBricks=2}, {Tagset="EU", MinBricks=1}]
Thumbnail Mutator=[http://www.imagetransform.org/thumbnailer?path=%p&size=%s]

In order to decide the applicability of this policy, the PRS **260** would first would look at the name extension of the file as a criteria. If it matches the regular expression '.*.jpg$' (that is, it ends with the text 'jpg'), then this policy applies. Any other files would not be deemed to be covered by this policy.

When executing the policy, the PRS **260** would select two bricks which have all the tags in the tagset USA, and one brick from the set which have all the tags in tagset EU. The bricks could be chosen by any criteria such as randomly, round robin, available space, cost of storage, proximity in network terms, bandwidth cost to transport the file, etc. Once three receipts come back from those bricks marked COMPLETE, the source JPEG file itself goes into the COMPLETE state, and the thumbnail mutator in the PRS parameter list gets called, substituting the metavariables % p with the full path to the object, and % s with the size in bytes of the image object. Other policies could have any number of mutators for storage, transcoding, replication, or other processing for a file or asset.

The policy mapping store **518** records which policies are mapped to various levels of the hierarchy. Various embodi-

ments could map policies to the national jurisdiction, regional jurisdiction, CDN-wide, POP-applicable, customer, sub-customer, directory, sub-directory, MIME-type, individual file, etc. to achieve any level of granularity. FIGS. **12A**, **12B** and **12C**, show embodiments of a different policy prioritization hierarchies **1200**. Each block represents a level in the hierarchy and can have a number of policies assigned to it. The policies for a particular level can be organized in a priority as between the others at that level. Policies in a higher block take precedence over those in a lower block during the disambiguation processes performed by the policy compiler **528**. The present embodiment only has two levels in the hierarchy as illustrated in FIG. **12A**. Some policies disassociate themselves with a particular level of the hierarchy once performed. For example, a policy could be used to update coding on the content library from a legacy format where all new content is received in the updated coding. After running the policy once, it can be disassociated from the customer with a PRS parameter in the policy that removes the association in the policy mapping store **518**.

The directory structure **532** for this example is illustrated in FIG. **6** where each customer has a directory with optional subdirectories. Each directory or subdirectory can hold file names **604** for content objects of the customer. The directory structure **532** is stored in the metadata directory **275** in this embodiment. Table II shows an example of a portion of the policy mapping **518** for the hierarchy in FIG. **12A** and the directory structure of FIG. **6**. The /ZBS_Radio client has subdirectories for /streams and /podcasts. All the files in the /ZBS_Radio/streams path has both ingest and host policies that are applied, while all the files in the /ZBS_Radio/podcasts path has ingest, transcode and host policies that are applied.

TABLE II

Policy Mapping

Directory	Subdirectory	File	Policy
ABS Movie Channel	—	—	Ingest, Replication, Host
...	...	...	...
ZBS Radio	Streams	—	Ingest, Host
	Podcasts	—	Ingest, Transcode, Host
Realure eBooks	—	Silverthorne Trails.epub	Ingest, Replication
		Keystone Boarding.epub	Ingest, Replication
		Aventneer.epub	Ingest
		Aventneer Audio.mp3	Ingest, Replication, Transcode

A UUID generator **532** assigns a 256 bit code to each path and filename stored in the CMA. The UUID becomes the file name for content objects stored with the various bricks **130** associate with the CMA. The UUID is a one-way function in that the path and file name cannot be determined from the UUID alone, but a query to the mapper cabinet **324** can give the bricks storing a particular UUID and the path and file name.

The metadata directory **275** maintains metadata, tags and tagsets that are used by the policies to process content objects. There is customer metadata **516** describing details about the customer. The customer metadata **516** is entered when the customer configures their account with the CDN and includes account holding personal information, account, any sub-account(s), zone, channel, confidentiality level, etc. The directory and subdirectory structure for a customer is stored in the directory structure **532** in this embodiment, but could be stored with other customer metadata **516** in other embodiments.

Also stored is user metadata **540** that is discerned about the end user **128**. Typically, the end user **128** does not intentionally interface with the CDN **110** so the user metadata **540** is largely discerned indirectly. Metadata includes usage habits, content preferences, demographic information, user identifier, POP location receiving request, and end user system location. The content player, IP address, cookies or other techniques may be used to discern one user from another. Privacy concerns can limit the availability of user metadata **540**.

Infrastructure tags and tagsets **508** are assigned to bricks **130** and resources **134**. The number of tags increase as customers want greater granularity in applying policies. Infrastructure tags include: carbon use, energy use, storage type (e.g., solid state, magnetic, optical, spinning, tape), equipment manufacturer, reliability (e.g., dual-location replication, RAID level, tape archive), write only, write once, interface speed (e.g., SATA, SAS, Ethernet, 10 Gb), retrieval latency, storage cost, physical security surrounding equipment, geographical location of content, various performance discriminators, equipment location to city, region, or country, POP, IPV4 or IPV6 compatibility, CDN hosted, user hosted, level of QoS. The tags can be applied to bricks and resources regardless of them being inside or outside the CDN **110**.

Content metadata **512** relates to content objects with the CMA. The content metadata **512** can additionally be stored in the content object itself and/or its file name. The content metadata includes MIME type, encoding, container format, copyright terms, cost terms, copyright year, actors, director, studio, program summary, content rating, critical review ranking, title, transcript, ad insertion locations, encryption, request popularity, etc. Some content metadata **512** is not stored in the database or store, but is discerned through interaction with the content file. For example, MIME type is readily discernable from the file itself without refereeing the content metadata **512** in the store or database.

Bricks **130** and resources **134** are expressly enrolled to work with the CMA. A registration interface **536** is used to enter the address or domain name for the brick **130** or resource **134** that is stored in a brick/resource information store **544**. The bricks **130** and resources **134** periodically report health and status information through a report interface **548** that is also stored in the brick/resource information store **544**. The metadata directory **275** can periodically request status and health information using the report interface **548** or can test the brick **130** or resource **134**. Where calls to the brick **130** or resource **134** fail or perform poorly, the brick/resource information **544** can be updated to reflect status.

With reference to FIG. **7**, a swim diagram of an embodiment of a process for using the CMA to retrieve a content object with a legacy edge server **235** is shown. The depicted portion of the process begins in block **704** where the end user system **102** requests a file from an the legacy edge server **235**. Where the legacy edge server **235** cannot fulfill the request, the file is requested from the mapper transport **245** in block **708**. The request from the legacy edge server **235** is typically a URL which is converted by the mapper transport **245** in block **712**. A multicast query is made to the lookup listeners **328** in block **714** that is received in block **716**. To achieve parallel requests to the lookup listeners **328**, multiple unicast requests are made overlapping in time.

In block **724**, a query is made from the lookup listener **328** to the mapper cabinet **324**. The first lookup listener **328** to find the result in its respective mapper cabinet **324**, responds to the

mapper transport **245** with the brick addresses and UUID in block **728** that receives them in block **734**. The mapper transport **245** determines which of the brick addresses to use where there are multiple ones in block **748**. The determination can be random or according to some other scheme. The UUID is requested from the address of the selected brick in block **752**. If unsuccessful as determined in block **756**, another address is attempted by looping back to block **748**.

Where the file with the UUID for the name is found on the brick **130**, the file is renamed and sent to the legacy edge server **235** in block **760**. The file is received in block **764** and returned to the end user system **102** in block **768**. In this way, the CMA is used like an origin server by any legacy process with the mapper transport **245** translating the interaction.

Referring next to FIG. **8**, a swim diagram of another embodiment of a process for using the CMA to retrieve a content object is shown. This configuration does not use the mapper transport **245** as the edge server **230** knows how to interact with the content directory **205** directly. In block **804**, a request for a file is made by the end user system **102** to the edge server **230**. The path and filename is requested using multicast to the lookup listeners **328**, where the content object is not found on the edge server **230** in block **814**. The content directory **205** receives the request in block **816**, determines or looks-up the UUID in block **820** and the brick names in block **824** from the mapper cabinet **324** to respond first with the answer.

The brick addresses and UUID are sent by the content directory **205** in block **828** and received by the edge server **230** in block **834**. The edge server **230** determines which brick address to try first in block **848** before requesting that UUID from the brick **130** in block **852**. If not found in block **856**, another brick address is attempted by looping back to block **848**. Where the file is found in block **856**, it is renamed and returned to the end user system **102** in block **868**.

With reference to FIG. **9**, a flowchart of an embodiment of a process **900** for applying policies to a content object is shown. The depicted portion of the process begins in block **904**, where the customer account, services and policies are configured, which is explained in greater detail in relation to FIG. **10** below. In block **908**, a content object file is received from the customer using a landing pad **210**, **215** or other API. Some embodiments can add content object files when requested from the CDN **110** and are located after a cache miss. Other embodiments can designate a path that is automatically reconciled with the CDN **110** using a policy.

Some policies are triggered by an action such as intake, user request, or other action that would affect the content object file. Other policies are run periodically or according to a schedule, for example, checking a directory for newly encoded files and moving the file back out to the origin server **112** of the content originator **106**. In any event, the potential policies are determined in block **912**, which is explained in greater detail in relation to FIG. **11**. The policies generally applicable to the content file is determined by analysis of all policies associated with the hierarchy **1200** in block **916**. Where there are more than one policy, a disambiguation process is performed by the policy compiler **528** to find the policy that has the highest priority.

In block **920**, the policy is interpreted and performed. The policy is represented as a number of PRS parameters that are interpreted to perform some processing on the content object file. The functionality of block **920** is explained in greater detail in relation to FIG. **13** below. If there are more policies applicable to the content object file that are still waiting to complete as determined in block **922**, processing loops back to block **912**. The just performed policy may have changed the content object file such that it qualifies for more or less policies so that is reanalyzed in block **912**. At some point, it is determined block **922** that all policies have been performed and the normal operation of the CDN **110** utilizes the content object file in its processed form is performed in block **924**.

Upon changes to user, customer or content metadata **540**, **516**, **512** referenced in the applicable policies or tags **508** associate with where a content object file is stored, the policies are run again. For each content object, the relevant input metadata **512**, **516**, **540** or tags **508** used in the policies are tracked. Where there are changes to any of the metadata **512**, **516**, **540**, tags **508** or policies as determined in block **928**, processing loops back to block **912** to rerun the policies.

Referring next to FIG. **10**, a flowchart of an embodiment of a process **904** for configuring a customer account is shown. The depicted portion of the process begins in block **1004** where the customer connects to the CDN **110** and authenticates their identity. Certain demographic and payment options may be entered along with customer metadata **516**. In block **1008**, a landing pad **210** is selected and customized. A landing pad **210** is an ingest point and can be configured in any number of ways to efficiently provide content objects to the CDN **110**.

In block **1012**, the customer can choose the number of POPs and/or their location that will host the landing pad. The customer can select that each POP would have an instantiated landing pad **210** or instantiate one upon request. To provide for high volume accounts, there can be a number of landing pads per POP that even scales up or down with demand. The customer can design the directory structure **532** for their account by renaming directories and adding sub-directories nested down any number of levels in block **1016**.

The customer can customize policy templates, design new policies or modify their existing policies in block **1020**. The policies are mapped to the hierarchy **1200** in block **1022**. Where there are multiple policies for a particular level in the hierarchy, they are put in order of importance to allow resolving potential conflicts during disambiguation. In block **1024**, the landing pads **210** start normal operation at the selected POP(s) **120**. The directory structure and loaded files for the customer can be viewed and modified through the landing pad in block **1028**.

With reference to FIG. **11**, a flowchart of an embodiment of a process **912** for disambiguation of policies is shown. The process **912** is best understood in reference to one hierarchy from FIG. **12A**, **12B** or **12C**. The depicted portion of the process **912** begins in block **1104** where all policies possibly applicable to the file are found throughout the hierarchy. The policies are all prioritized in block **1108**. The tags, metadata and criteria of the PRS parameters in the policies are gathered in block **1112**. Criteria can and other filters in the policies can make many policies irrelevant to a particular content file.

In block **1116**, the inapplicable and conflicting policies are removed from the list. Each policy has a criteria that may make the policy inapplicable to the file. Additionally, there can be conflicting policies where the lower priority policy is removed. The policy compiler **528** also checks for syntax errors or other problems in block **1120**. The invalid policies are removed from the list in block **1124**. The list of potential policies are known at the point so that the highest priority can be executed.

With reference to FIG. **13**, a flowchart of an embodiment of a process **920** for performing a policy is shown. The depicted portion of the process begins in block **1302** where the policy compiler **528** checks the policy for errors prior to running the policy after being loaded by the policy manager **504**. The policy manager **504** checks the policy for any criteria identi-

fied in a PRS parameter. Where there is a criteria, a determination is made in block **1308** to see if the criteria is satisfied to allow further evaluation of the policy. Should the criteria exclude further processing, processing passes from block **1308** to block **1312** where processing of the policy is complete.

Should the policy criteria be satisfied in block **1308**, processing continues to **1316** to see if there is a mutator PRS parameter in the policy. Where there is no mutator, processing goes to block **1344** where the content object a disposition PRS parameter defines how the resulting content object is stored. The storage may be dependent on variables, metadata and/or tags. The metadata directory **275** can be queried for bricks **130** that comply with a given tag that can be specified in the disposition PRS parameter to find the one or more bricks **130** to use. For example, the disposition could be to store the content object in three locations with a USA tag. A query would be made to the metadata directory **275** and it would choose three from all the bricks **130** with the USA tag and specify those bricks **130**. The policy manager **504** would write the content object to the addresses specified by the metadata directory **275**. Where there is one or more mutators as determined in block **1316**, processing continues to block **1320**.

In block **1320**, the address of the resource is parsed from the mutator URL. Where the address is a domain name, that is resolved to an IP address using a domain name service (DNS). The resource **134** API uses a URL in this embodiment so that requesting the URL in block **1324** passes the source file location and other embedded variables. In some embodiments, the mutator specifies a group of resources that comply with a tag. The metadata directory **275** is queried to choose from the group of resources with the tag and return the address of the particular resource to use. For example, the mutator may specify using a transcoder service in Russia. The metadata directory would find all transcoder services with a Russia tag and return one. The one chosen could be random, round robin, based upon loading status from the resource information database **544** or some other algorithm.

The resource **134** parses the source file location from the URL in block **1328**. The content object is retrieved by the resource **134** in block **1332**. In block **1336**, the resource **134** performs the requested processing according to any other variables passed in the URL or other API. If there is another mutator in the policy as determined in block **1340** processing loops back to block **1320** for processing. Where it is determined in block **1340** that there are no additional mutators in the policy, processing goes to block **1344** for execution of the disposition PRS parameter. In this way, a policy is performed to process a content object file.

Referring next to FIG. **14**, a flowchart of an embodiment of a process **1400** for enrolling a resource **134** or brick **130** into the CMA is shown. The metadata directory **275** knows addresses for the enrolled resources **134** and bricks **130** along with the infrastructure tags **508** associate with each. The depicted portion of the process **1400** begins in block **1404** where a new resource **134** or brick **130** is identified for inclusion in the CMA. A unique identifier or name or an address is assigned to the resource **134** or brick **130** in block **1408**. The address can be a virtual one that is resolved through DNS. Infrastructure tags for the new enrollee are entered into the CMA in block **1412**.

The resource **134** or brick **130** is added to the tag or tagset groups in block **1416**. A query for a tag or tagset can quickly return all the addresses for resources **134** or bricks **130** by arranging those groups beforehand. The resource is added to the CMA by being coupled to the switching fabric **240** even though it may be inside or outside the CDN **110**. Periodically, when polled or according to a schedule, the resource **134** or brick **130** reports health and other status to the metadata directory **275** in block **1424** for retention in the resource information store **544**. The metadata directory **275** can avoid assigning new content objects to a resource **134** or brick **130** that is less healthy or more overloaded than the other members in the tag or tagset group.

With reference to FIG. **15**, a flowchart of an embodiment of a process **1500** for delivering a content object using the CMA is shown. The depicted portion of the process begins in block **1504** where the end user system **102** queries for content from the CDN **110**. Through Anycast, redirection, switching, or DNS resolution, the request finds its way to a 'nearby' POP **120**. Closeness is a function of network proximity which generally corresponds to geographic proximity, but not necessarily so.

In block **1512**, a caching or hosting edge server **230** is assigned. Where the edge server **230** cannot satisfy the request locally, a request is made to the CMA for the content object. Through the process outlined in FIGS. **7** and **8** above, the resource **134** or brick **130** is located and the content is relayed or proxied to the edge server **230** in block **1520**. In block **1524**, the edge server fulfils the request. The end user plays the content in block **1528**.

Referring next to FIG. **16**A, a flowchart of an embodiment of a process **1600-1** for elastically managing propagation of content objects is shown. This workflow is implemented with the CMA using various policies to scale-up or scale-down the propagation of content objects in a fluid manner. The depicted portion of the process **1600** begins in block **1604** where a policy measures popularity content metadata **512** for a content object. A counter in the CDN tracks the popular content and updates the content metadata **512** in the metadata directory **275** accordingly. Popularity could be measured on a scale of one to one hundred with the most popular being at one hundred and the least popular being at one. The policy measures acceleration in popularity by keeping a number of past data points for the content objects.

In block **1608**, the quickest changes are addressed by adding the name of the content file to a list in a policy. The footprint algorithm could be run separately in each POP **120** to measure local popularity or could be run to measure popularity in all POPs **120**. The footprint is a function of how many requests are likely, the size of the content object, the QoS desired for delivery, and the level of CDN service ordered by the customer. In block **1612**, the policy is passed variables that manage propagation of the content objects to accomplish a desired footprint. The variables would updates a disposition PRS parameter accordingly in the policy.

In block **1616**, the policy implementing the footprint change for the content objects experiencing the quickest changes is performed by the policy manager **504**. The bricks **130** to store or delete in each tag or tagset group are determined in block **1620**. The tag and tagsets chosen as criteria define the footprint even though the metadata directory **275** chooses the individual bricks **130** that have the tag or tagset assigned to it. The policy adds or deletes copies of the content objects with the highest acceleration accordingly.

Referring next to FIG. **16**B, a flowchart of an embodiment of a process **1600-1** for elastically managing propagation of content objects is shown. Unlike the embodiment of FIG. **16**A, this embodiment measures the velocity as a function of the popularity metadata. When the velocity changes, a policy triggers a reevaluation of the footprint in block **1603**. The velocity may have to change by a certain percentage before triggering the reevaluation or may be checked periodically.

Other embodiments could measure poor QoS feedback from end user systems **102** or complaints before modifying the footprint in block **1608**. When the footprint changes, processing continues through the remainder of the blocks in a manner similar to FIG. **16**A.

With reference to FIG. **17**, a flowchart of an embodiment of a process **1700** for using policies to change the jurisdiction used to process a content object is shown. The POP **120** where the content object lands can be different from where it is processed and stored. Policies are used to accomplish customized processing for the different movements of a source file and its various processed versions. Resources **134** or bricks **130** may be less costly or underutilized in different parts of the CDN **110**. For example, transcoder resources in India could be underused in the middle of the day in the United States due to the time differences. A policy could move transcodes to resources **134** in India.

The depicted portion of the process **1700** begins in block **1704** where the customer manually or automatically provides the content object to a landing pad **210** in one jurisdiction. A jurisdiction could be any legal, geographical or political boundary in this embodiment. The demarcations of a jurisdiction can be a custom geography perhaps defined in a license or other agreement. It is determined in block **1708** that a jurisdictional policy is active and applicable to the content object. In block **1712**, the jurisdictional domain to perform the processing defined in the policy is determined. The processing could be a resource **134** and/or a brick **130**. The content object is sent to brick **130** and/or resource **134** in target jurisdictional zone or domain. In block **1720**, the content object is processed in the chosen jurisdictional zone **1720**.

In block **1724**, the disposition PRS parameter defining how to store the content object is analyzed. The jurisdictional zone for storage of the processed file can be different from the jurisdictional zone that received the file and processed the file. Bricks **130** with the proper tags or tagsets of the jurisdictional zone are selected from to assign individual bricks **130**. In other examples, the policy may require that certain files be processed in the same jurisdictional zone as received. Further, embodiments may require receipt, processing and storage to be in the same jurisdictional zone. Without a policy restricting movement, processing and storage could choose resources and bricks outside the jurisdictional zone based upon, for example, those resources and bricks being under utilized.

A number of variations and modifications of the disclosed embodiments can also be used. For example, the above embodiments have a particular arrangement of the CMA, but blocks could be combined or split in any manner to still achieve the same functionality. Additionally, portions of the CMA could reside outside of the CDN. For example, the PRS, metadata directory, and/or content mapper could be maintained outside of the CDN.

Specific details are given in the above description to provide a thorough understanding of the embodiments. However, it is understood that the embodiments may be practiced without these specific details. For example, circuits may be shown in block diagrams in order not to obscure the embodiments in unnecessary detail. In other instances, well-known circuits, processes, algorithms, structures, and techniques may be shown without unnecessary detail in order to avoid obscuring the embodiments.

Implementation of the techniques, blocks, steps and means described above may be done in various ways. For example, these techniques, blocks, steps and means may be implemented in hardware, software, or a combination thereof. For a hardware implementation, the processing units may be implemented within one or more application specific integrated circuits (ASICs), digital signal processors (DSPs), digital signal processing devices (DSPDs), programmable logic devices (PLDs), field programmable gate arrays (FPGAs), processors, controllers, micro-controllers, microprocessors, other electronic units designed to perform the functions described above, and/or a combination thereof.

Also, it is noted that the embodiments may be described as a process which is depicted as a flowchart, a flow diagram, a data flow diagram, a structure diagram, or a block diagram. Although a flowchart may describe the operations as a sequential process, many of the operations can be performed in parallel or concurrently. In addition, the order of the operations may be re-arranged. A process is terminated when its operations are completed, but could have additional steps not included in the figure. A process may correspond to a method, a function, a procedure, a subroutine, a subprogram, etc. When a process corresponds to a function, its termination corresponds to a return of the function to the calling function or the main function.

Furthermore, embodiments may be implemented by hardware, software, scripting languages, firmware, middleware, microcode, hardware description languages, and/or any combination thereof. When implemented in software, firmware, middleware, scripting language, and/or microcode, the program code or code segments to perform the necessary tasks may be stored in a machine readable medium such as a storage medium. A code segment or machine-executable instruction may represent a procedure, a function, a subprogram, a program, a routine, a subroutine, a module, a software package, a script, a class, or any combination of instructions, data structures, and/or program statements. A code segment may be coupled to another code segment or a hardware circuit by passing and/or receiving information, data, arguments, parameters, and/or memory contents. Information, arguments, parameters, data, etc. may be passed, forwarded, or transmitted via any suitable means including memory sharing, message passing, token passing, network transmission, etc.

For a firmware and/or software implementation, the methodologies may be implemented with modules (e.g., procedures, functions, and so on) that perform the functions described herein. Any machine-readable medium tangibly embodying instructions may be used in implementing the methodologies described herein. For example, software codes may be stored in a memory. Memory may be implemented within the processor or external to the processor. As used herein the term "memory" refers to any type of long term, short term, volatile, nonvolatile, or other storage medium and is not to be limited to any particular type of memory or number of memories, or type of media upon which memory is stored.

Moreover, as disclosed herein, the term "storage medium" may represent one or more memories for storing data, including read only memory (ROM), random access memory (RAM), magnetic RAM, core memory, magnetic disk storage mediums, optical storage mediums, flash memory devices and/or other machine readable mediums for storing information. The term "machine-readable medium" includes, but is not limited to portable or fixed storage devices, optical storage devices, wireless channels, and/or various other storage mediums capable of storing that contain or carry instruction(s) and/or data.

While the principles of the disclosure have been described above in connection with specific apparatuses and methods, it

is to be clearly understood that this description is made only by way of example and not as limitation on the scope of the disclosure.

What is claimed is:

1. A content delivery network (CDN) having a plurality of geographically distributed points of presence (POPs) for processing content objects with a plurality of resources, the CDN comprising:

a landing pad to receive a content object from a client;

one or more databases comprising a list of the plurality of resources, each of the plurality of resources being associated with one or more tags, each tag indicating a characteristic of the associated resource; and

a policy reconciliation service (PRS) for maintaining and processing policies, the PRS being coupled to the one or more databases, the PRS comprising:

a policy store comprising a plurality of policies, each of the plurality of policies defining specific processing of content objects, the plurality of policies including a first policy and a second policy,

wherein each of the first policy and the second policy comprises an applicability parameter indicating criteria that a content object must satisfy in order for the content object to be processed in accordance with the respective first or second policy, the criteria indicated in the first policy's applicability parameter being different from the criteria indicated in the second policy's applicability parameter,

wherein the first policy comprises a disposition parameter indicating criteria that a resource must satisfy in order for the resource to effect the first policy, and

wherein the first policy comprises one or more mutators, each mutator comprising a template for inclusion of an address of a resource and/or a location of a received content object; and

a policy manager configured to:

determine that the first policy is applicable to the received content object and that the second policy is not applicable to the received content object, the determination being based on the first policy's criteria and metadata related to the received content object, a requester of the received content object and/or a provider of the received content object at the CDN; and

identify a resource of the plurality of resources for effecting the first policy based on the disposition parameter and a tag associated with the first policy.

2. The CDN having a plurality of geographically distributed POPs for processing content objects with a plurality of resources as recited in claim 1, wherein the mutator comprises a template for a universal resource locator.

3. The CDN having a plurality of geographically distributed POPs for processing content objects with a plurality of resources as recited in claim 1, wherein the resource comprises an HTTP-based application programming interface.

4. The CDN having a plurality of geographically distributed POPs for processing content objects with a plurality of resources as recited in claim 1, wherein the specific processing defined in the first policy comprises storing the content object.

5. The CDN having a plurality of geographically distributed POPs for processing content objects with a plurality of resources as recited in claim 1, wherein the one or more databases comprise the metadata related to a content object, a requester of the content object and/or a provider of the content object at the CDN.

6. The CDN having a plurality of geographically distributed POPs for processing content objects with a plurality of resources as recited in claim 1, wherein one of the one or more mutators comprises a template for inclusion of the address of the resource and a location of the received content object.

7. The CDN having a plurality of geographically distributed POPs for processing content objects with a plurality of resources as recited in claim 1, wherein the resource comprises a storage device located outside of the CDN.

8. A method for processing content objects with a plurality of resources associated with a content delivery network (CDN) having a plurality of geographically distributed points of presence (POPs), the method comprising:

Enrolling, using a registration interface, the plurality of resources to be accessible from the CDN;

categorizing each of the plurality of resources using a plurality of tags that categorize the plurality of resources, and the plurality of resources includes a resource;

receiving selection of a policy from a plurality of policies, wherein the plurality of policies define processes to perform on content objects stored at the CDN, wherein the selected policy includes an applicability criteria and a call to the resource;

receiving, at the CDN, metadata related to a content object, a requester of the content object and/or a provider of the content object

receiving the content object for storage at the CDN;

determining, with a policy manager, that the policy is applicable and that other policies are not applicable to the received content object based on an analysis of the metadata and the applicability criteria of the policy; and

calling, with the policy manager, the resource according to the call in the policy to cause the resource to perform specified processing on the received content object.

9. The method for processing content objects with the plurality of resources associated with the CDN having the plurality of geographically distributed POPs as recited in claim 8, wherein calling the resource comprises embedding a delivery path into a uniform resource locator (URL).

10. The method for processing content objects with the plurality of resources associated with the CDN having the plurality of geographically distributed POPs as recited in claim 8, further comprising:

detecting a change to the metadata or one of the plurality of resources associated with one of the tags; and

determining if the policy still applies to the received content object.

11. The method for processing content objects with the plurality of resources associated with the CDN having the plurality of geographically distributed POPs as recited in claim 8, further comprising choosing the resource from a larger group of resources all associated with a tag from the plurality of tags, wherein the tag is identified in the policy.

12. The method for processing content objects with the plurality of resources associated with the CDN having the plurality of geographically distributed POPs as recited in claim 8, further comprising the resource:

parsing a file location for the received content object from the call;

parsing a variable from the mutator;

reading the received content object from the file location; and

processing the received content object according to the variable with the resource.

13. The method for processing content objects with the plurality of resources associated with the CDN having the plurality of geographically distributed POPs as recited in claim **8**, further comprising resolving overlap between the policy and the other policies found applicable to the content object.

14. The method for processing content objects with the plurality of resources associated with the CDN having the plurality of geographically distributed POPs as recited in claim **8**, wherein:

the applicability criteria specifies a tag, and

the tag is used to select the resource from a plurality of resources.

15. The method for processing content objects with the plurality of resources associated with the CDN having the plurality of geographically distributed POPs as recited in claim **8**, wherein the specified processing includes storing the received content object on a storage medium that has a tag specified in the policy.

16. A content delivery network (CDN) having a plurality of geographically distributed points of presence (POPs) for processing content objects with a plurality of resources, the CDN comprising two or more hardware servers having one or more processors for:

enrolling the plurality of resources to be accessible from the CDN;

categorizing each of the plurality of resources using a plurality of tags that categorize the plurality of resources, and the plurality of resources includes a resource;

receiving selection of a policy from a plurality of policies, wherein the plurality of policies define processes to perform on content objects stored at the CDN, wherein the selected policy includes an applicability criteria and a call to the resource;

receiving, at the CDN, metadata related to a content object, a requester of the content object and/or a provider of the content object;

receiving the content object for storage at the CDN;

determining that the policy is applicable and that other policies are not applicable to the received content object based on an analysis of the metadata and the applicability criteria of the policy; and

calling the resource according to the call in the policy to cause the resource to perform specified processing on the received content object.

17. The CDN having the plurality of geographically distributed POPs for processing content objects with a plurality of resources as recited in claim **16**, wherein the resource is a storage brick.

18. The CDN having the plurality of geographically distributed POPs for processing content objects with a plurality of resources as recited in claim **16**, wherein at least some of the plurality of resources are located outside the CDN.

19. The CDN having the plurality of geographically distributed POPs for processing content objects with a plurality of resources as recited in claim **16**, wherein the resource is a storage medium, an application program interface (API) to a software processor, a transcoder, a compression engine, an ad insertion function, a syndication engine, an cryptoengine, a thumbnail generator, an ad insertion engine, or a content syndication service.

20. The CDN having the plurality of geographically distributed POPs for processing content objects with a plurality of resources as recited in claim **16**, wherein the policy further includes a storage disposition defining how to store the received content object.

* * * * *