

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

MICROSOFT CORPORATION,
Petitioner

v.

SANDPIPER CDN, LLC,
Patent Owner.

Case IPR2026-00180

Patent No. 10,701,173

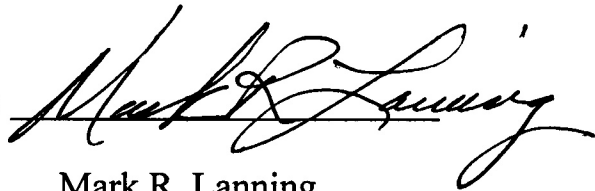
DECLARATION OF MARK R. LANNING

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true, and further, that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Date:

12/23/2025

By:



Mark R. Lanning

TABLE OF CONTENTS

I.	BACKGROUND AND QUALIFICATIONS	1
A.	Educational Background	1
B.	Professional Experience	2
II.	MATERIALS CONSIDERED	5
III.	LEGAL STANDARDS	5
IV.	TECHNOLOGY BACKGROUND.....	10
A.	The World Wide Web and the Internet	11
1.	Domain Names and URLs	11
2.	Web Request and Response	12
3.	Data Resources/Objects	12
4.	The HTTP and HTTPS Protocols	13
5.	Domain Name Servers and Redirection.....	15
B.	Content Delivery Networks.....	16
1.	CDN Server Databases	23
V.	'173 PATENT.....	25
A.	Specification	26
B.	Prosecution History	30
VI.	PRIOR ART.....	32
A.	O'Rourke (U.S. Patent No. 7,912,921)	32
B.	Middleton (U.S. Pub. No. 2011/0060812)	36
C.	Blumofe (U.S. Pub. No. 2012/0096106).....	39
D.	[MS-MSSO]: Microsoft - Media Streaming Server System Overview	41
VII.	LEVEL OF ORDINARY SKILL	42
VIII.	CLAIM CONSTRUCTION	43
IX.	FOUNDATIONS	44
A.	Ground 1A: Obviousness over O'Rourke	44

TABLE OF CONTENTS
(continued)

		Page
1.	Claims 1, 8	44
2.	Claims 2, 9	68
3.	Claims 4, 11: [The method as recited in claim 1]/[The computer program product as recited in claim 8] wherein the current cache policy is determined from a location distinct from the node.....	71
4.	Claims 5, 12: [The method as recited in claim 4]/[The computer program product as recited in claim 11] wherein the location is a database.	72
5.	Claims 6, 13: [The method as recited in claim 1]/[The computer program product as recited in claim 8] wherein the cache policy relates to caching content within the CDN.	73
B.	Ground 1B: Obviousness over O’Rourke and MS-MSSO	74
1.	Motivation to Combine O’Rourke and MS-MSSO	74
2.	Claims 3, 10: [The method as recited in claim 1]/[The computer program product as recited in claim 8] wherein the request comprises an HTTP request and wherein the content comprises one or more resources.	77
C.	Ground 1C: Obviousness over O’Rourke and Blumofe	78
1.	Motivation to Combine O’Rourke and Blumofe	78
2.	Claims 7, 14: [The method as recited in claim 6]/[The computer program product as recited in claim 13] wherein the cache policy also relates to caching content outside the CDN.	85
D.	Ground 2A: Obviousness over Middleton	88
1.	Claims 1, 8	88
2.	Claims 2, 9	120
3.	Claims 4, 11: [The method as recited in claim 1]/[The computer program product as recited in	

TABLE OF CONTENTS
(continued)

	Page
claim 8] wherein the current cache policy is determined from a location distinct from the node.....	126
4. Claim 5, 12: [The method as recited in claim 4]/[The computer program product as recited in claim 11] wherein the location is a database.	126
5. Claims 6, 13: The method as recited in claim 1]/[The computer program product as recited in claim 8] wherein the cache policy relates to caching content within the CDN.	128
E. Ground 2B: Obviousness over Middleton and MS-MSSO.....	129
1. Motivation to Combine Middleton and MS-MSSO	129
2. Claims 3, 10: [The method as recited in claim 1]/[The computer program product as recited in claim 8] wherein the request comprises an HTTP request and wherein the content comprises one or more resources.	132
F. Ground 2C: Obviousness over Middleton and Blumofe.....	134
1. Motivation to Combine Middleton and Blumofe	134
2. Claims 7, 14: [The method as recited in claim 6]/[The computer program product as recited in claim 13] wherein the cache policy also relates to caching content outside the CDN.	141
X. SUMMARY AND ADDITIONAL REMARKS	143

I, Mark R. Lanning, do hereby declare:

I. BACKGROUND AND QUALIFICATIONS

1. I have been retained by Petitioner Microsoft Corporation (“Microsoft” or “Petitioner”) to provide my opinions regarding claims 1-14 (the “Challenged Claims”) of U.S. Patent No. 10,701,173 (“the ’173 patent,” Ex-1001).

2. I am currently the President of Telecom Architects, Inc., and Reticle Consulting, LLC.

3. Available at Ex-1004 is a copy of my current curriculum vitae, which provides further details about my academic and professional background and qualifications.

4. I am being compensated for my services and reimbursed for reasonable expenses that I may incur while working on this matter. I have no financial interest in the outcome of this matter or any other matter that may exist between Petitioners and Sandpiper LLC (“Sandpiper” or “Patent Owner”). This compensation does not depend in any way on the conclusions I reach.

A. Educational Background

5. I received a B.S. in Computer Science from Southern Methodist University (SMU) in 1983.

B. Professional Experience

6. I have over 40 years of experience working in the telecommunications industry that began in the U.S. Army Signal Corp.

7. In 1983, Digital Switch Corporation (DSC), now a part of Alcatel, hired me, where I was a software development manager on the team responsible for updating DSC's PSTN telephone switch into a Mobile Switching Center (MSC) for Motorola to sell as a part of their cellular product offering in the U.S. and many other countries.

8. In 1991, I began working as a consultant to Motorola for its "SuperCell" base station product and as a consultant to British Telecom to upgrade its current analog cellular network, and I was one of the network architects responsible for the design and rollout of its Global System for Mobile Communications (GSM) network known as Cellnet. Beginning in the early 1990s, I was responsible for implementation of the SMS service, including working with suppliers of the SMS Center (SMSC), Mobile Switching Centers (MSCs) and cellular phones to define and roll out the functionality that was to be provided.

9. I was personally involved with Nokia, Ericsson, Motorola and other equipment suppliers in this effort.

10. Since 1995, I have also provided second generation (2G) and third generation (3G) Code Division Multiple Access ("CDMA") network architecture

and equipment design and implementation consulting services to companies such as Sprint, Nextel, Nokia, and Ericsson. While consulting to Nextel, which has since become part of Sprint, as one of the network architects for its iDEN network, one of my responsibilities was to define the network and mobile phone functionality required to support MMS and advanced data communications capability.

11. I am a member of the Institute of Electrical and Electronics Engineers (IEEE), including the IEEE Standards Association. I am also a member of the Association for Computing Machinery (ACM). While employed at DSC, I was a member of the ANSI T1 and T1X1 standard groups responsible for the definition and standardization of the Advanced Intelligent Network (AIN) and Signaling System 7 (SS7) protocol.

12. I am currently the president of two consulting companies: Telecom Architects, Inc. and Reticle Consulting, LLC. Telecom Architects provides consulting services to fixed and wireless telecom service providers and their equipment suppliers. I have been President of Telecom Architects since 1999.

13. My experience specifically relevant to this declaration is that I have worked on the design of packet network topologies and their routing algorithms, the development of computer systems and communication protocols for packet networks since 1976, which is before the current Internet was “invented” in 1983.

Specifically, I began this work when I was assigned to a special project associated with the ARPANET while I was serving in the U.S. Army Signal Corp.

14. Since 1976, I have been involved with the design, development and implementation of many of the different types of computer systems and communication protocols used in the Internet. While working at Digital Switch Corporation (DSC), the engineering team that I managed designed and developed the software and hardware from scratch for at least the TCP/IP, X.25, Session and Transport communication protocols so the DSC telephone switches could connect to packet-based networks. Later, while working for Tandem Computers, my development team wrote the software for the HTTP communication protocol so that our computer systems could provide Internet server functionality.

15. Beginning in mid-2007, I worked with Google regarding the design and development of its extensive Content Delivery Network, named the Google Global Cache (GGC), that was mainly used at the time to cache Search requests/responses. During this time, Google had at least 500,000 servers in its network and used a multi-tier topology. Akamai Technologies, Inc., referenced by multiple prior art references, was one of Google's competitors.

16. As a technical expert on legal cases, I have evaluated various Internet packet routing methods, Internet network topologies, high-speed Internet routers, client/server computer systems, cellular network and Internet interworking

protocols, multi-media streaming across the Internet, specialized systems and methods for redirecting Internet users, and specialized CDNs for supplying real-time marketing data and transfer of very large files.

II. MATERIALS CONSIDERED

17. In developing my opinions below relating to the '173 patent, I have considered the materials cited herein.

III. LEGAL STANDARDS

18. Counsel has informed me of the legal principles that apply in determining whether patent claims are unpatentable for obviousness. Paragraphs below beginning with “I understand,” “I have been informed,” or similar phrases capture my understanding of the law as provided to me by counsel. The other paragraphs are my opinion.

19. I understand that a claim may be unpatentable as obvious under § 103(a) if the subject matter described by the claim as a whole would have been obvious to a hypothetical “person of ordinary skill in the art” (“POSITA”) in view of a prior-art reference, or combination of prior-art references at the time the claimed invention was made.

20. I understand that “the time the claimed invention was made” is often referred to as the priority or critical date.

21. I understand that, before considering the prior art, the claims are first construed to determine their scope. I understand that the claims are read from the perspective of POSITA at the time of the alleged invention. I understand that the words of a claim are generally given their ordinary and customary meaning as understood by a POSITA in light of the entire patent specification in which they appear, as well as the prosecution history of the patent.

22. I understand, however, that the words of a claim may be given a meaning other than their plain and ordinary meaning if: (1) a patentee expressly and unambiguously defined a term in a way that is different from its plain meaning; or (2) a patentee “disclaims” the full scope of a claim term’s plain meaning, for instance, by making clear and unambiguous statements during prosecution that a claim term does or does not encompass certain features.

23. I understand that, after construing the claims, the teachings of the prior art are considered. I understand that the prior art includes references that were known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before the date the invention was allegedly made by the patent holder. For a printed publication to qualify as prior art, I understand that the Petitioner must demonstrate that the publication was disseminated or otherwise sufficiently accessible to the public. However, I also understand that a POSITA is

assumed to be aware of all relevant prior art in the field of endeavor covered by the patent-at-issue and all analogous prior art.

24. I understand that although the ultimate question of the obviousness of a claimed invention is a legal determination, obviousness is based on several factual inquiries, including: the scope and content of the prior art, the differences between the claimed subject matter and the prior art, the level of ordinary skill in the art at the time of the invention, and any “objective indicia” or “secondary considerations” of non-obviousness, which must all be considered.

25. I understand that, in determining the scope and content of the prior art, a reference is considered relevant prior art if it falls within the field of the inventor’s endeavor, or if it is reasonably pertinent to the particular problem that the inventor was trying to solve. A reference is reasonably pertinent if it logically would have commended itself to an inventor’s attention in considering his problem.

26. I understand that, to assess the differences between prior art and the claimed subject matter, the claimed invention must be considered as a whole. I understand that this involves showing that a POSITA, confronted by the same problems as the inventor and with no knowledge of the claimed invention, would have identified the elements in the prior art and combined them in the claimed manner.

27. I understand that there are several rationales for combining prior-art references. For instance, I understand that it is considered obvious to:

- combine prior-art elements according to known methods to yield predictable results;
- substitute one known element for another to obtain predictable results (also called a “simple substitution”);
- use the prior-art elements in a predictable way according to their established functions;
- apply a known technique to a known device (method or product) ready for improvement to yield predictable results (also called a “known technique”);
- choose from a finite number of identified, predictable solutions, with a reasonable expectation of success (also called “obvious to try”);
- or if there is some teaching, suggestion, or motivation in the prior art that would have led a POSITA to modify a prior-art reference or combine prior-art teachings to arrive at the claimed invention (also called “teaching-suggestion-motivation”).

I understand, however, that a POSITA does not need to have the same motivations or reasons to combine the prior art as the inventors, and can arrive at the claimed invention for completely different reasons than the inventors.

28. However, I also understand that the prior art cannot be modified or combined using “hindsight.” I understand that hindsight refers to situations in which the patent-at-issue is used as a framework to pick and choose pieces from the prior art to arrive at the claims, or adopting rationales identified by the patent-at-issue that those in the art would not have recognized. Instead, I understand that the obviousness inquiry is evaluated from the perspective of a POSITA as of the critical date of the patent-at-issue, with all the prior art before them, to determine whether the POSITA would have independently arrived at the claimed invention without the benefit of the patent-at-issue for guidance.

29. I understand that “objective indicia” or “secondary considerations” of non-obviousness include whether: (1) there was a long-felt need for the claimed invention; (2) the claimed invention has achieved commercial success; (3) there was copying of the claimed invention by others; (4) there were failed attempts by others to make the alleged invention; (5) there was praise of the invention in the field; and (6) there was skepticism the claimed invention could be achieved, among a few others.

30. As of the time that I prepared this declaration, the Patent Owner had not identified any secondary considerations of non-obviousness. Therefore, I reserve the right to elaborate on my understanding of these secondary considerations and to address any secondary considerations that Patent Owner may subsequently identify.

31. Finally, I understand that, in an *inter partes* review proceeding, the obviousness of a claim must be demonstrated by “a preponderance of the evidence,” and that this burden falls on the Petitioner. I understand that the preponderance of the evidence standard requires that a reasonable factfinder could find a material fact is more probable than the nonexistence of that fact. It does not allow for speculation regarding specific facts and is instead focused on whether the evidence more likely than not demonstrates the existence or non-existence of specific material facts. I understand that “preponderance of the evidence” is a lower standard than “clear and convincing evidence” (which requires a fact to be substantially more likely to be true than untrue), or “beyond a reasonable doubt” (which is an exceedingly high standard that I understand is generally reserved for criminal matters).

32. I have applied the “preponderance of the evidence” standard throughout my declaration.

IV. TECHNOLOGY BACKGROUND

33. The purported invention of the '173 Patent relates to content delivery and content delivery networks (CDNs). More specifically, “to caching in content delivery networks.” (Ex-1001, Field of the Invention, 1:34-36).

34. This section describes, at a high level, what a CDN is, typical CDN frameworks and components, how a CDN fits in the Internet and the protocols used to implement a CDN.

A. The World Wide Web and the Internet

35. At the inception of the World Wide Web (WWW), aka the Internet, in the early 1990s and up to the early 2000s, when an end user (client) wanted to access specific content over the Internet, they would typically use the browser on their computer to retrieve specific web pages, data, stream music or movies, etc. These requests would use a Uniform Resource Locator (URL) that was used to deliver the request to the correct web server.

1. Domain Names and URLs

36. These web servers were typically computer systems that had high-speed communication interfaces and databases for organizing and storing their content. In order to route the client's request to the correct web server over the Internet that consisted of thousands of potential routers and paths, each of these servers was assigned a unique Internet Domain Name that was included in the URL. We all have used URLs with different Domain Names when we wanted to access many different types of data on the Internet. For example, <http://www.example.com/home> is a URL that consists of three main parts. The first part is called the "scheme" and it describes the protocol being used to access the server. For this example, the "http://" defines that the HTTP communication protocol is to be used (described in more detail below). The second part is the server's Internet address, e.g., "www.example.com." where the "www" stands for world wide web

and is a required part of the URL, the [“example.com”](#) portion is the domain name of the server. The third part of the URL names a specific resource on the web server, “/home” is referencing the home page on the server called “home.” The web server owners or their representative typically maintained the content on their server with updates as they were needed.

2. Web Request and Response

37. The diagram below shows, at a very high level, a client (user with a browser) making a request to a web server for a specific resource and the server responds by providing the resource using the HTTP communication protocol to transfer both the request and response.

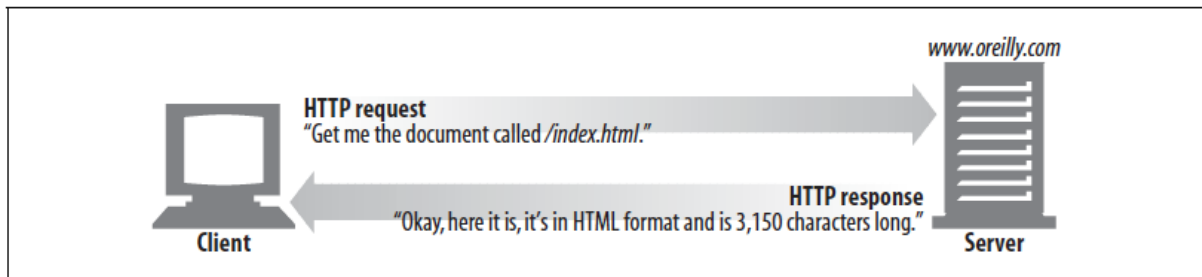


Figure 1-1. Web clients and servers

Ex-1019, 4 (HTTP The Definitive Guide by Gourley.pdf).

3. Data Resources/Objects

38. Web servers host web resources (also referred to as objects). A web server is the source of web content. “The simplest kind of web resource is a static file on the web server’s filesystem. These files can contain anything: they might be

text files, HTML files, Microsoft Word files, Adobe Acrobat files, JPEG image files, AVI movie files, or any other format you can think of.” Ex-1019, 4.

4. The HTTP and HTTPS Protocols

39. The Hypertext Transfer Protocol (HTTP) and the HTTP Secure (HTTPS) are request-response protocols that are used by programs to communicate over the Internet. Both are used to provide two-way communication between the web browser (client) and a web server. HTTPS provides essentially the same functionality as the HTTP protocol but it is used to encrypt the plain text messages used by HTTP to prevent eavesdropping or tampering. The diagram below shows this relationship.

a. HTTP/HTTPS Message Formats

40. The structure of HTTP/HTTPS messages include three parts called: 1) the Start line; 2) the Header fields; and 3) the Body. The start line indicates “what to do for a request or what happened for a response.” The Header fields “follow the start line. Each header field consists of a name and a value, separated by a colon (:) for each parsing.” The Body follows the Header field, is optional and can contain “any kind of data. . . . the body can contain arbitrary binary data (e.g., images, videos, audio tracks, software applications). Of course, the body can also contain text.” Ex-1019, 11.

41. The start line of every HTTP message includes a request command called the “HTTP method.” The “method tells the server what action to perform (fetch a web page, run a gateway program, delete a file, etc.).” Ex-1019, 8. The table below shows five common HTTP methods.

Table 1-2. Some common HTTP methods

HTTP method	Description
GET	Send named resource from the server to the client.
PUT	Store data from client into a named server resource.
DELETE	Delete the named resource from a server.
POST	Send client data into a server gateway application.
HEAD	Send just the HTTP headers from the response for the named resource.

Id. pp. 8-9.

42. Every HTTP response message is returned with a status code. “The status code is a three-digit numeric code that tells the client if the request succeeded, or if other actions are required. A few common status codes are shown in Table 1-3 below.” *Id.*, 9. The 302 Redirect code is used extensively to route requests to the correct server in a CDN network (described below).

Table 1-3. Some common HTTP status codes

HTTP status code	Description
200	OK. Document returned correctly.
302	Redirect. Go someplace else to get the resource.
404	Not Found. Can’t find this resource.

Id., 9.

43. The figure below shows the message details for a typical HTTP transaction between a client that is requesting information about a saw blade from Joe's hardware that is on special. I note the "Get" method is used in the request and the 200 "Status Code" is used in the response.

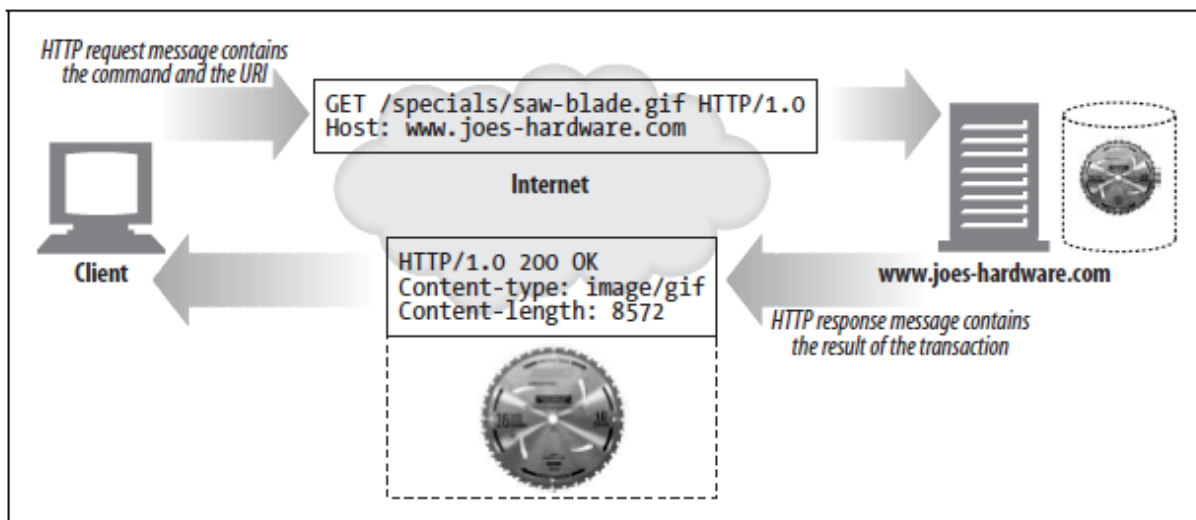


Figure 1-5. HTTP transactions consist of request and response messages
Id., 8.

5. Domain Name Servers and Redirection

44. The Internet routing protocol uses an IP address, which is a series of numbers, to route packets between the client and server and it cannot use the text www.joes.hardware. A special server in the Internet called a Domain Name Server (DNS) is used to convert these textual domain names that are received from the client into the correct IP addresses. A CDN uses a DNS to convert a requested URL into an IP address for an appropriate edge server, e.g., the CDN edge server that is closest to the requesting client. Thus, the request is not delivered to the real web server.

Instead, the CDN handles the request, which offloads the real web server (also called the origin server in most CDNs).

45. Redirection is an HTTP method that directs the requesting client to use a different URL for its request by returning the HTTP 302 response code described above along with the new URL it is to use. It can be used for many reasons. For Content Delivery Networks, the Redirect method is typically used to direct the client to a different CDN server for many different reasons that are described in the prior art and would have been understood by a person of ordinary skill in the art.

B. Content Delivery Networks

46. As internet usage grew exponentially in the early 2000s, those in the art recognized that “serving Web content from a single location...present[ed] serious problems for site scalability, reliability, and performance,” as servers would overload and crash if they received too many requests at one time. Ex-1013, 50.

47. Additionally, significant physical distance between a requesting network element and responding server often leads to greater latency (or delay) between a request and a response. *See generally* Ex-1010, 45, 51-52. For instance, if a user in Australia tries to stream a movie hosted on a United States-based server, the stream may experience notable delay and buffering as the movie data navigates globe-spanning connections to deliver the movie data to the end user.

48. To solve these problems, companies began building networks specifically intended to distribute content around the world faster while avoiding overloading a content provider's server. These networks would employ "different levels of local, regional, [and] international caches at geographically distributed locations." Ex-1010, 9, 16.

49. CDNs were well-established years before the '173 patent. *See, e.g.*, Ex1020, Abstract, Figs. 1, 4, ¶¶5-9; Ex-1021, Abstract, 1:15-2:3, Fig. 1. Companies including Akamai, Digital Island, and Speedera were first to implement CDNs. *See, e.g.*, Ex-1020, Cover (22), ¶5 (describing conventional CDNs in the publication of a patent filed in 2001).

50. The content provider's server was typically referred to as an "origin" server, since it was the originating source of all content for a particular content provider. Ex-1010, 5, 37 ("The server where the definitive version of the content resides is called the origin server. It is updated by the content provider."). The content provider's origin server would then connect to a CDN—typically offered by a third-party company—which hosted "*caching and/or replica servers* located in different geographical locations to replicate content." *Id.*, 5. "A cache server makes copies (i.e. caches) of content at the edge of the network in order to bypass the need of accessing origin server to satisfy every content request." *Id.*, 37. These "caching" and/or "replica" servers were also called "edge," "parent," "surrogate," or other

names, depending on their purpose in the network. *See, e.g., id.; see also* Ex-1015, Fig. 2.

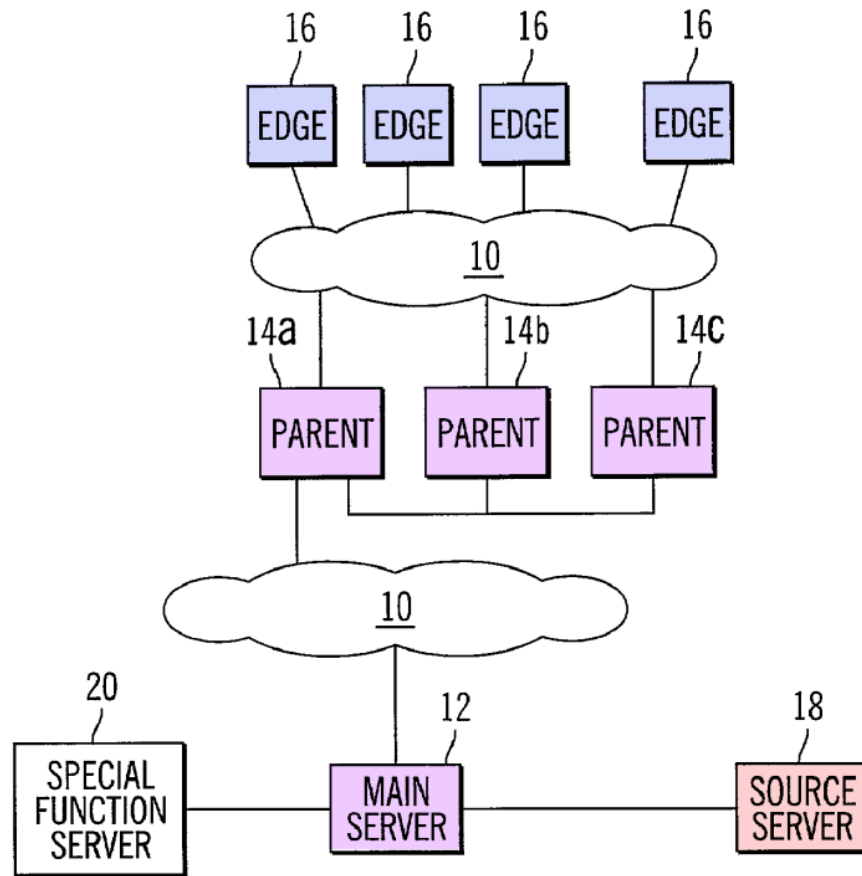


FIG. 2

Ex-1015, Fig. 2 (annotated)¹.

51. To distribute the load of both requests and responses, as well as reduce delivery times, CDNs were often arranged in a hierarchy. Ex-1014, 57; Ex-1010, 9. Fig. 1.1 below is one example of a hierarchical CDN topology. The content

¹ All annotations are added unless I note otherwise.

provider's origin server (shown in red, below) would sit furthest from the end user in the hierarchy. The CDN would then include, for instance, intermediate servers—called “surrogate” origin servers and/or parent servers (purple)—that may be geographically distributed and would cache at least some parts of the content owner's library. These surrogate-origin servers and/or parent servers would then support clusters of also-geographically distributed “edge” or “child” servers located closest to the end user, which would also cache at least part some of the content owner's library.

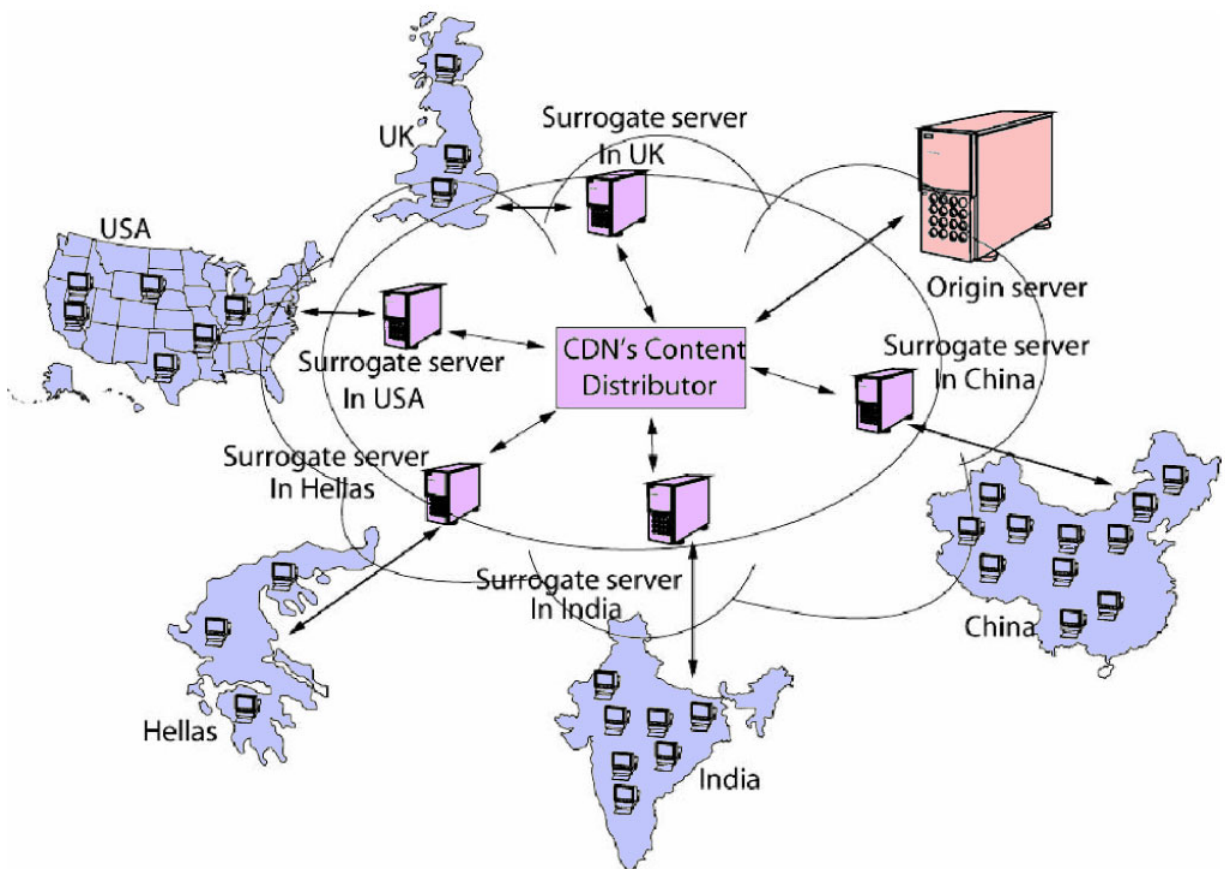


Fig. 1.1 Model of a CDN
Ex-1010, 5 (annotated).

52. Content would be replicated across servers based on a number of factors. Fig. 2.6 below provides one model for how data could be replicated across servers. For instance, content providers could choose between “full-site” caching (all content is hosted by the CDN) or “partial-site” caching (select content is hosted by the CDN). Ex-1010, 44. When “partial-site” caching is used, what content to host in the CDN can be selected based on empirical (heuristic-based manual selection), popularity (where “the most popular objects are replicated” throughout the CDN), objects (selection of specific types of objects to host), or clusters (hosting content that has been correlated based on specified factors). *Id.*

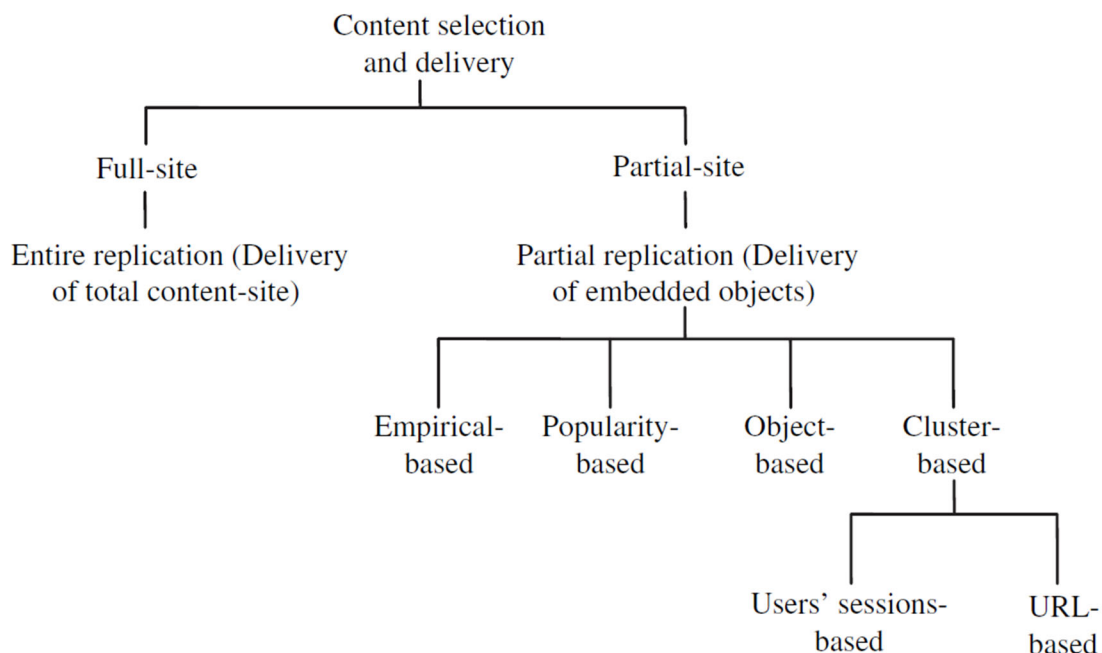
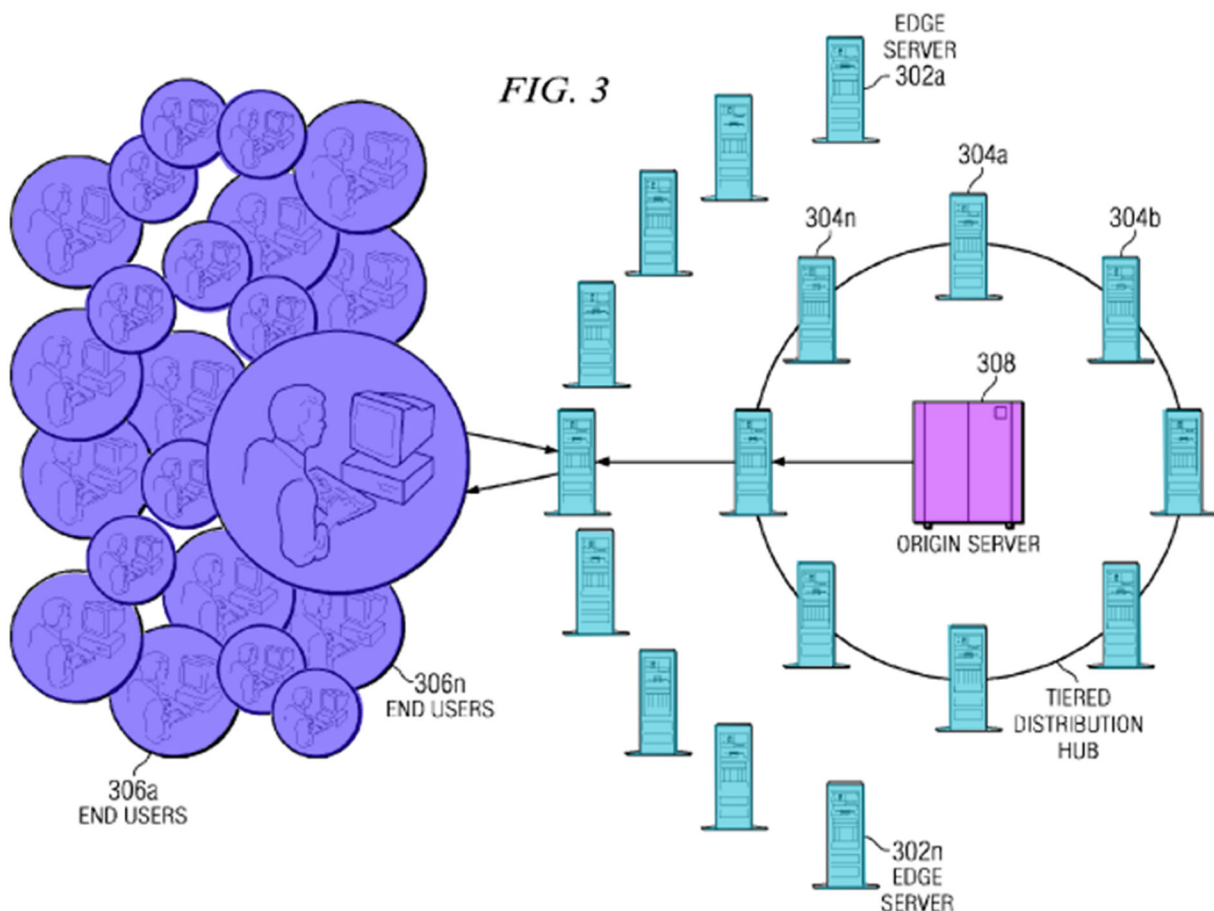


Fig. 2.6 Taxonomy of content selection and delivery

53. Fig. 3 below shows yet a different CDN hierarchical topology that was used. Thus, several different CDN topologies were known, and the exact configuration, including the number of servers, placement of servers, storage procedures, request-routing procedures, and delivery procedures depended “on the content/services it provides to its users,” and factors such as cost, quality of service, latency, redundancy, and others. Ex-1010, 36; *see also id.*, 37-58, 63-64, 67-69 (describing considerations for each choice and providing example choices by certain CDN providers), 58-59 (summarizing key metrics that impacted design choices).



Ex-1021, Fig. 3 (annotated), 3:5-9, 4:9-35.

54. In hierarchical systems comprising edge, parent, and origin servers, when an end user requests the delivery of specific content, the request would typically be routed to the optimal edge server that would try to fill the request itself, minimizing the delivery time and avoiding burdening the content owner's origin server. Ex-1010, 6; Ex-1013, 50. But if the edge server could not locate the content in its cache, the edge server could either search for the content on, or redirect the request to, another server (such as a parent or surrogate origin server) that may store the requested content instead. Ex-1014, 57.

55. "To locate content in CDN peering, a centralized directory model, Distributed Hash Table (DHT), flooded request model, or document routing model can be used." Ex-1010, 58. A centralized directory model uses a centralized directory to look up which server should be storing the requested content. *Id.* This approach, however, "is subject to a single point of failure," and scalability "is limited to the capacity of the directory." *Id.* With DHTs, "peers are indexed through hashing keys within a distributed system," which effectively builds in "load balancing by offloading excess loads to less-loaded peers." *Id.* Flooded request models involve servers that propagate requests to all of their connected peers until one peer in the network responds saying it has the requested content. *Id.* But this approach has drawbacks in that "it generates unnecessary network traffic and hence, it requires

enormous bandwidth.” *Id.* Finally, the document routing model uses requests that contain a file identifier, which is forwarded to a peer whose identifier is most similar to the file identifier, which locates the file and provides it to the requesting peer. *Id.*

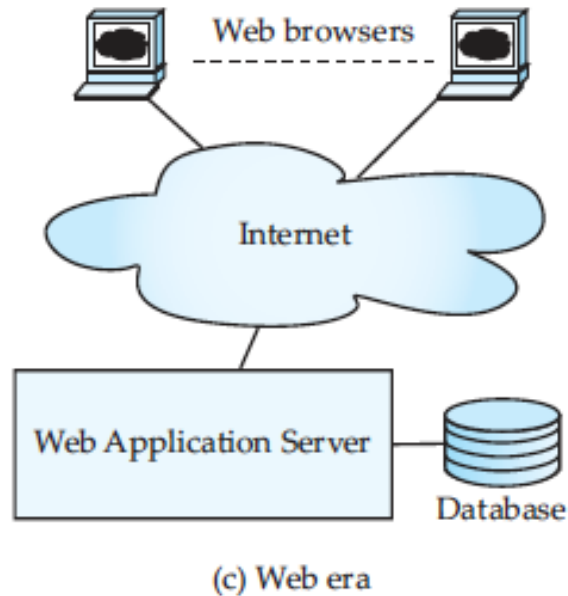
56. If the requested content still could not be located, only then would the request be routed to the content provider’s origin server. When this situation occurs, one or more of the involved servers would typically determine if some or all of the requested content should be stored in its cache.

1. CDN Server Databases

57. There are many examples of different types of CDN servers accessing local and/or remote databases in the prior art and these databases were well known to a POSITA. For example, a database system concepts textbook written years before the ’173 patent that was “intended for a first course in databases at the junior or senior undergraduate, or first-year graduate, level” provides many examples. EX-1011.

58. Using a database to store CDN content, policies and other metadata was greatly preferred over a basic file system to address at least the following problems: 1) data redundancy and inconsistency; 2) difficulty in accessing data; 3) data isolation; 4) data integrity problems; 5) atomicity (partial transaction) problems; 6) concurrent-access anomalies; and 7) security problems. *Id.*, 3-6.

59. A POSITA would have understood that the web application server architecture using a database as shown below was a standard configuration.



Id., 376

60. Additionally, Buyya describes three different methods an edge server can use to update its database.

a. Content Blind Caching

61. When Content Blind Caching is performed “edge servers cache the results of previous queries to the database. In such a way, the server may process locally future identical or similar queries, thus improving performance and relieving the load on the origin server back-end layer. The approach of caching query results to replicate the back-end layer is highly popular. For example, the GlobeCGC [40]

system explicitly aims to cache queries on the edge servers of a geographically distributed systems such as a CDN.” Ex-1010, 117.

b. Content Aware Caching

62. When Content Aware Caching is performed “each edge server runs its own database server, which contains a partial view of the centralized database. The typical approach for partial data replication is to push section of the database close to the edge, according to access patterns. . . . A significant example of replication mechanism is provided by GlobeDB [43], that uses partially replicated databases based on data partition to reduce update traffic.” *Id.*, p. 118.

c. Full Database Replication

63. Full database replication “maintains identical copies of the database at multiple locations. By moving a copy of the database to the edge servers and keeping the database copies coordinated among them, it becomes possible to completely deliver dynamic content at the edge of the network, without the need to modify each deployed application.” *Id.*, p. 119.

V. '173 PATENT

64. The '173 patent was originally filed as U.S. Application No. 15/588,935 on May 8, 2017. It claims priority as a continuation to another patent application (No. 14/570,512) filed on December 15, 2014—which issued as U.S. Patent No. 9,648,127.

65. I have therefore been asked to assume that the '173 patent is entitled to the earliest possible priority date of December 15, 2014 for this declaration and I have not performed any analysis or opinions regarding this date.

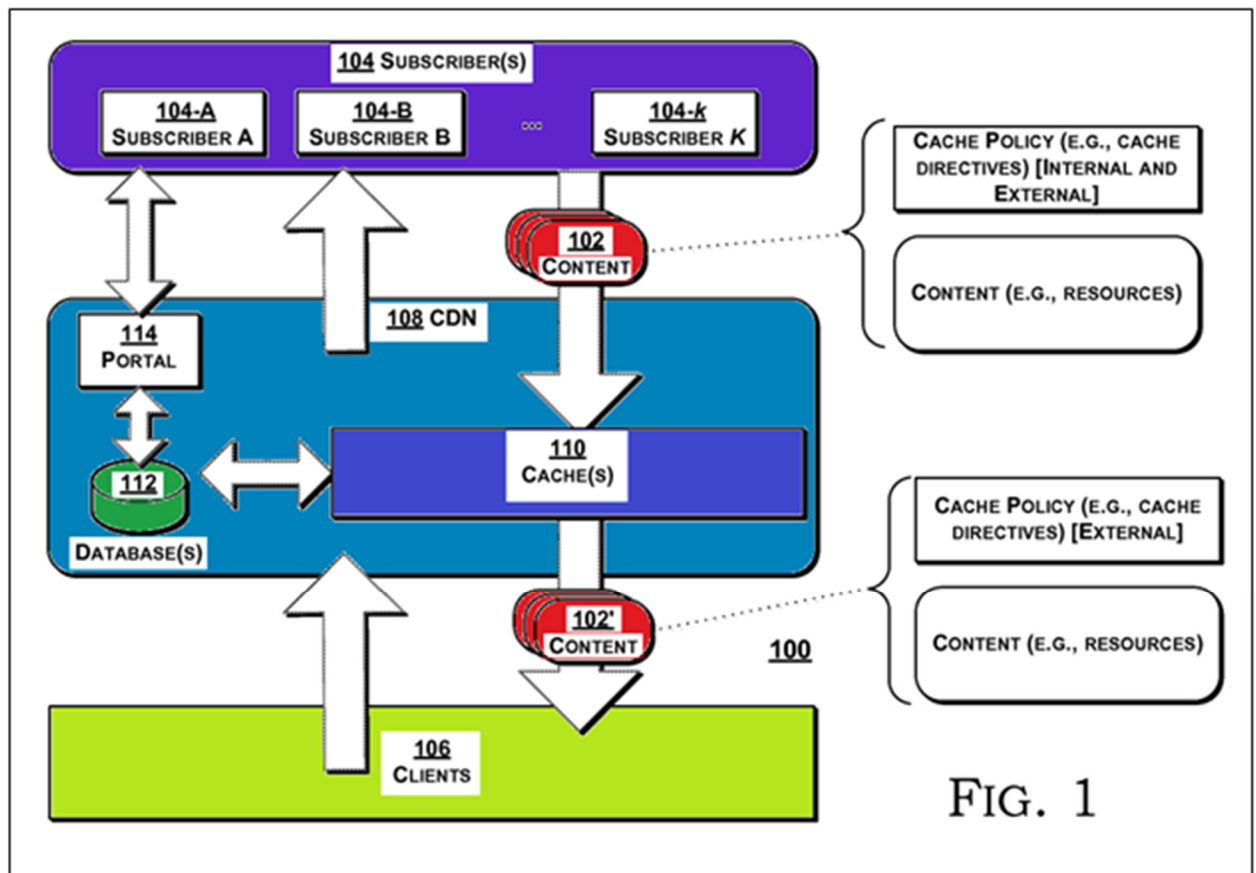
66. I have reviewed the '173 patent and its file history and provide the following summary.

A. Specification

67. The '173 patent is primarily concerned with CDNs, Ex-1001, 2:7-15, and specifically to caching content within a CDN, *id.*, 1:34-36.

68. The '173 patent explains that CDNs are used to distribute content associated with subscribers to clients. *Id.*, 2:21-25. CDNs contain caches, also referred to as “caching nodes” or simply “nodes,” that are used to store content within the CDN. *Id.*, 3:1-10. Caches 110 are also referred to as “caching nodes” or simply “nodes.” *Id.*, 4:53-61. In order to determine whether cached content has expired, CDNs use caching policies. *Id.*, 2:3, 3:11-22. Caching policies specify how long cached content remains valid before it is deemed expired (also known as “time-to-live” or “TTL”). *Id.* CDNs can also include data storage, such as, for example, a database, to store caching policy information. *Id.*, 4:28-37. The caches in the CDN can access this data storage to retrieve caching policy information associated with content. I have provided below an annotated version of Figure 1, which shows the architecture of a CDN, wherein content 102 (red) associated with subscribers 104

(purple) is distributed to clients 106 (light green) via CDN 108 (light blue), which caches content 102 in one or more caches 110 (dark blue). Caching policies are stored in data storage, such as a database 112 (dark green). The caches 110 can access the data storage to retrieve caching policy information associated with content 102.

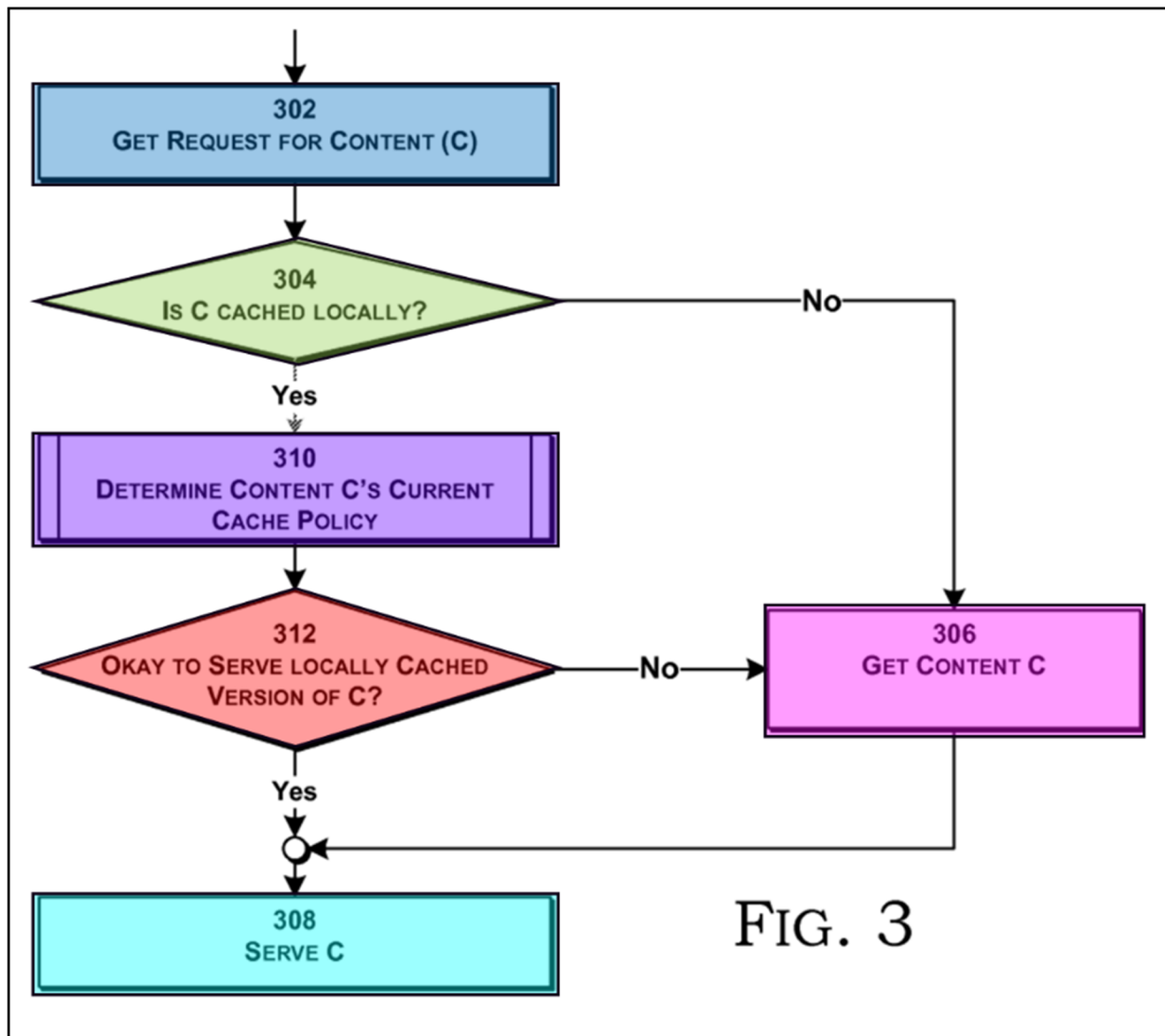


69. The '173 Patent further explains that when a cache receives a request for content, it checks for a locally-cached copy. *Id.*, 5:20-30. If the cache does not have a local copy, it retrieves the requested content from an appropriate node in the CDN, referred to as a “fill-authoritative node.” *Id.*, 5:28-34, Fig. 3. In contrast, if the

cache does have a local copy of the requested content, it obtains the current cache policy associated with the content, and then uses this policy to determine whether it can serve the copy that it has. *Id.*, 5:39-45. This can include calculating a time to live, or TTL, for the content. For example, if the cache policy specifies that the content may be cached for one hour, the node compares the fill time of the locally-cached copy to the specified policy (and the current time) to determine whether the cached copy of the content can be served or if it is expired and therefore needs to be refreshed or reloaded. *Id.* at 4:62-5:2; *see also id.* at 3:57-61, 5:55-63. If this process indicates that the content is still valid, the cache serves its locally-cached version of the content. On the other hand, if this process indicates that the content has expired, the cache obtains the current version of the content and then serves it to the requesting client.

70. I have provided below an annotated version of Figure 3, which shows this process. Specifically, as shown in annotated Figure 3, the cache receives a content request 302 (blue) and checks for a locally-cached copy 304 (green). If there is no local copy, the cache retrieves the content 306 (pink) and serves it 308 (turquoise). If there is a locally-cached copy of the content, the cache obtains the current cache policy associated with the content 310 (purple) and determines whether the content can be served based on the cache policy 312 (coral). *Id.*, 5:39-45. For example, if the cache policy specifies that the content may be cached for one

hour, the node compares the fill time of the locally-cached copy to the specified policy (and the current time) to determine whether the cached copy of the content can be served. If the content is valid, the cache serves the locally-cached content 308 (turquoise); if the content is invalid, the cache obtains 306 (pink) and serves 308 (turquoise) the current version of the content.



71. The alleged advantage of this approach is that the cache obtains the caching policy at the time content is requested, rather than when the content was originally cached, a concept the patent refers to as “late-binding” of the caching policy. *Id.*, 3:62-4:3. However, caches that determine caching policies at request time were known prior to the ’173 patent’s effective date, as I describe further below.

B. Prosecution History

72. I have reviewed the ’173 prosecution history and provide the following summary of relevant facts.

73. In prosecution, Applicant asserted that the “late-binding policy” reflected the purported point of novelty, distinguishing the Examiner’s applied prior art from the claims because in the prior art “the actions of [node] are not dependent on the current cache policy. Either way, the [node] will seek the newest version from the [origin server] (either the complete version or the delta).” ’173FH at 118-20. Applicant further argued that the prior art “fail[ed] entirely to identify what it considers the ‘current cache policy’” because the “no-cache directive would not be used to determine whether already stored content could be served pursuant to the content request... [and] does not comprise a time period for expiration of content in the cache.” *Id.*, at 121.

74. The O'Rourke reference was cited several times during prosecution of the application that would mature into the '173 Patent. Specifically, O'Rourke was presented to the Examiner on April 9, 2020, and May 9, 2017 (and considered by the Examiner on May 11, 2020, and August 14, 2017). *See* '173FH at 53, 304, 45, 272, respectively. In addition, the Examiner was made aware of (and considered) the Written Opinion of the International Searching Authority from the '173 Patent's PCT counterpart, in which the International Searching Authority found that all pending claims lacked novelty under PCT Article 33(2) as being anticipated by O'Rourke. *Id.* at 277-284. Yet despite these disclosures, the Examiner never applied O'Rourke as a basis for rejection.

75. Middleton was filed on July 14, 2010 and claims priority to a provisional filed on Sept. 10, 2009. *See* Middleton. It was published on March 10, 2011, several years before the '173 priority application was filed in 2014. *See* Middleton. Middleton is assigned to Level 3 Communications, LLC—the same assignee as the '173 patent. Middleton; '173 Patent. Yet Level 3 Communications never disclosed Middleton to the PTO at any time during the prosecution of the '173 Patent. *See* 173FH.

VI. PRIOR ART

A. O'Rourke (U.S. Patent No. 7,912,921)

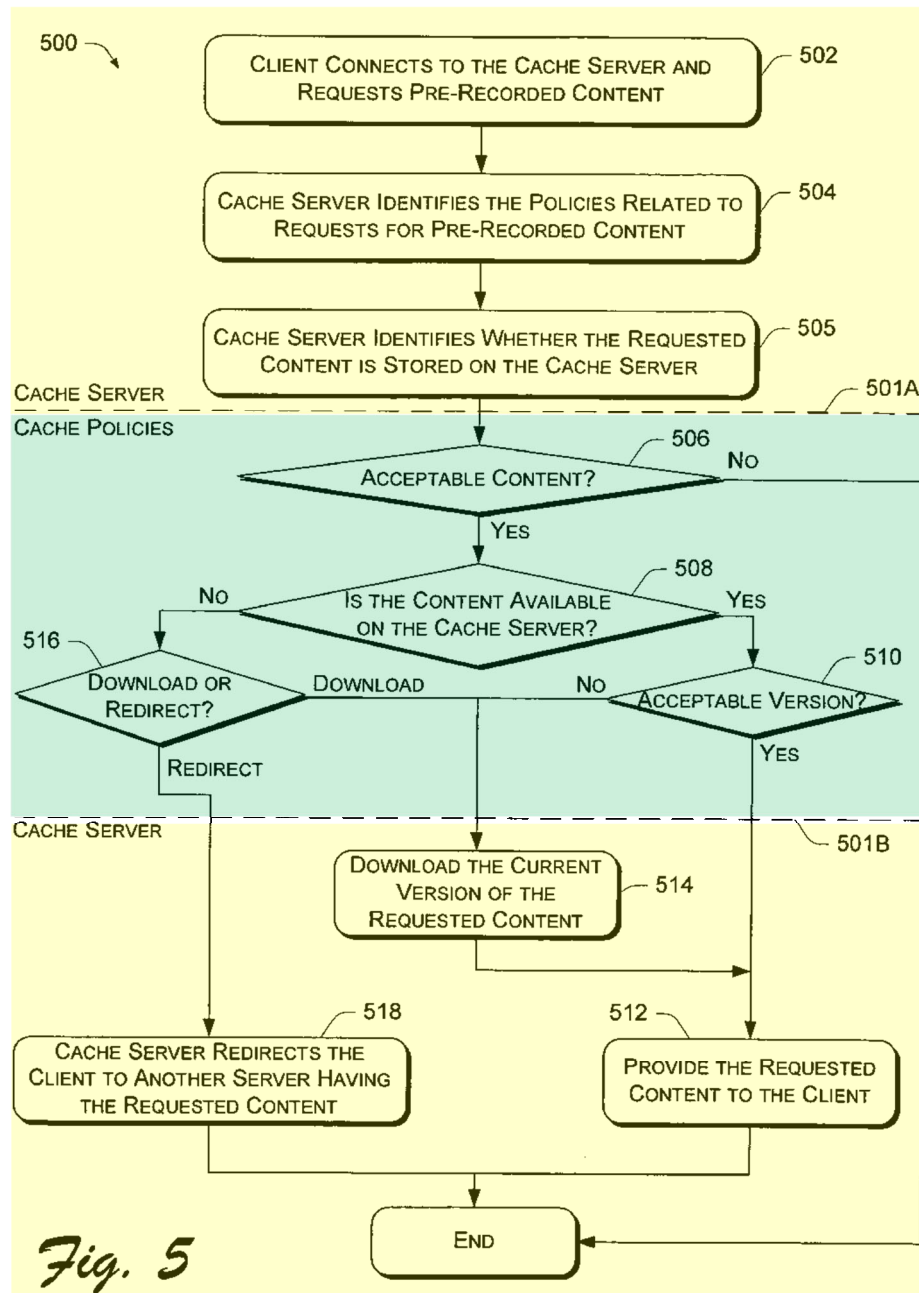
76. O'Rourke is directed to a "cache server [that] includes a media serving engine that is capable of distributing media content" across a network. O'Rourke, Abstract. O'Rourke noted that existing caches servers were "difficult to customize" and provided limited "opportunities for a customer to modify the cache server to meet their specific needs." *Id.*, 1:19-21, 2:8-9, 2:19-26. To address this problem, O'Rourke teaches a cache server that can be configured using "a set of cache policies that are determined by the user of the cache server." *Id.*, 3:26-29; *see also id.*, 2:32-35.

77. O'Rourke teaches a caches server that includes a cache engine, which performs various caching functions, such as "downloading media content from an origin server, distributing media content to one or more clients, redirecting clients to a different server for particular media content, and deleting cached media content to release storage space for caching different media content." *Id.*, 4:43-50, Fig. 4. The cache server also includes a set of cache policies. *Id.*, 4:50-52, Fig. 4. The cache policies can be defined by an administrator, and are used by the cache server's cache engine to perform various caching functions. *Id.*, 4:50-52, 7:41-44. For example, the cache policies may define whether the cache engine should download and cache content, or whether the cache engine should delete cached content. *Id.*, 4:52-56.

78. O'Rourke explains how the cache policies can be changed during operation. Specifically, O'Rourke explains that the cache policies (also referred to as policy plug-ins) can be added or removed at any time, "thereby providing flexible management of the cache policies. Thus, the operation of a cache server can be modified simply by changing the cache plug-in modules in the cache server." *Id.*, 3:42-45, 5:2-8. This description in O'Rourke explains that the cache policies, which dictate how the cache engine operates, can be changed during operation of the cache server.

79. In Figure 5, O'Rourke illustrates "a procedure 500 for handling a request to receive pre-recorded content." *Id.*, 6:1-2. O'Rourke explains that the procedure 500 is split into "the functions performed by the cache server as well as the decisions made by the cache policies." *Id.*, 6:2-5. O'Rourke illustrates this split as a "pair of broken lines 501A and 501B [which] identify the sections of procedure 500 that represent functions performed by the cache server (the sections above line 501A and below line 501B) and the sections of procedure 500 that represent decisions made by the cache policies (the section between lines 501A and 501B)." *Id.*, 6:5-10. This description in O'Rourke illustrates that the functions performed by the cache server (e.g., the cache engine) are based on decisions made based on the cache policies. And because the cache policies can be changed during operation, the actions of the cache server (e.g., whether to serve cached content) thus depend on

the changing cache policies. I have provided below an annotated version of Figure 5 of O'Rourke, which illustrates the functions performed by the cache server (yellow) and the decisions made based on the cache policies (green). O'Rourke, 6:5-10.



80. As shown in Figure 5 of O'Rourke, the first section of functions performed by the cache server (i.e., the section above line 501A) are those that identify information used in making the cache policy decisions (e.g., identifying the policies related to requests for pre-recorded content, identifying whether the requested content is stored on the cache server), while the second section of functions performed by the cache server (i.e., the section below line 501B) are those that perform operations based on the cache policy decisions (e.g., download the requested content, provided the requested content, etc.). It would have been obvious to a POSITA to perform the "information identifying functions" (i.e., above line 501A) in a different order without altering the functionality of the procedure 500. For example, it would have been obvious to a POSITA to, in response to a request for pre-recorded content, identify whether the requested content is stored on the cache server (block 505) and then identify the policies related to requests for pre-recorded content (block 504), since neither function depends on the other function. O'Rourke's disclosure also suggests this reordering, explaining that "[a]lthough the description above uses language that is specific to structural features and/or methodological acts...the specific features and acts are disclosed as exemplary forms of implementing the invention." O'Rourke, 10:40-45. Moreover, identifying whether content was in a cache (in response to a request for the content) prior to identifying cache policies associated with the request or content, was known to a

POSITA. *See, e.g.,* Wein, 20:26-21:8, Fig. 5 (checking for a cache hit before applying policies found in metadata or another file); Akamai EdgeSuite, 6 (edge server checking its internal cache to determine if a page has been requested before). In other words, it would have been obvious, in response to a request for content (block 502), to identify whether the requested content is stored in the cache (block 505) and then identify policies related to requests for pre-recorded content (block 504).

B. Middleton (U.S. Pub. No. 2011/0060812)

81. Middleton discloses “an extensible content delivery platform” that “provide[s] a logical flow of actions [for]...the content delivery process of the content delivery network.” Middleton, Abstract. As Middleton explains, in the CDNs of the time, cache servers were relatively complex. Middleton, [0005], [0006]. These cache servers were “feature rich,” but were difficult to configure and generally required significant code changes to add new features. *Id.*, [0006].

82. To address these perceived problems, Middleton teaches an “extensible content delivery platform” in which “[r]ich and well-formed metadata and ‘metacode’ can be associated with content and/or requests in a consistent and programmatic manner.” *Id.*, [0007]. Middleton explained that this extensible content delivery platform “provides for flexible implementation (e.g., enriched extensibility, customization, and application integration potential) without core code changes,”

provided by an edge server with an “event-based application programming interface (API).” *Id.*, [0010], [0035].

83. In Middleton’s platform, the edge server is part of a CDN and “is configured to provide requested content to a requester” (which could be, for example, an end user computer or intermediate device). *Id.*, [0031], [0032]. When the edge server receives a request for content, it “determine[s] whether the requested content is available locally...to be provided to the requester.” *Id.*, [0033]. For example, the edge server determines whether the content is stored locally in cache and is not stale. *Id.* If the content is stored locally and not stale, the edge server provides the requested content to the requester. *Id.*

84. On the other hand, if the edge server determines that the content is stored locally but is stale, the edge server attempts to revalidate the content. *Id.*, [0034]. This is done by sending a request to a media access server (MAS). *Id.* If the content is revalidated, the MAS sends the edge server a new time-to-live (TTL) value for the content, and the edge server then updates its TTL value for the content. *Id.*, [0057], [0059]. In other words, based on the MAS response, the edge server re-characterizes locally stored content (that had been considered stale) using the new TTL value.

85. If, instead of revalidating the content, the MAS indicates that the content is not revalidated, the edge server deletes the stale content from the cache.

Middleton, [0059]. In other words, the ultimate determination of whether locally-stored content is truly stale depends on the results of a revalidation request made by the edge server to the MAS.

86. I have provided below Figure 4 of Middleton, which illustrates the states of content in the described system. Middleton, [0069]. As shown in Figure 4, content is stored in the cache and not stale (state C, 406) until the content's age exceeds the content's specified TTL value. *Id.*, [0076]. When the content's age exceeds the content's specified TTL value, the content transitions to being characterized as stale (state D, 408). *Id.* However, the content may be recharacterized as not stale if it is successfully revalidated. *Id.*, [0077]-[0078].

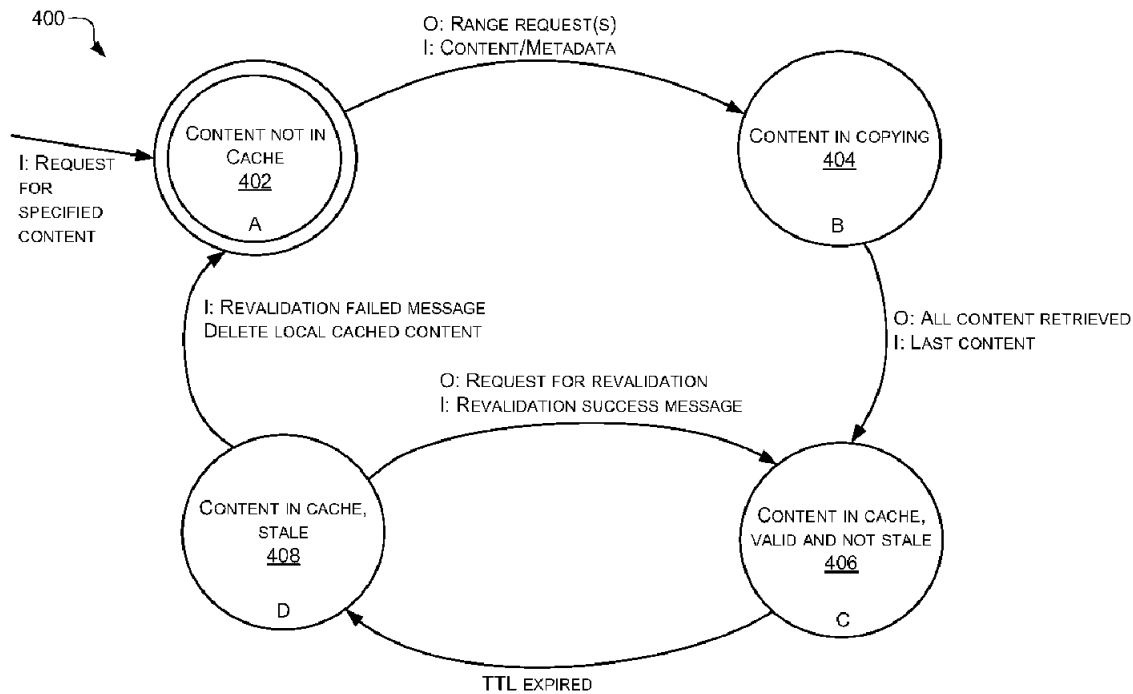


Fig. 4

C. Blumofe (U.S. Pub. No. 2012/0096106)

87. Blumofe teaches methods for “extend[ing] [a CDN] into one or more mobile or wireline networks that cannot or do not otherwise support fully-managed CDN edge servers.” Blumofe, Abstract. Blumofe explained that that “[b]ecause of the way that fixed or wireless mobile networks are architected and operated, it is not always possible or practical to deploy fully-managed CDN servers therein (i.e., close to end-users).” *Id.*, [0007]. For example, in 3G wireless mobile networks, the gateways used to connect wireless end-users to the Internet, at which CDN servers could be deployed, were often thousands of miles and hundreds of milliseconds away from the end-users. *Id.*

88. Blumofe describes a CDN Extender that can be used to address these issues by caching CDN-provisioned content closer to end-users. Blumofe, [0025]. Blumofe describes that the Extender may be implemented in wireless mobile networks or other network environments that do not support CDN services, and that in these environments, the Extender operated as a cache beyond the edge of the CDN. *Id.*, [0025], [0027]. I have provided below an annotated version of Figure 3 of Blumofe, which illustrates how the Extender 300 (blue) is implemented in a network infrastructure (green) outside the CDN (red).

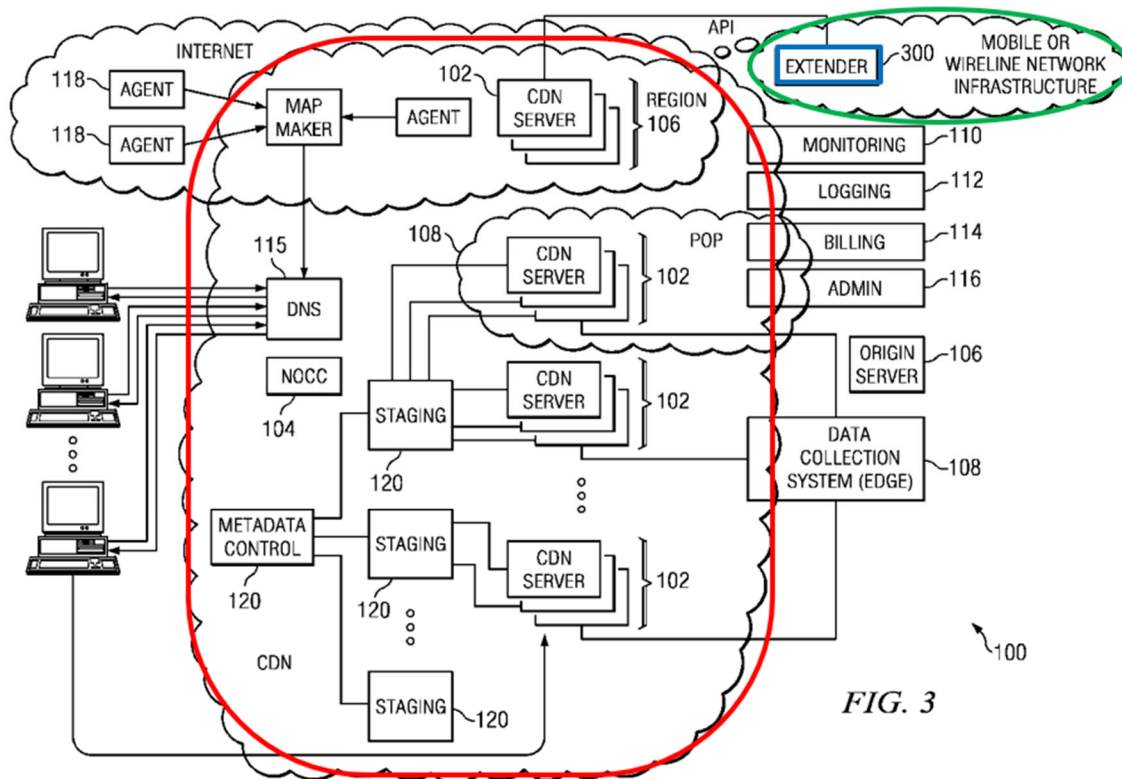


FIG. 3

89. Blumofe explains that when the Extender receives an end-user request for content, it determines whether the requested content is already in the Extender's cache. Blumofe, [0033]. If the content is not its cache, the Extender sends an HTTP GET request to the CDN to fetch the content, then responds to the end-user. *Id.* The CDN's response to the Extender's request may also include cache control information, such as expiration time. *Id.*, [0037].

D. [MS-MSSO]: Microsoft - Media Streaming Server System Overview

90. MS-MSSO, published by Microsoft, "provides an overview of the Media Streaming Server System Overview Protocol Family." MS-MSSO, 2. MS-MSSO "provides an overview of the Windows Media protocols that are implemented in the Windows Server operating systems and that are used to interoperate or communicate with Windows client operating systems." MS-MSSO, 2. These protocols are used "streaming digital media to and from a media server." MS-MSSO, 2.

91. MS-MSSO includes a "System Summary," which describes a "Media Streaming Server (MSS) System [that] is designed to deliver an end-to-end experience for components involved in the creation, distribution, and playback of audio and video content." MS-MSSO, 10. The MSS System components include "a computer running a media encoder, a server running as a media server, and a number

of client computers running media play clients. The encoder converts both live and prerecorded audio and video content to a media format. The server then distributes the content over a network or the Internet. The media player client then receives the content.” MS-MSSO, 10. MS-MSSO explains that the MSS System may additionally include distribution servers “to scale and meet network demands.” MS-MSSO, 10.

92. The MSS System components use “member protocols to enable scenarios ranging from live broadcast playback to on-demand playback.” MS-MSSO, 10. The member protocols include the Media Stream Broadcast (MSB) Protocol, Microsoft Media Server (MMS) Protocol, and Windows Media HTTP Streaming Protocol. MS-MSSO, 10. MS-MSSO characterizes these member protocols as “Windows Media Protocols,” and specifies which versions of Microsoft Windows Server operating systems are required to implement the different member protocols on a given server system. MS-MSSO, 10, 36. The member protocols may thus be used by media servers running a Windows Server operating system (i.e., a Windows Media Server).

VII. LEVEL OF ORDINARY SKILL

93. I understand that when both interpreting the claims and assessing the prior art, I must do so from the perspective of a person of ordinary skill in the art

(“POSITA”) as of the relevant priority date. My understanding is that the earliest possible priority date of the ’173 patent is December 15, 2014.

94. In my opinion, in the 2014 timeframe, a POSITA would have had at least a bachelor’s degree in computer science, electrical engineering, or a related field, and at least two years of work or research experience in the field of content-delivery management systems or networks. However, work experience can substitute for formal education and vice versa.

95. I was at least a POSITA as of the earliest possible priority date of the ’173 patent, and still am.

VIII. CLAIM CONSTRUCTION

96. As I noted above, I understand that unless claim terms are provided an express construction, the terms must be given their plain and ordinary meaning.

97. I understand that Petitioner is not advancing any constructions at this time, and I do not consider any claim constructions necessary for my purposes (to compare the plain and ordinary meaning of the claims to the prior art to evaluate whether the prior art would have rendered the claimed invention obvious). I have therefore applied the plain and ordinary meaning of each term in the claims throughout this declaration.

IX. GROUNDS

A. Ground 1A: Obviousness over O'Rourke

1. Claims 1, 8

a. 1[pre]: A computer-implemented method, operable with a content delivery network (CDN), the method comprising:

98. In my opinion, if the preamble is limiting, O'Rourke discloses or suggests it.

99. O'Rourke discloses “*methods...that perform[] a variety of cache-related functions.*”² O'Rourke, 3:26-27. O'Rourke's disclosed *methods* may be performed by a cache server, such as a Windows Media Server. O'Rourke, 4:57-60, 6:1-5 (“The procedure 500 illustrates the functions performed by the cache server”). The cache server of O'Rourke is a type of *computer*, and because the *methods* of O'Rourke are performed on this *computer*, O'Rourke's cache-related functions are *computer-implemented methods*.

100. The cache-related functions described in O'Rourke's include the cache server “distributing media content to one or more clients” over a communication network, such as a high-speed LAN. O'Rourke, 3:36-38, 4:45-50. Within the cache server, the cache engine “performs the various functions necessary to cache media content,” such as “downloading media content from an origin server, distributing

² Italicized text is claim text unless otherwise noted.

media content to one or more clients, redirecting clients to a different server for particular media content, and deleting cached media content to release storage space for caching different media content.” *Id.*, 4:42-50. In other words, the cache servers within the network are adapted to perform these functions relating to downloading media content and distributing it to clients, and the communications network is used by the cache servers for content delivery. O’Rourke’s communication network is therefore a *content delivery network*, and O’Rourke’s cache-related functions are thus *operable with a content delivery network (CDN)*.

- b. 8[pre]: A computer program product having computer readable instructions stored on non-transitory computer readable media, the computer readable instructions including instructions for implementing a computer-implemented method, said method operable on a device comprising hardware including memory and at least one processor and running a service on said hardware, said method operable in a content delivery service (CDN), said method comprising:**

101. In my opinion, if the preamble is limiting, O’Rourke discloses or suggests it.

102. O’Rourke describes “a computer 642 that can be used in accordance with the invention” and can perform the functions of a cache server. O’Rourke, 8:45-49. The computer includes “one or more processors or processing units 644, a system memory 646,” and drives (e.g., hard disk drive 656). O’Rourke, 8:49-52, 8:62-9:4. O’Rourke explains that “[t]he drives and their associated computer-readable media

provide nonvolatile storage of computer readable instructions, data structures, program modules and other data for computer 642.” O’Rourke, 9:4-7. The computer readable instructions and program modules on hard disk 656 are *[a] computer program product having computer readable instructions stored on non-transitory computer readable media, the computer readable instructions including instructions for implementing a computer-implemented method.*

103. O’Rourke’s describes that procedure 500 may be performed by the computer 642. O’Rourke, 6:2-5 (“The procedure 500 illustrates the functions performed by the cache server”). Because the program modules are performed on computer 642 (which may be a cache server), the method is *operable on a device comprising hardware including memory and at least one processor and running a service on said hardware, said method operable in a content delivery service (CDN).*

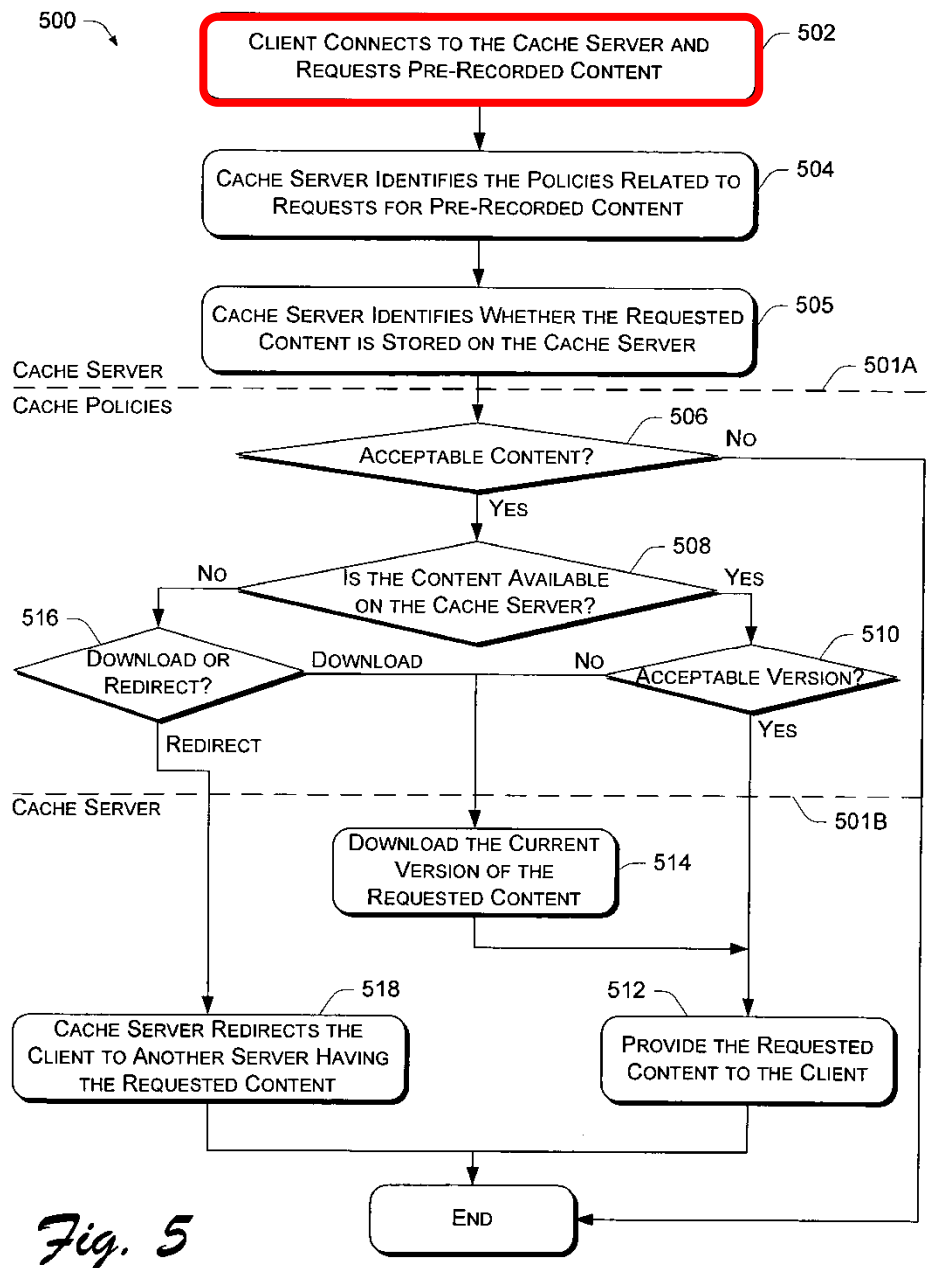
- c. 1[a]/8[a]: (A) in response to a request for content, determining, by a node [in]/[of] the CDN, if the content is cached locally at the node;**

104. In my opinion, O’Rourke discloses or suggests these limitations.

105. As described above, O’Rourke discloses “a procedure 500 for handling a client request to receive pre-recorded content,” performed by a cache server based on cache policies. O’Rourke, 6:1-5. The cache server “is able to redistribute...media content [from an origin server] to multiple local clients.” O’Rourke, 3:46-55. The

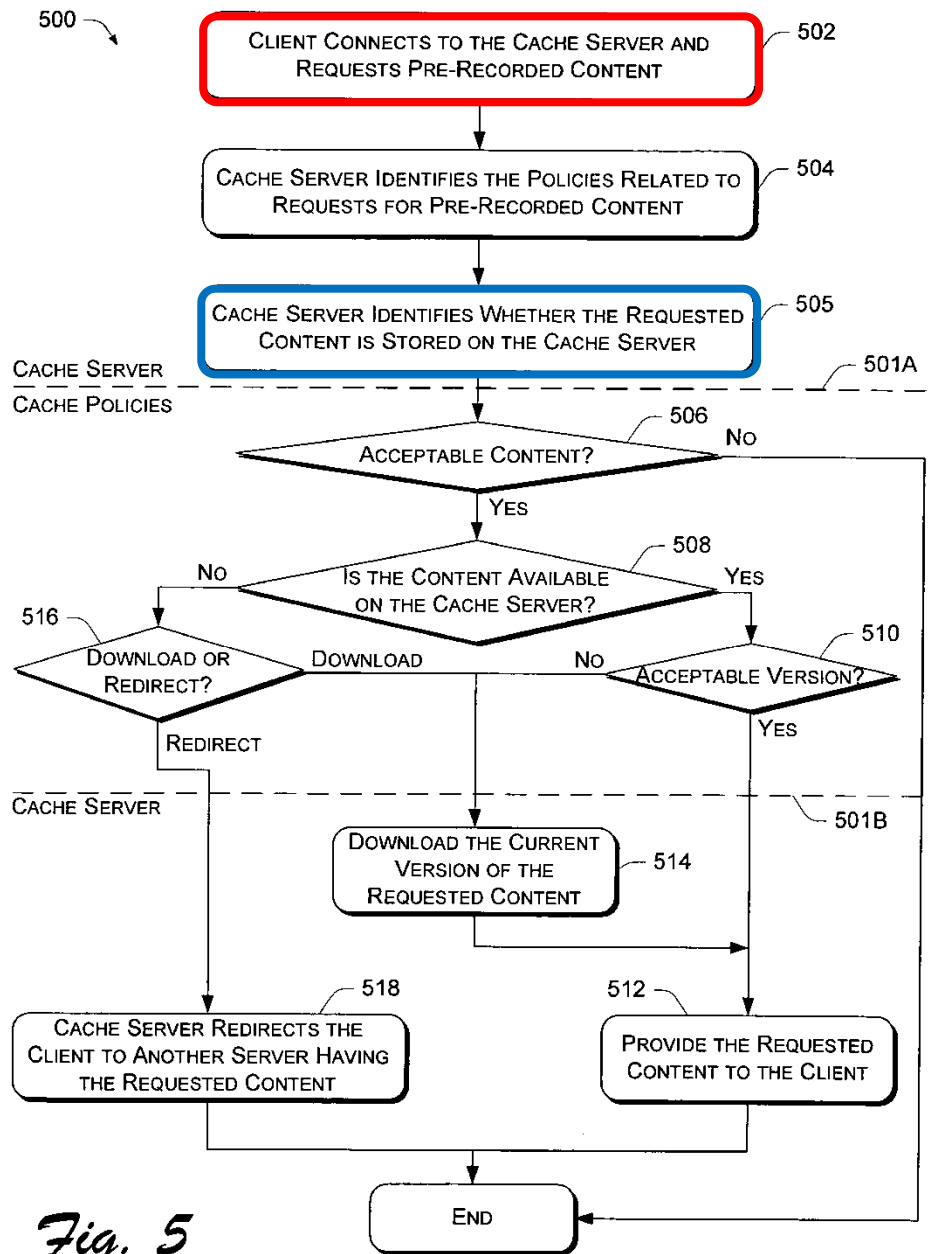
cache server (*node*) is thus in a *CDN* or of a *CDN*, and O'Rourke's procedure 500 is performed *by a node [in]/[of] the CDN*.

106. The procedure 500 described in O'Rourke is "for handling a client request to receive pre-recorded content." O'Rourke, 6:1-2. I have provided below an annotated version of Figure 5 of O'Rourke, which illustrates that the first block of procedure 500 is block 502 (in red), where the client connects to the cache server to request the content. O'Rourke, 6:1-12. The blocks of procedure 500 that follow block 502 are thus *in response to a request for content*.



107. During procedure 500, “[t]he cache server then identifies whether the requested content is stored on the cache server” (*determining...if the content is cached locally at the node*). O’Rourke, 6:14-15. As illustrated in Figure 5 of O’Rourke, the determining of whether the requested content is stored on the cache

server is performed at block 505 (blue), after block 502 (red), and is thus *in response* to the client request for content. O'Rourke, 6:10-12, 6:14-15.



108. Additionally, as I describe above, it would have been obvious in response to a request for pre-recorded content (block 502) for O'Rourke's procedure

500 to identify whether the requested content is stored on the cache server (block 505) and then identify cache policies related to pre-recorded content (block 504). In other words, it would have been obvious after the cache server receives the request for pre-recorded content (block 502) (*in response to a request for content*) for the cache server to then identify whether the requested content is stored on the cache server (block 505) (*determining, by a node [in]/[of] the CDN, if the content is cached locally at the node*). O'Rourke discloses or suggests this limitation for this additional reason.

d. 1[b]/8[b]: (B) when it is determined [by the node]/[] that said content is cached locally, then, after determining that the content is cached locally:

109. In my opinion, O'Rourke discloses or suggests these limitations.

110. As I describe above IX.A.1.b, O'Rourke teaches "identif[ying] whether the requested content is stored on the cache server (block 505)." O'Rourke, 6:14-19. "For example, the cache server may examine a URL associated with the requested content to determine whether the content is available on the cache server." *Id.* In addition, O'Rourke describes how "the cache policies determine whether the requested content is available on the cache server (block 508)." *Id.*, 6:28-31. As described in greater detail below, O'Rourke further describes different functions performed by procedure 500 depending on the outcome of the determination at block 508 (i.e., if the content is available on the cache server, determining whether it is an

acceptable version; if the content is not available on the cache server, determining whether to download it). O'Rourke, 6:31-34, 6:55-58. In other words, O'Rourke describes identifying whether requested content is available on the cache server (block 505), and then responding to the client request differently depending on whether the content is available on the cache server (block 508). O'Rourke thus "determine[s] whether the cached version of the requested content is an acceptable version (block 510)" *when it is determined [...] that said content is cached locally.*

111. O'Rourke describes that this determination is made by the cache server (*node*). O'Rourke explains that "[i]n one embodiment, the cache server is a Windows Media Server and the set of cache policies are contained in one or more cache plug-in modules that are loaded in the Windows Media Server and called by the Windows Media Server...Alternatively, the cache plug-in modules can run as part of another process on the same computer system." O'Rourke, 4:57-5:1. Similarly, Figure 4 of O'Rourke (reproduced below) illustrates that the "[c]ache server 300 includes a cache engine 302 which performs the various functions necessary to cache media content," and a "set of cache policies 304 are accessed and utilized by the cache engine 302 in performing its various caching functions." O'Rourke, 4:42-52. Since the cache policies may be loaded in and called by the cache server, the decisions made based on the cache policies (further described herein) are also performed by the cache server (*node*). Thus, when "the cache

policies determine whether the requested content is available on the cache server,” that determination is made *by the node* as recited in 1[b]. O’Rourke,6:28-31.

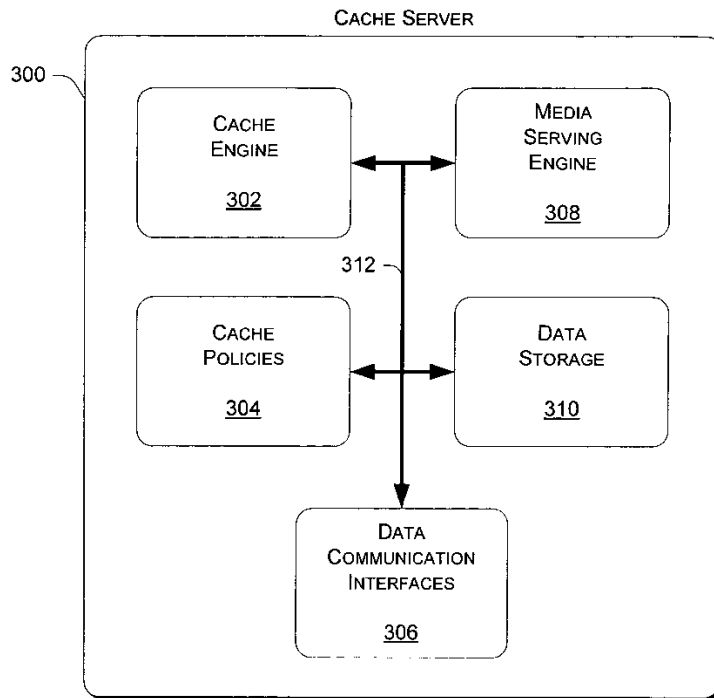
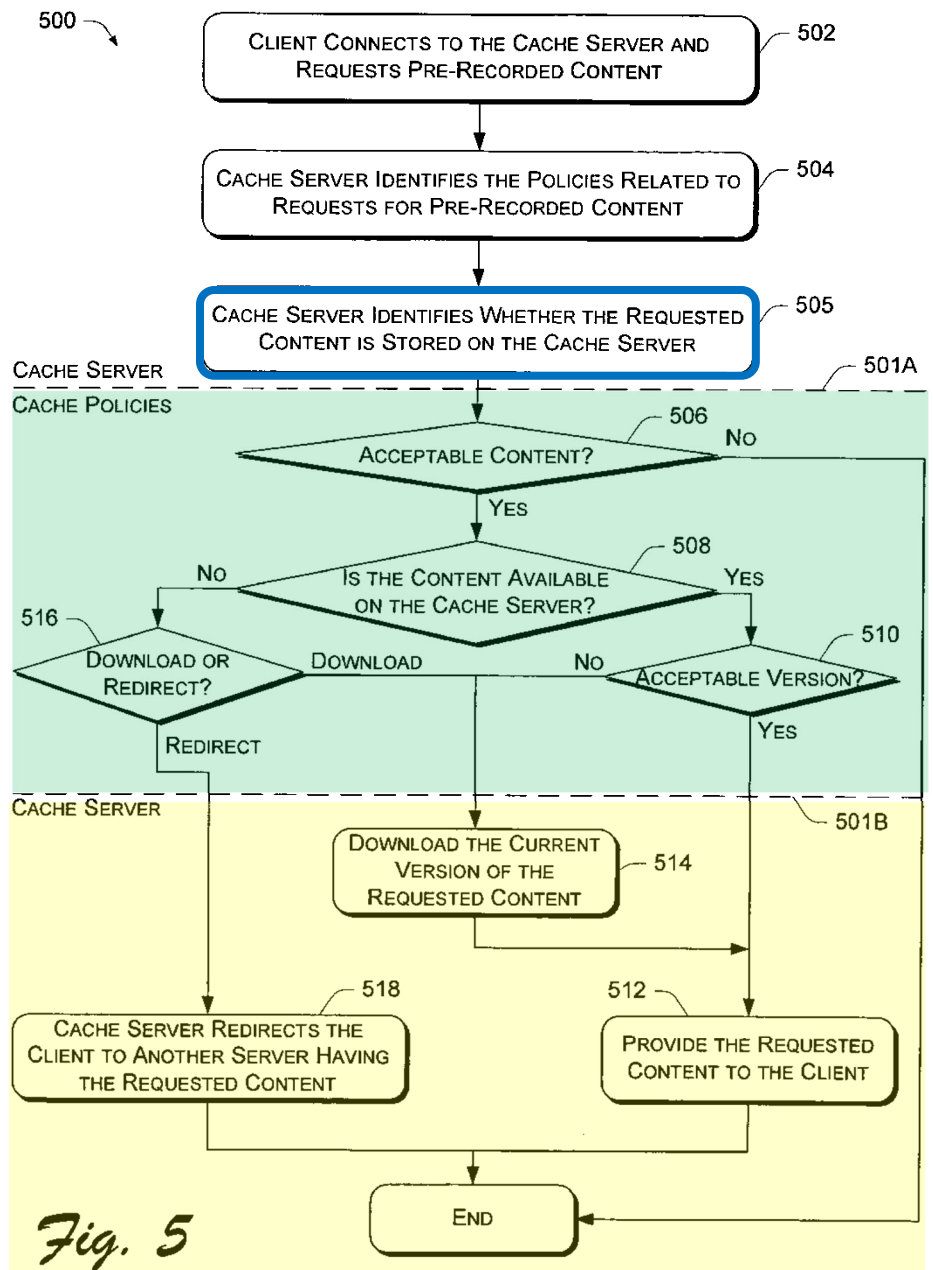


Fig. 4

112. As described above, O’Rourke’s procedure 500 includes “functions performed by the cache server as well as the decisions made by the cache policies.” O’Rourke, 6:2-5. I have provided an annotated copy of Figure 1 of O’Rourke below, showing the section below line 501B (yellow) which represents functions performed by the cache server, and the section between lines 501A and 501B (green) which represents decisions made by the cache policies (which, as described above, may

also be performed by the cache server). O'Rourke, 6:5-10; Both highlighted sections of procedure 500 are performed after the cache server (*node*) "identifies whether the requested content is stored on the cache server" at block 505 (blue). O'Rourke, 6:14-15. Thus, the blocks following block 505 are performed *after determining that the content is cached locally*.



113. Furthermore, as I describe above, it would have been obvious in response to a request for pre-recorded content (block 502) for O'Rourke's procedure 500 to identify whether the requested content is stored on the cache server (block 505) and then identify cache policies related to pre-recorded content (block 504).

Thus, in addition to the blocks below line 501A being performed *after determining that the content is cached locally* (as described above), it would have been obvious (as described further below) for the identification of cache policies related to pre-recorded content (block 504) to have also been performed *after determining that the content is cached locally* at block 505.

- e. **1[b][i]/8[b][i]: (B)(1) determining, by the node, a current cache policy associated with the content, wherein determining the current cache policy comprises determining a time period for expiration of content in the cache; and then**

114. In my opinion, O'Rourke discloses or suggests these limitations.

115. As described above, O'Rourke teaches that cache policy decisions may be made by the cache server (*node*). Thus, cache policy decisions are *determin[ed]*, *by the node*.

116. O'Rourke teaches that after determining the requested content is stored on the cache server, "the cache policies determine whether the cached version of the requested content is an acceptable version." O'Rourke, 6:31-34. The cache policies may be "based on [an] administrator's knowledge and/or expectations with respect to, for example, the type of content requests received by the cache server." *Id.*, 7:48-51. As one example, "[t]hese policies may also determine whether the cache server is required to have the latest version of the content or whether a 'stale' version of the content is sufficient in particular situations." *Id.*, 7:64-67. Thus, the cache policies,

such as the *cache policy* that determines whether the cached version of the requested content is an acceptable version, are *associated with the content*.

117. O'Rourke teaches that the cache policy can determine whether the content stored on the cache server is an "acceptable version" based on "whether the cached version is the same as the origin server's version or is within thirty minutes of the origin server's version." O'Rourke, 6:34-37. Similarly, O'Rourke teaches that the cache policies may determine "whether a 'stale' version of the content is sufficient in particular situations." *Id.*, 7:64-67. In other words, per the cache policy, stored content may not be an "acceptable version" if it is too stale (e.g., stored on the cache server for more than thirty minutes). And to determine whether content is too stale, the cache policy must be able to determine when the change from acceptably stale (e.g., within thirty minutes) to too stale (e.g., more than thirty minutes) occurs. It would have been obvious to a POSITA that this change, from acceptably stale to too stale, is a *time period for expiration* (e.g., thirty minutes after the content was stored on the cache server). Thus, the *cache policy*, for determining whether the content stored on the cache server is an acceptable version, *comprises determining a time period for expiration of content in the cache*.

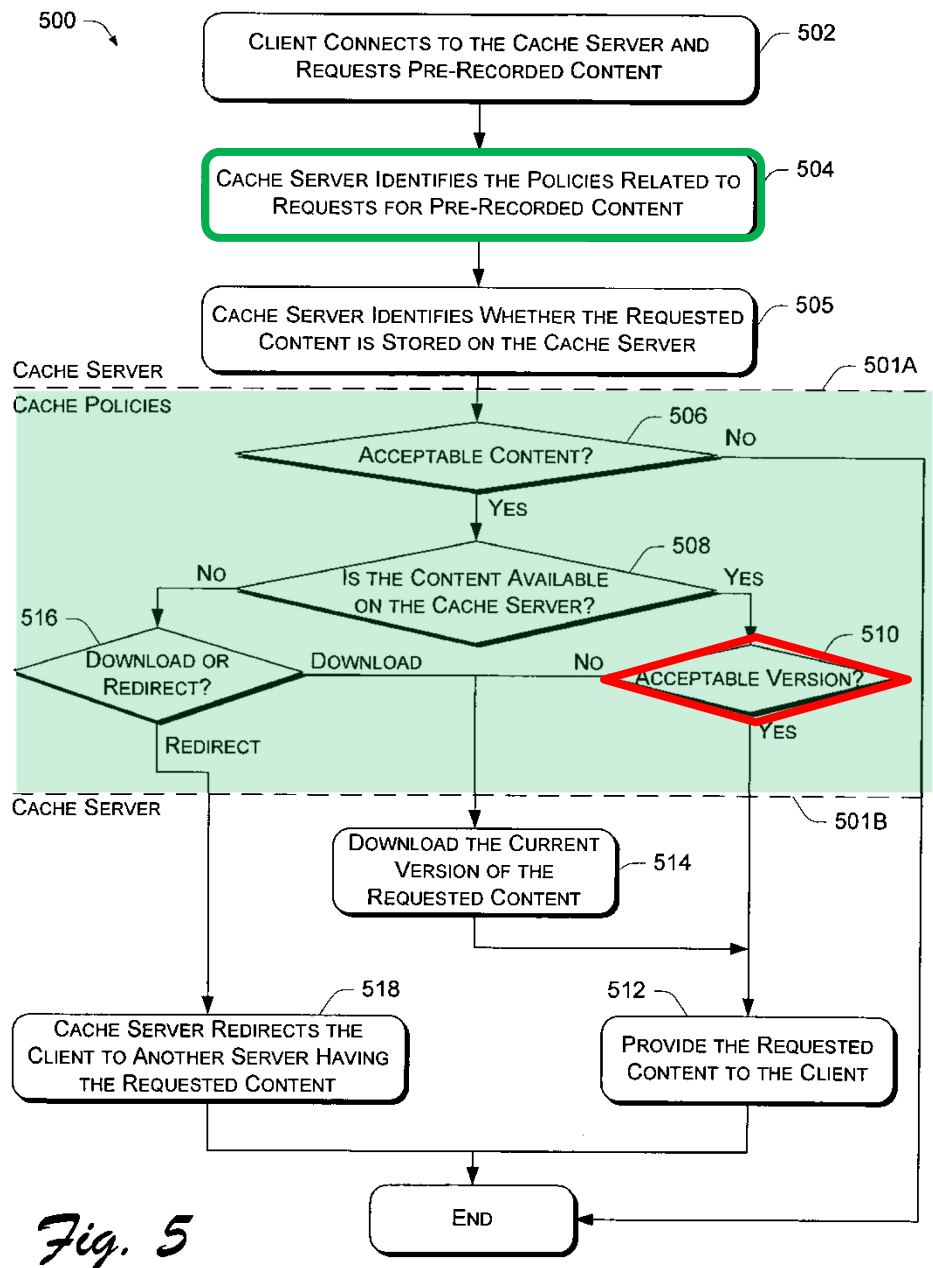
118. In addition, O'Rourke's teaching includes *determining the current cache policy*. By way of background, I note that the Specification of the '173 Patent describes systems and methods for accomplishing the claimed step of

determining...a current cache policy as follows. The '173 Patent describes a system that “allows for specification, e.g., within a CDN node, of a cache expiration policy of content the CDN caches on behalf of its subscribers.” EX-1001, 3:42-44. The '173 Patent describes how “a CDN subscriber...use[s] a portal to set a caching policy for a particular kind of their content (e.g., for their PDF files or the like), e.g., so that those files are cached for a particular time period (e.g., a week, an hour, etc.).” *Id.*, 3:57-61; *see also id.*, 5:38-41 (CDN supports “a subscriber or other user or entity setting or overriding an internal caching policy for resources, e.g., via a self-service portal or the like (e.g., via an API).”). The '173 patent further describes that “the CDN’s internal request processing process may determine the caching policy of requested content at request time rather than fill time,” a concept the Specification refers to as “late-binding.” *Id.*, 3:64-67; *see also id.*, 4:30-34. This so-called “late-binding” “assure[s] that the latest internal caching policy is used.” *Id.*, 3:64-4:3. The policy retrieved at request time is referred to as “the current cache policy.” *Id.*, 5:39-42; The '173 Patent describes the purported “benefit” of such a system as “having relatively instant policy updates” but notes the “downside of this approach is that the caching policy determination must be made for every request.” *Id.*, 6:4-9. In other words, the '173 Patent teaches a system that allows subscribers to set or update the cache policies while the CDN is in use, and teaches that in such a system, the claimed

step of *determining...a current cache policy* can be accomplished by determining the caching policy at request time to ensure that the latest policy is used.

119. O'Rourke operates in a similar manner. Specifically, O'Rourke describes how "the operation of [the] cache server can be modified simply by changing" the cache policies of the cache server. O'Rourke, 5:1-8. That is, the cache policies (such as the *cache policy* for determining whether stored content is acceptable) may change during operation of the cache server. O'Rourke characterizes the operative policy at any point in time as "the currently active cache policy." O'Rourke, 12:15-19 (claim 9: "An apparatus as recited in claim 6, further comprising a data communication interface coupled to the cache engine and the media serving engine, wherein the data communication interface is to transmit instructions to the server to change the currently active cache policy."). O'Rourke also describes that after the request for content is received at the cache server, "[t]he cache server identifies the policies related to requests for pre-recorded content (block 504)." O'Rourke, 6:12-14. I have provided an annotated copy of Figure 5 of O'Rourke below, showing that the identified cache policies (block 504, green) are then used to make the cache policy decisions (the section between lines 501A and 501B, shaded green), including the decision of whether the stored content is an acceptable version (block 510, red). O'Rourke teaches determining the cache policy of requested content at request time rather than fill time, i.e., the so-called "late-

binding” of the ’173 Patent. The cache policies (e.g., identified at block 504) used to make the cache policy decisions (e.g., a determination of whether stored content is an acceptable version at block 510) are thus each *a current cache policy*, and determining the cache policy in response to a request is *determining...a current cache policy*.



120. O'Rourke describes Figure 5 by explaining that its "procedure 500 for handling a client request to receive pre-recorded content" consists of functions performed by the cache server (i.e., the sections above line 501A and below line

501B) and the decisions made based on the cache policies (i.e., the section between lines 501A and 501B). O'Rourke, 6:1-10. Before making cache policy decisions, "the cache server identifies the policies related to requests for pre-recorded content (block 504)" and "identifies whether the requested content is stored on the cache server (block 505)." O'Rourke, 6:12-15. Then, cache policy decisions are made based on cache policies of the content, such as whether the requested content is acceptable (block 506) and whether a cached version of the content is an acceptable version (block 510). O'Rourke, 6:20-22, 6:31-34. In other words, O'Rourke's procedure 500 transitions at line 501A from a prior identification of policies related to requests for pre-recorded content (block 504) to application of cache policies for the requested content (e.g., block 510). Thus, after line 501A (which occurs *after determining that the content is cached locally* as recited in 1[b]/8[b]) O'Rourke *determin[es]...a current cache policy* (to determine whether a cached version of the content is an acceptable version at block 510).

121. To the extent PO argues that cache policy decisions performed after line 501A do not disclose or suggest *determining...a current cache policy* that is performed *after determining that the content is cached locally* (as recited in 1[b]/8[b]), O'Rourke's identification of cache policies (block 504) does. As I described above, it would have been obvious for O'Rourke's procedure 500 to identify whether the requested content is stored on the cache server (block 505) and

then identify cache policies related to pre-recorded content (block 504). Thus, identifying the policies related to requests for pre-recorded content (block 504) (*determining...a current cache policy*) is performed *after determining that the content is cached locally* as recited in 1[b]/8[b].

- f. 1[b][ii]/8[b][ii]: (B)(2) determining, by the node and based on said current cache policy associated with the content, whether it is acceptable to serve the content that is cached locally;**

122. In my opinion, O'Rourke discloses or suggests these limitations.

123. O'Rourke teaches that "[i]f the requested content is available, then the cache policies determine whether the cached version of the requested content is an acceptable version." O'Rourke, 6:31-34. And as described further below, if the cached version of the content is determined to be acceptable, it is provided to the client. *Id.*, 6:37-40. Thus, the cache policies determine *whether it is acceptable to serve the content that is cached locally*. And as I described above, each of the cache policies (including the cache policy that determines whether the content stored on the cache server is acceptable) is a *current cache policy associated with the content* and its decisions are *determin[ed], by the node*.

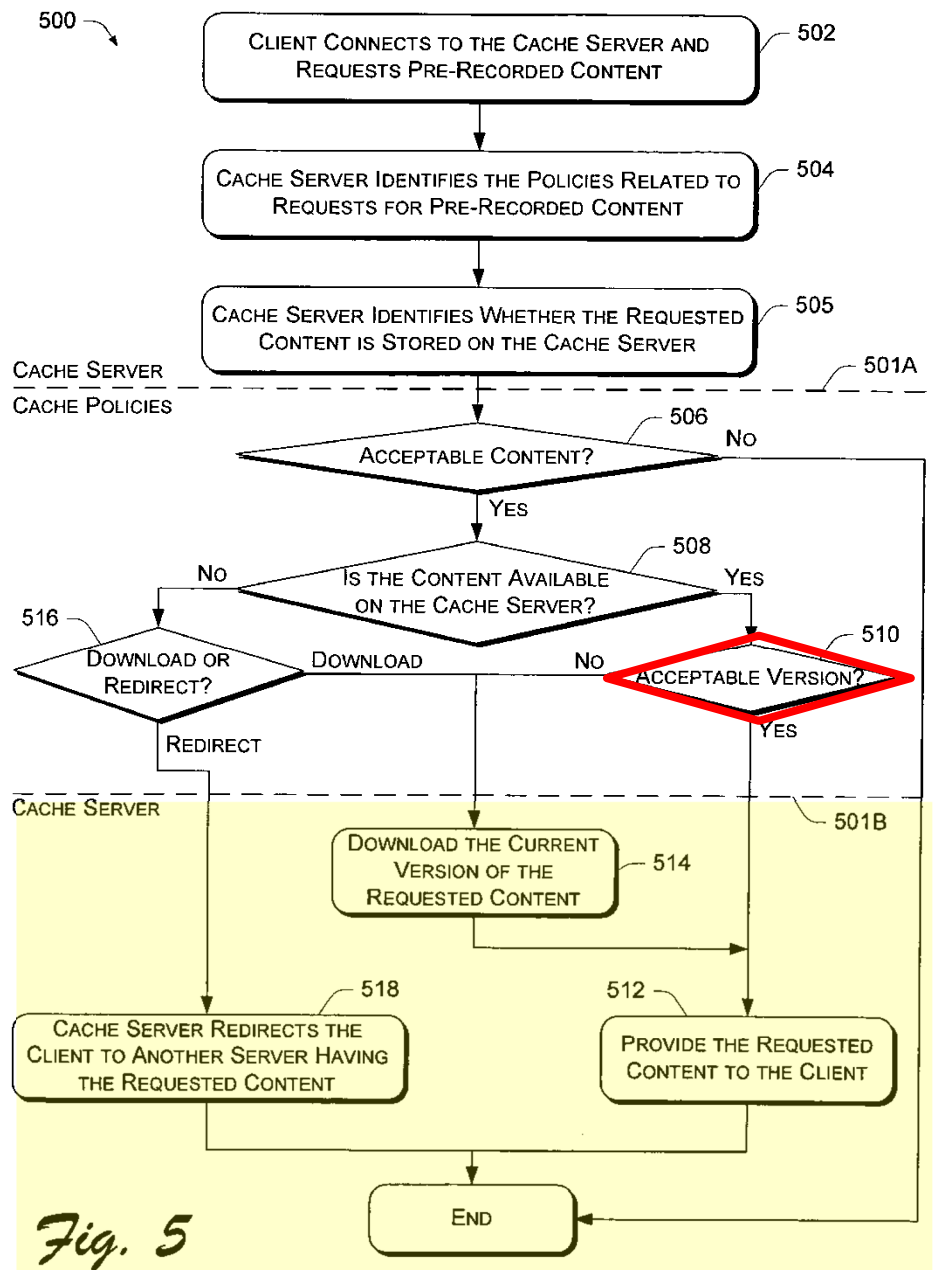
- g. 1[b][iii]/8[b][iii]: (B)(3) based on said determining in (B)(2),**

124. In my opinion, O'Rourke discloses or suggests these limitations.

125. In this limitation, *(B)(2)*, as recited in this claim element (*based on said determining in (B)(2)*), refers to the immediately preceding element, which I have

labeled 1[b][ii]/8[b][ii]. As described in my discussion of that limitation, above, O'Rourke's cache policy determination of whether content stored on the cache server is an acceptable version discloses 1[b][ii]/8[b][ii].

126. I have provided an annotated copy of Figure 5 of O'Rourke below, showing the determination of whether stored content is an acceptable version at block 510 (red). Based on that determination, procedure 500 proceeds to the appropriate block within the section below line 501B (yellow). O'Rourke, 6:5-10; Thus, the blocks of procedure 500 performed after block 510 are based on said determining in (B)(2).

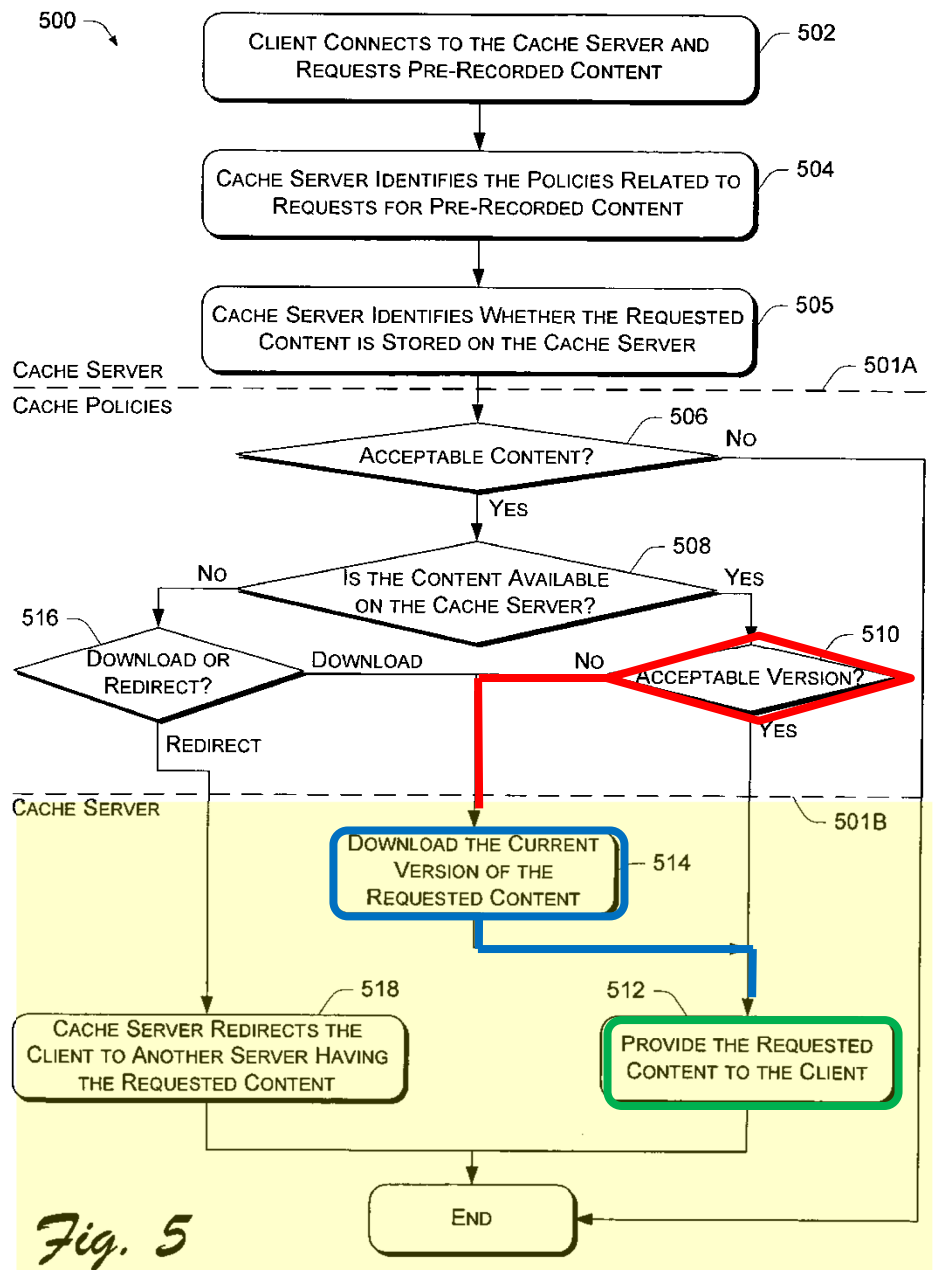


- h. 1[b][iii][A]/8[b][iii][A]: (i) when it is not acceptable to serve the content that is cached locally, obtaining, by the node, a new version of the content and then serving, by the node, the new version of the content, and**

127. In my opinion, O'Rourke discloses or suggests these limitations.

128. O'Rourke teaches that “[i]f the cached version of the requested content is not acceptable, then the procedure branches from block 510 to block 514, where the cache server downloads the current version of the requested content from the origin server.” O'Rourke, 6:40-43. This functionality is *when it is not acceptable to serve the content that is cached locally, obtaining, by the node, a new version of the content*). O'Rourke further describes that, after downloading a current version of the content from the origin server, “the cache server provides the current version to the client (block 512).” *Id.*, 6:51-54. This functionality is *and then serving, by the node, the new version of the content*.

129. I have provided an annotated copy of Figure 5 of O'Rourke below, showing downloading of the current version of the requested content (block 514, blue) and providing the downloaded content to the client (block 512, green) when the cache policy determines the cached version is not acceptable (block 510, red).

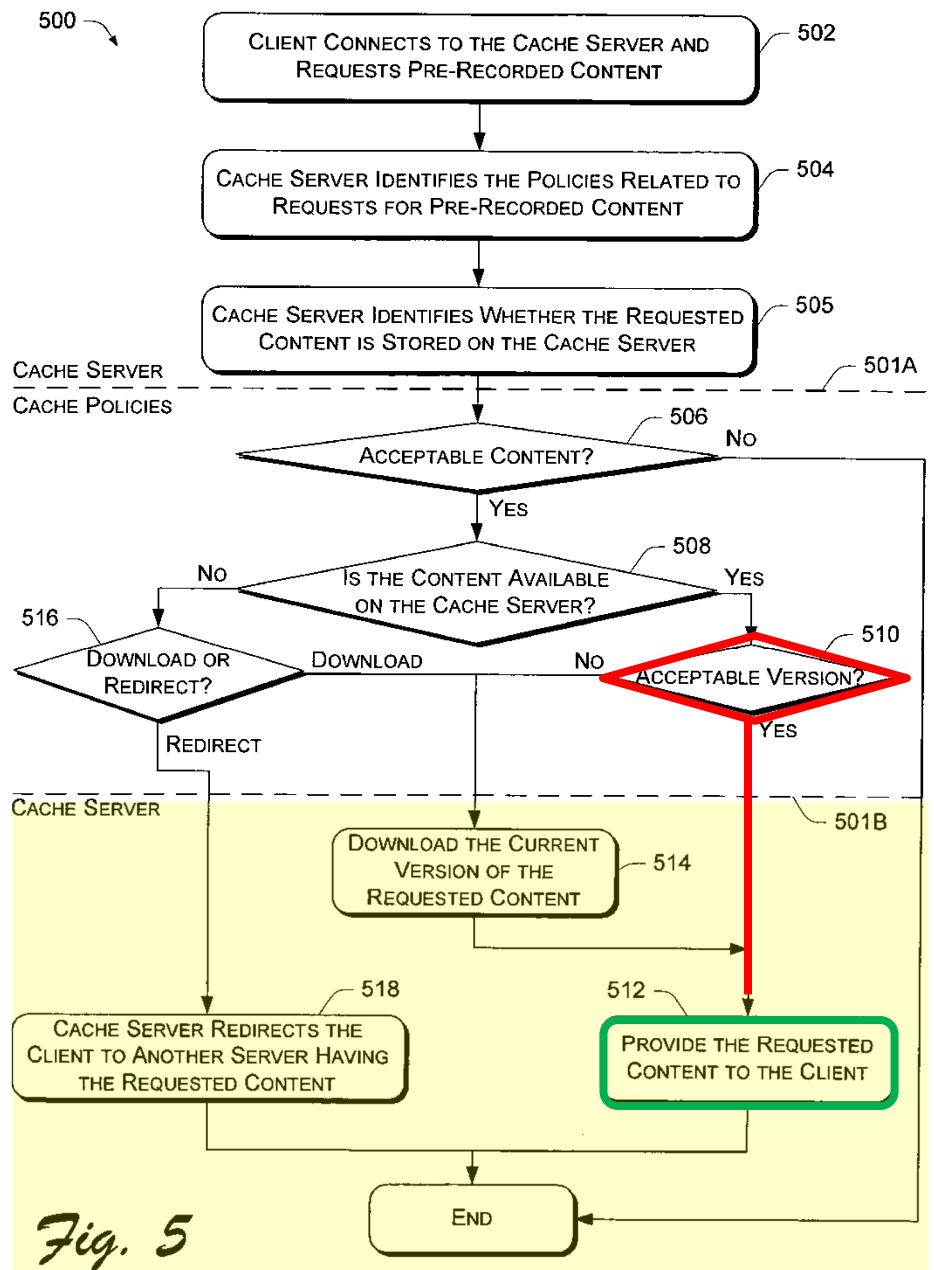


- i. **1[b][iii][B]/8[b][iii][B]: (ii) when it is acceptable to serve the content that is cached locally, serving, by the node, the content that is cached locally.**

130. In my opinion, O'Rourke discloses or suggests these limitations.

131. O'Rourke describes that "[i]f the cached version is acceptable, the procedure continues [from block 510] to block 512, where the cache server provides the cached version of the requested content to the client." O'Rourke, 6:37-40. This functionality is *when it is acceptable to serve the content that is cached locally, serving, by the node, the content that is cached locally.*

132. I have provided an annotated copy of Figure 5 of O'Rourke below, showing providing the content to the client (block 512, green) when the cache policy determines the cached version is acceptable (block 510, red).



2. Claims 2, 9

a. 2[pre]: The method as recited in claim 1 further comprising:

133. In my opinion, O'Rourke discloses or suggests these limitations. As I describe above, O'Rourke teaches "a procedure 500 for handling a client request to

receive pre-recorded content. The procedure 500 illustrates the functions performed by the cache server as well as the decisions made by the cache policies.” O’Rourke, 6:1-5. O’Rourke’s procedure 500 discloses or suggests *the method as recited in claim 1*.

b. 9[pre]: The computer program product as recited in claim 8 wherein the method further comprises:

134. In my opinion, O’Rourke discloses or suggests these limitations. As I describe above, O’Rourke teaches “computer-readable media [that] provide[s] nonvolatile storage of computer readable instructions, data structures, program modules and other data for computer 642.” O’Rourke, 9:4-7. For example, O’Rourke’s procedure 500 may be performed by the computer 642. O’Rourke, 6:2-5 (“The procedure 500 illustrates the functions performed by the cache server”). The computer readable instructions and program modules are the *computer program product as recited in claim 8*, and O’Rourke’s procedure 500 discloses or suggests what *the method further comprises*.

c. 2[a]/9[a]: (C) when it is determined that said content is not cached locally, then (C)(1) obtaining the content and then (C)(2) serving the content obtained in (C)(1).

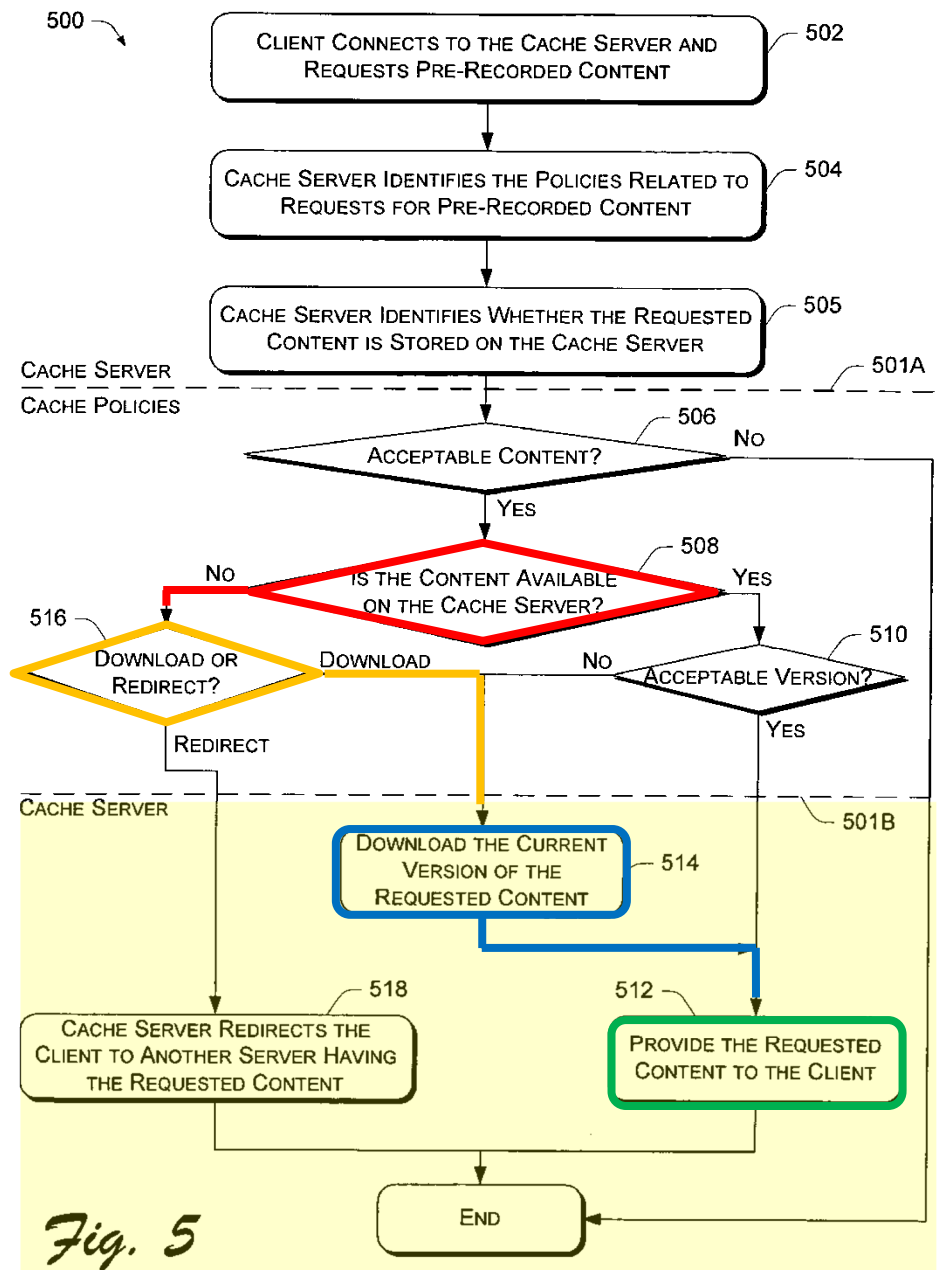
135. In my opinion, O’Rourke discloses or suggests these limitations.

136. O’Rourke teaches that if it is “determine[d] that the requested content is not available on the cache server, then the cache policies determine whether to download the requested content or redirect the client to a different server.”

O'Rourke, 6:55-58. This process of O'Rourke is *(C) when it is determined that said content is not cached locally*. Based on the cache policies, "the cache server downloads the current version of the requested content from the origin server."

O'Rourke, 6:58-62. This process is *then (C)(1) obtaining the content*. After downloading the content from the origin server, the cache server *then (C)(2) serves the content obtained in (C)(1)*. O'Rourke, 6:51-54, Fig. 5.

137. I have provided an annotated copy of Figure 5 of O'Rourke below, showing downloading the current version of the requested content from the origin server (box 514, blue) and providing the content to the client (block 512, green) when the requested content is not available on the cache server (block 508, red) and the cache policies determine to download the content (block 516, orange).



3. **Claims 4, 11:** [The method as recited in claim 1]/[The computer program product as recited in claim 8] wherein the current cache policy is determined from a location distinct from the node.
138. In my opinion, O'Rourke discloses or suggests these limitations.

139. O'Rourke teaches that "cache policies are contained in one or more cache plug-in modules," and that "the cache plug-in modules can run as part of another process...on a different computer system that supports the Windows Media Server." O'Rourke, 4:57-60, 4:66-5:2. That is, the cache plug-in modules (containing the *current cache policy*) can be *determined from a location distinct from the node* (i.e., a different computer system than the cache server).

4. Claims 5, 12: [The method as recited in claim 4]/[The computer program product as recited in claim 11] wherein the location is a database.

140. In my opinion, O'Rourke discloses or suggests these limitations.

141. As I describe above, O'Rourke teaches that "cache policies are contained in one or more cache plug-in modules," and that "the cache plug-in modules can run as part of another process...on a different computer system that supports the Windows Media Server." O'Rourke, 4:57-60, 4:66-5:2. Further, a POSITA would have understood that O'Rourke's storing of cache policies and/or cache plug-in modules on a different computer system (*location*) includes being stored on a database of the different computer system. For example, O'Rourke describes the use of data storage devices to for storing data and other information used, such as cache policies. *See, e.g.*, O'Rourke, 3:39-42, 5:17-21. A POSITA would have understood that when storing large amounts of data (e.g., cache policies) on storage devices, databases provided various advantages such as improved

security, data integrity, and scalability. *See, e.g.*, Silberschatz, pp. 3-6 (describing the advantages of databases over other approaches to maintaining data, such as file-processing systems). And databases were commonly used to maintain information within CDNs. *See, e.g.*, Buyya, pp. 117-119 (describing different approaches to maintain and replicate databases at different locations of a CDN). Thus, it would have been obvious for the different computer system to utilize a *database* in the context of claims 5 and 12.

5. Claims 6, 13: [The method as recited in claim 1]/[The computer program product as recited in claim 8] wherein the cache policy relates to caching content within the CDN.

142. In my opinion, O'Rourke discloses or suggests these limitations.

143. O'Rourke teaches that the cache server can be a Windows Media Server, and "[a]ny number of cache plug-in modules can be loaded into the Windows Media Server. The combination of all loaded cache plug-in modules defines the set of cache policies." O'Rourke, 4:57-63. The cache plug-in modules loaded by the system can change, such that "the operation of a cache server can be modified simply by changing the cache plug-in modules in the cache server." O'Rourke, 5:6-9. The cache policies control the operation of the cache server, such as what type of content to cache on the cache server, and how to respond to a cache miss. O'Rourke, 7:52-53, 8:1-2. Thus, the cache policies *relate to caching content within the CDN* (e.g., at the cache server).

B. Ground 1B: Obviousness over O'Rourke and MS-MSSO

1. Motivation to Combine O'Rourke and MS-MSSO

144. In my opinion, a POSITA would have been motivated to modify the content delivery system of O'Rourke to utilize the HTTP Streaming Protocol taught by MS-MSSO.

145. As explained above, O'Rourke identified the problem that “[e]xisting cache servers are manufactured with a substantially fixed architecture that is difficult to customize” and thus “minimize the opportunities for a customer to modify the cache server to modify their specific needs.” O'Rourke, 2:8-26. In other words, O'Rourke identified that prior art solutions suffered from the shortcoming of not allowing content providers to specify how their content was cached.

146. To address this perceived problem, O'Rourke taught a cache server with a “flexible architecture that separates the various cache functions from the policies that determine which functions are to be performed in a particular situation.” *Id.*, 2:29-32. O'Rourke explained that this allowed a “common cache server to be configured in a variety of different ways depending on the set of policies applied to the cache server” and allowed “different manufacturers or end users [to] create different cache server functionality based on the set of policies selected.” *Id.*, 2:32-37. O'Rourke explains that its cache server may be a Windows Media Server (*id.*, 4:57-60), a Microsoft product that was commercially available at the time.

147. O'Rourke describes this implementation in detail. Specifically, in this implementation, "the set of cache policies are contained in one or more cache plug-in modules that are loaded in the Windows Media Server and called by the Windows Media Server" and these cache plug-in modules can "run in the same space as the Windows Media Server" or, alternatively, "can run as part of another process on the same computer system or on a different computer system that supports that Windows Media Server....Thus, the operation of a cache server can be modified simply by changing the cache plug-in modules in the cache server." *Id.*, 4:57-5:8.

148. A POSITA, aware of the benefits of O'Rourke to configure existing cache servers including the Windows Media Server, would have looked to MS-MSSO for further details about how the Windows Media Server operated and what protocols and functions were supported. And a POSITA would further have recognized that the Windows Media HTTP Streaming Protocol, used by Windows Media Servers, provides additional advantages when, for example, "the multimedia data needs to pass through a proxy or a firewall." MS-MSSO, 47. For these reasons, a POSITA, aware of the benefits both of allowing customizable caching policies and of optimizing data transmission in various network environments, including those involving proxies and firewalls, would have been motivated to further improve upon O'Rourke (cache servers with customizable caching policies, configured as a Windows Media Server) to utilize the Windows Media HTTP Streaming Protocol,

as taught by MS-MSSO. This combination would have involved combining prior art elements (MS-MSSO's Windows Media HTTP Streaming Protocol, and O'Rourke's CDN incorporating a Windows Media Server) according to known methods (deploying a Windows Media Server as a cache server as taught by O'Rourke, and utilizing the Windows Media HTTP Protocol as taught by MS-MSSO) to yield predictable results (improved caching and network performance).

149. A POSITA would have had a reasonable expectation of success combining O'Rourke with MS-MSSO in this manner. O'Rourke already taught a cache server that could be implemented as a Windows Media Server. O'Rourke, 2:54-59, 5:10-15, 7:41-44. A POSITA would have recognized that the Windows Media Server taught in O'Rourke could be readily configured to utilize the Windows Media HTTP Streaming Protocol, and the combination would yield predictable results. This combination configuration would have been well within the capabilities of a POSITA because O'Rourke provides details regarding how to implement the invention using a Windows Media Server as its cache server, and MS-MSSO describes the use of various protocols within such a Windows Media Server, including the Windows Media HTTP Streaming Protocol. Additionally, the resulting system's elements would perform functions performed prior to combination—O'Rourke's CDN, utilizing a Windows Media Server, would allow users to

customize caching policies, and utilizing the Windows Media HTTP Streaming Protocol would improve network performance.

2. Claims 3, 10: [The method as recited in claim 1]/[The computer program product as recited in claim 8] wherein the request comprises an HTTP request and wherein the content comprises one or more resources.

150. In my opinion, the combination of O'Rourke and MS-MSSO discloses, or at least suggests, claims 3 and 10's additional limitations.

151. As described above, O'Rourke describes that its cache server may be a Windows Media Server. O'Rourke, 4:57-60. As a POSITA would have understood, cache servers, such as Windows Media Servers, utilize HTTP. For example, the MS-MSSO reference describes that Windows Media HTTP Streaming Protocol, used by Windows Media Servers, "depends on HTTP for the transfer of all protocol messages, including the transfer of the multimedia data." MS-MSSO, 47. It further explains that "[t]he Windows Media HTTP Streaming Protocol depends on HTTP 1.0." MS-MSSO, 47. As a POSITA would have understood, the use of Windows Media HTTP Streaming Protocol was advantageous in various circumstances, for example, when "the multimedia data needs to pass through a proxy or a firewall." MS-MSSO, 47. A POSITA would also have understood it was typical for content delivery systems to use HTTP to request and deliver content. Buyya, p. 18 ("Akamai delivers static and dynamic content over HTTP") and p. 19 ("[Limelight Networks']

content delivery services include HTTP/Web distribution of digital media files such as video, music, games, software and social media... Limelight Networks support...Microsoft Windows Media”). It would therefore have been obvious in the combined O’Rourke-MS-MSSO system for the cache server configured as a Window Media Server (as taught by O’Rourke) to utilize the Windows Media HTTP Streaming Protocol (as taught by MS-MSSO). Thus, in the O’Rourke-MS-MSSO combination, requests to the cache server *comprises an HTTP request*.

152. O’Rourke describes “procedure 500 for handling a client request to receive pre-recorded content, such as “media content, such as movies, television programs, concerts, sporting events, and the like,” and the cache server is capable of “providing a higher quality real-time video image to the local clients.” O’Rourke, 3:46-55; 6:1-5. In other words, in O’Rourke, the requested content *comprises one or more resources* (e.g., audio and/or video data). Similarly, MS-MSSO explains that “[t]he Windows Media HTTP Streaming Protocol is used to transfer real-time multimedia data (that is, audio and video)”). MS-MSSO, 46.

C. Ground 1C: Obviousness over O’Rourke and Blumofe

1. Motivation to Combine O’Rourke and Blumofe

153. In my opinion, a POSITA would have been motivated to modify the content delivery system of O’Rourke to add an extender as taught by Blumofe.

154. The Background of O'Rourke describes that within a network, cache servers could be deployed between origin servers and end users to improve network performance. O'Rourke, 1:31-41. O'Rourke identified a purported problem with conventional solutions that "[e]xisting cache servers are manufactured with a substantially fixed architecture that is difficult to customize" and this "minimize[s] the opportunities for a customer to modify the cache server to modify their specific needs." O'Rourke, 2:8-26. In other words, O'Rourke identified that prior art solutions suffered from the shortcoming of not allowing content providers to specify how their content was cached.

155. To address this problem, O'Rourke taught a cache server with a "flexible architecture that separates the various cache functions from the policies that determine which functions are to be performed in a particular situation." O'Rourke, 2:29-32. In this system, a "common cache server [can] be configured in a variety of different ways depending on the set of policies applied to the cache server." O'Rourke, 2:32-35. This allows "different manufacturers or end users [to] create different cache server functionality based on the set of policies selected." O'Rourke, 2:35-37.

156. O'Rourke disclosed that its solution may be used in various network environments. For example, O'Rourke taught that the invention could be advantageously used "to distribute media content across a network" and "[c]ache

server 300 may use any type of data communication interfaces necessary to communicate with other devices.” O’Rourke, 2:54-59, 5:10-15. O’Rourke also taught that the cache policies could be defined based on the type of network and the other components in the system. O’Rourke, 7:41-44 (“The cache policies are typically defined based on an administrator’s knowledge of the origin servers, cache servers, clients, networks and other components in the environment in which the cache server is located...”). O’Rourke disclosed particular embodiments that could be used in LAN and WAN networks. O’Rourke, 3:34-39, 9:39-53. However, O’Rourke was silent as to how the invention could be extended to other types of networks, such as, for example, wireless networks.

157. As described above, Blumofe taught “extend[ing] [a CDN] into one or more mobile or wireline networks that cannot or do not otherwise support fully-managed CDN edge servers.” Blumofe, Abstract. Blumofe explained that because of the way these networks “are architected and operated, it is not always possible or practical to deploy fully-managed CDN servers therein (i.e., close to end-users).” Blumofe, [0007]. Blumofe observed that in 3G wireless mobile networks, for example, the gateways used to connect wireless end-users to the internet, where CDN servers could be deployed, could be thousands of miles and hundreds of milliseconds away from the end-users. Blumofe, [0007].

158. To address these perceived problems, Blumofe described a CDN Extender, which could be implemented in network environments that do not support CDN services (such as a wireless mobile network), and operated as a cache beyond the edge of the CDN. Blumofe, [0025], [0027]; see also Blumofe, Fig. 3. Blumofe described that the “Extender” could be deployed on the network “as a passive web caching proxy that is beyond the edge of the CDN but that serves CDN-provisioned content under the control of the CDN.” Blumofe, [0009]. The CDN controlled the Extender, and caching policies were supplied by the CDN to the Extender, i.e., caching policies within the CDN relate to caching content outside of the CDN. Blumofe, [0036] (“As noted above, the Extender caches and serves the CDN customer content based on information it receives, in terms of content and cache-control instructions, from the CDN edge servers.”). This solution enabled serving of “CDN-provisioned content under the control of the CDN” where it would otherwise not be “possible or practical to deploy fully-managed CDN servers.” Blumofe, [0007]-[0009].

159. A POSITA, aware of the benefits both of allowing customizable caching policies and of providing caching functionality close to wireless end users in various network environments, would have been motivated to further improve upon O’Rourke (cache servers with customizable caching policies) to allow for

further caching resources within a wireless network. And a POSITA would have further recognized that Blumofe would provide this benefit to O'Rourke.

160. Both references were directed to content delivery networks. O'Rourke, 1:19-21; Blumofe, [0005]. And both described shortcomings with conventional CDN architectures and solved these problems by managing caches deployed between origin servers and end users. O'Rourke, 4:10-22; Blumofe, [0009]. In addition, both references described methods for the content owner to customize the cache policies used for caching content. O'Rourke, 5:41-47; Blumofe, [0030].

161. Blumofe taught an Extender that was designed to work in combination with an existing CDN, such as the CDN of O'Rourke. For example, Blumofe taught that the Extender "is deployed in the mobile or wireline network, preferably as a passive web caching proxy that is beyond the edge of the CDN but that serves CDN-provisioned content" and that it "preferably delivers all forms of CDN content (i.e., content that has been off-loaded by content providers for delivery by the CDNSP [CDN service provider, i.e., the entity managing the CDN] over the CDN)." Blumofe, [0009], [0029].

162. A POSITA would have found it obvious to combine O'Rourke, which described a method of managing cache servers within a CDN, with Blumofe, which described managing caches both within the CDN and outside the CDN (e.g., within the Extender). That is, a POSITA would have found it obvious to utilize an extender,

as taught by Blumofe, with the CDN taught by O'Rourke. A POSITA would have recognized the benefits this combination provides. For example, using an extender as taught by Blumofe would have allowed O'Rourke's cache server to be used in conjunction with other network environments (e.g., wireless networks) where it otherwise could not have been deployed. And this combined system would likewise have allowed content to be cached closer to the wireless users, providing improved network performance. Using an extender (as taught in Blumofe) in combination with O'Rourke's cache server would enable cache policies to be enforced consistently both inside and outside the CDN. This would have involved applying a known technique (caching content in wireless networks) to improve similar devices (CDNs) in the same way (deploying caches in other network environments and/or closer to the user). In addition, this would have involved combining prior art elements (an extender as taught in Blumofe and O'Rourke's CDN) according to known methods (providing cache servers between origin servers and end users) to yield predictable results (improved network performance).

163. A POSITA would have had a reasonable expectation of success combining O'Rourke with Blumofe in this manner. O'Rourke already taught a cache server that could be adapted to various networks. O'Rourke, 2:54-59, 5:10-15, 7:41-44. A POSITA would have recognized that an extender as taught in Blumofe could be readily incorporated into O'Rourke's system, and the combination would yield

predictable results. This combination would be well within the capabilities of a POSITA because an extender as taught in Blumofe was already designed to work with an existing CDN. Blumofe, [0009] (“[t]he CDN is extended into one or more mobile or wireline networks that cannot or do not otherwise support fully-managed CDN edge servers . . . [the] ‘Extender’ is deployed in the mobile or wireline network, preferably as a passive web caching proxy that is beyond the edge of the CDN but that serves CDN-provisioned content.”), [0029] (the Extender “preferably delivers all forms of CDN content (i.e., content that has been off-loaded by content providers for delivery by the CDNSP over the CDN).” In addition, O’Rourke and Blumofe describe similar CDN environments and communications between origin servers, caches, and clients, such that a POSITA would have found it straightforward to modify O’Rourke’s CDN to implement an extender as taught in Blumofe. The resulting system’s elements would perform functions performed prior to combination—O’Rourke’s CDN would allow users to customize caching policies, and the extender would enable content to be cached in in wireless networks and/or closer to the user.

164. A POSITA would have understood that in such a combined system, the CDN would send the cache control information to the extender so that the extender would perform according to the CDN customer’s rules, and the cache policy that configures the cache server within the CDN (e.g., with respect to whether cached

content has expired or is an acceptable version) would also be used to configure the extender outside the CDN. Blumofe, [0030] (“To deliver CDN content, the Extender acts under the control of the CDN. This control is necessary because, typically, it is desired (by the CDN customer) that the correct content be delivered according to customer metadata rules”), [0036] (“the Extender caches and serves the CDN customer content based on information it receives, in terms of content and instructions, from the CDN edge servers”), [0037] (“When the CDN edge server responds to an Extender, preferably it communicates some or all available cache control information in one or more HTTP response headers. This cache control information includes, for example, time-to-live (TTL) expiration times...”). The combination would have been well within the capabilities of a POSITA because it would have merely required modifying the CDN of O’Rourke to include an extender taught by Blumofe.

2. Claims 7, 14: [The method as recited in claim 6]/[The computer program product as recited in claim 13] wherein the cache policy also relates to caching content outside the CDN.

165. In my opinion, the combination of O’Rourke and Blumofe discloses, or at least suggests, claims 7 and 14’s additional limitations.

166. As described above, Blumofe teaches a CDN Extender, which operates so that “[t]he CDN is extended into one or more mobile or wireline networks that cannot or do not otherwise support fully-managed CDN edge servers.” Blumofe,

[0009]. The Extender “is deployed in the mobile or wireline network, preferably as a passive web caching proxy that is beyond the edge of the CDN but that serves CDN-provisioned content.” *Id.* The extender of the combined O’Rourke-Blumofe system is thus *outside the CDN*. Blumofe, [0033] (“[T]he Extender is located within or in association with a network that is distinct from the CDN overlay network.”). Although it is not part of the CDN, the extender “preferably delivers all forms of CDN content (i.e., content that has been off-loaded by content providers for delivery by the CDNSP over the CDN).” Blumofe, [0029]. Thus, “the Extender can cache and deliver cacheable content, including on-demand media.” *Id.* In other words, any CDN content cached by the extender in the combined O’Rourke-Blumofe system is similarly *cached content outside the CDN*. Blumofe, [0050] (“[T]he Extender is implemented within a UMTS 3G-compliant [cellular] network in association with the operator’s core network infrastructure...and used to facilitate delivery of content (including CDN content) to mobile-connected devices.”).

167. As I describe above, when the Extender of Blumofe receives an end-user request for content, the Extender determines whether the requested content is already in its cache. Blumofe, [0033]. If the content is not in its cache, the Extender sends an HTTP GET request to the CDN to fetch the content. *Id.* Based on the response received from the CDN (e.g., with the content), the Extender “serv[es] to

the requesting end-user an appropriate HTTP response code and/or the content request.” Blumofe, [0033].

168. As described above, Blumofe teaches that “[t]o deliver CDN content, the Extender acts under the control of the CDN.” Blumofe, [0030]. This is necessary because “it is desired (by the CDN customer) that the correct content be delivered according to customer metadata rules.” *Id.* Blumofe further explains that “the Extender caches and serves the CDN customer content based on information it receives, in terms of content and cache-control instructions, from the CDN edge servers.” Blumofe, [0036]. For example, “[w]hen the CDN edge server responds to an Extender, preferably it communicates some or all available cache control information in one or more HTTP response headers.” Blumofe, [0037]. The cache control information can include, for example, “time-to-live (TTL) expiration times.” Blumofe, [0037].

169. In this combination, the cache control information returned by the CDN controls the expiration time of the cached content on the extender, and thus *relates to caching content outside the CDN*. Moreover, Blumofe describes that the CDN sends the cache control information to the Extender so that the Extender performs according to the CDN customer’s rules. Blumofe, [0030], [0036]. And it would have been obvious, in the combined O’Rourke-Blumofe system, that the *cache policy* that configures the cache server within the CDN (e.g., with respect to whether cached

content has expired or is an acceptable version) would also be used to configure the extender outside the CDN. Otherwise, the extender would operate differently than the CDN's cache server. Blumofe, [0054] ("Preferably, an Extender serves a response from its cache if the CDN edge server to which it is directed would have served the same response to the client."). Thus, the *cache policy* (as described in §Error! Reference source not found.) *also relates to caching content outside the CDN.*

D. Ground 2A: Obviousness over Middleton

1. Claims 1, 8

a. 1[pre]: A computer-implemented method, operable with a content delivery network (CDN), the method comprising:

170. In my opinion, if the preamble is limiting, Middleton discloses or suggests it.

171. Middleton discloses "*methods*, systems and devices for providing an extensible content delivery platform." Middleton, Abstract. Middleton's disclosed *methods* "include identifying a plurality of discrete events of a content delivery process of a *content delivery network* and providing a structured object model comprising a plurality of objects instantiated and available at the plurality of discrete events." *Id.* Since Middleton's disclosed methods are part of a content delivery process of a content delivery network (CDN), the *methods* are *operable with a content delivery network (CDN).*

172. Middleton's Figure 1 "illustrates an example network environment 100 suitable for distributing content supporting an extensible content delivery platform according to various embodiments. A computer user may access a content distribution network (CDN) 102 using a computing device, such as a desktop computer 104." Middleton, [0027]. In other words, Middleton's CDN 102 supports the "extensible content delivery platform" also referenced in Middleton's Abstract (described above). The CDN 102 includes a "streaming server computer 108," which Middleton also refers to as an "edge server 108" or "edge server computer 108." *Id.*, [0031], [0031]. I have provided an annotated copy of Middleton's Figure 1 below, with the CDN 102 highlighted in red and the edge server 108 highlighted in blue.

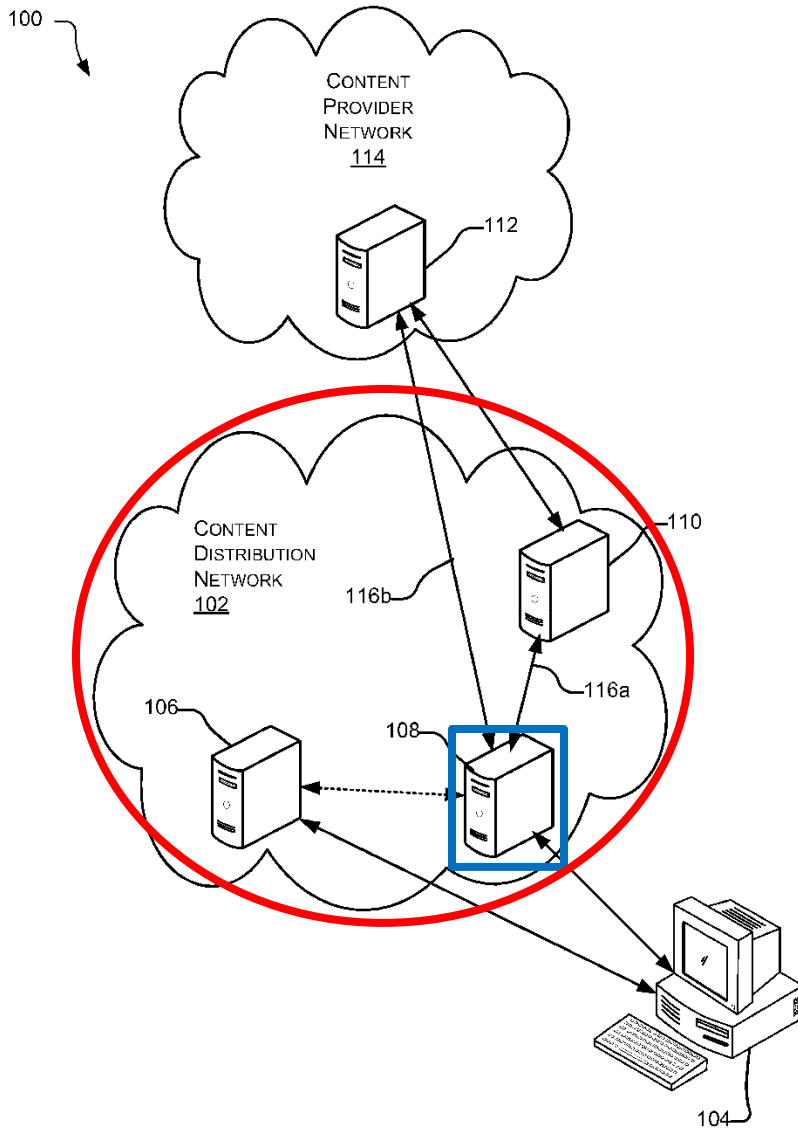


Fig. 1

173. Middleton discloses that the “edge server 108 comprises an extensible content delivery platform, such as an event-based application programming interface (API) that provides an extensible content delivery platform. The extensible content delivery platform provides for applications and services to be built around

underlying content to be served.” Middleton, [0035]. The methods that provide Middleton’s extensible content delivery platform, including those provided by the edge server 108 using APIs, are *computer-implemented method[s]*.

- b. 8[pre]: A computer program product having computer readable instructions stored on non-transitory computer readable media, the computer readable instructions including instructions for implementing a computer-implemented method, said method operable on a device comprising hardware including memory and at least one processor and running a service on said hardware, said method operable in a content delivery service (CDN), said method comprising:**

174. In my opinion, if the preamble is limiting, Middleton discloses or suggests it.

175. Middleton discloses “a computer system 1200 upon which embodiments of the present invention may be implemented and carried out. For example, one or more computing devices 1200 may be used to support an extensible content delivery platform (e.g., for streamed content within a content distribution network). Computer system 1200 generally exemplifies any number of computing devices, including general purpose computers (e.g., desktop, laptop or server computers) or specific purpose computers (e.g., embedded systems).” Middleton, [0123]. Middleton’s computer system 1200 is a *computer program product*.

176. Middleton discloses that the computer system 1200 (*computer program product*) includes at least one processor 1202, main memory 1204, and mass storage

1207 (*non-transitory computer readable media*). Middleton, [0123], [0124]. Furthermore, “[m]ass storage 1207 can be used to store information and instructions” (*computer readable instructions*). *Id.*, [0125].

177. Middleton explains that “functionality supported by computer system 1200 and, more particularly, functionality associated with extensible content delivery application 1250-1 and extensible content delivery process 1250-2 is discussed above with reference to FIGS. 1-11.” Middleton, [0131]. That is, the information and instructions stored on the computer system 1200 are used to perform Middleton’s extensible content delivery process (*instructions for implementing a computer-implemented method, said method operable on a device comprising hardware including memory and at least one processor and running a service on said hardware, said method operable in a content delivery service (CDN)*).

- c. 1[a]/8[a]: (A) in response to a request for content, determining, by a node [in]/[of] the CDN, if the content is cached locally at the node;**

178. In my opinion, Middleton discloses or suggests these limitations.

179. Specifically, Middleton discloses a process for handling a request to deliver content (*in response to a request for content*), performed by an edge server (*node*), that determines whether the requested content is locally cached (*determining...if the content is cached locally at the node*).

180. Figure 2 of Middleton “illustrates a streaming content delivery framework 200 adapted to supporting an extensible content delivery platform including an edge server computer 202 and a media access server computer 204.” Middleton, [0047]. The edge server computer (*node*) “is configured with modules operable to retrieve content from the MAS 204, if necessary, while streaming the content to an entity that has requested the content.” *Id.* The edge server computer modules include “a media streaming server 206, a media streaming broker 208, a stream caching module 210 and a content cache 212.” *Id.*, [0048]. I have provided an annotated copy of Middleton’s Figure 2 below, with the content delivery framework 200 highlighted in red, edge server 202 highlighted in blue, and the MAS 204 highlighted in green.

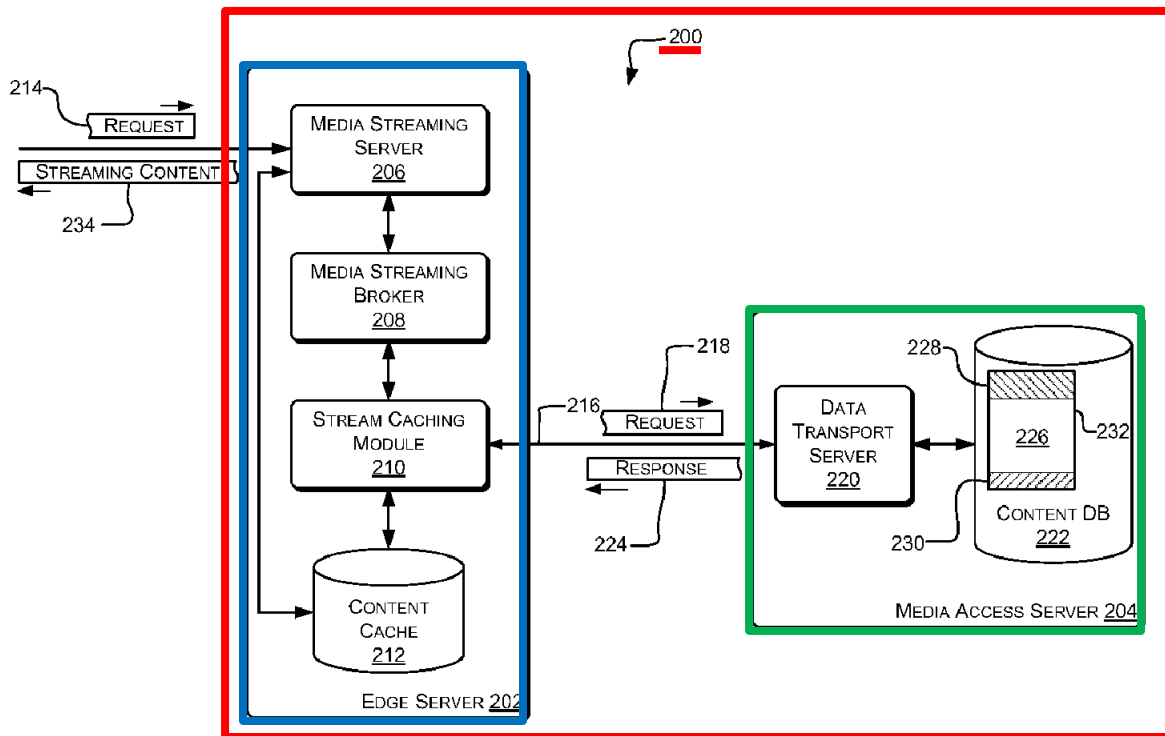


Fig. 2

181. As illustrated in Figure 2, the edge server 202 is part of (and in) the content delivery network 200. The edge server 202 is therefore *a node in the CDN* as recited in 1[a], and *a node of the CDN* as recited in 8[a].

182. Figures 5-7 of Middleton “are flow charts illustrating processes for handling a request to deliver content. As described below, the extensible content delivery platform is supported in the processes.” Middleton, [0079]. Figure 5, in particular, illustrates a “content request handling operation 500.” *Id.*, [0080]. The operations of the processes for handling a request to deliver content, such as content request handling operation 500, “can be performed by functional modules such as one or more of the media streaming server 206, streaming media broker 208 and

stream caching module 210 (FIG. 2), or other modules.” *Id.*, [0079]. As illustrated in the annotated Figure 2 that I provided above, the aforementioned modules are part of the edge server 202 (*node*). In other words, the operations of Middleton’s processes for handling a request to deliver content, including the content request handling operation 500 of Figure 5, are performed by modules of the edge server (*node*). Thus the steps performed by those processes are done *by a node in the CDN*.

183. As I described above, Middleton’s Figure 5 illustrates a “content request handling operation 500.” Middleton, [0080]. As illustrated in Figure 5 (which I have provided below), the content request handling operation 500 begins when “a request is initially received for specified content in receiving operation 502.” *Id.* Middleton explains that “[t]he requested *content* is identified in the *request*.” *Id.* Therefore, referring to the annotated Figure 5 below, operations of the content request handling operation 500 that follow operation 502 (*a request for content*) are performed *in response to* operation 502 (the *request for content*).

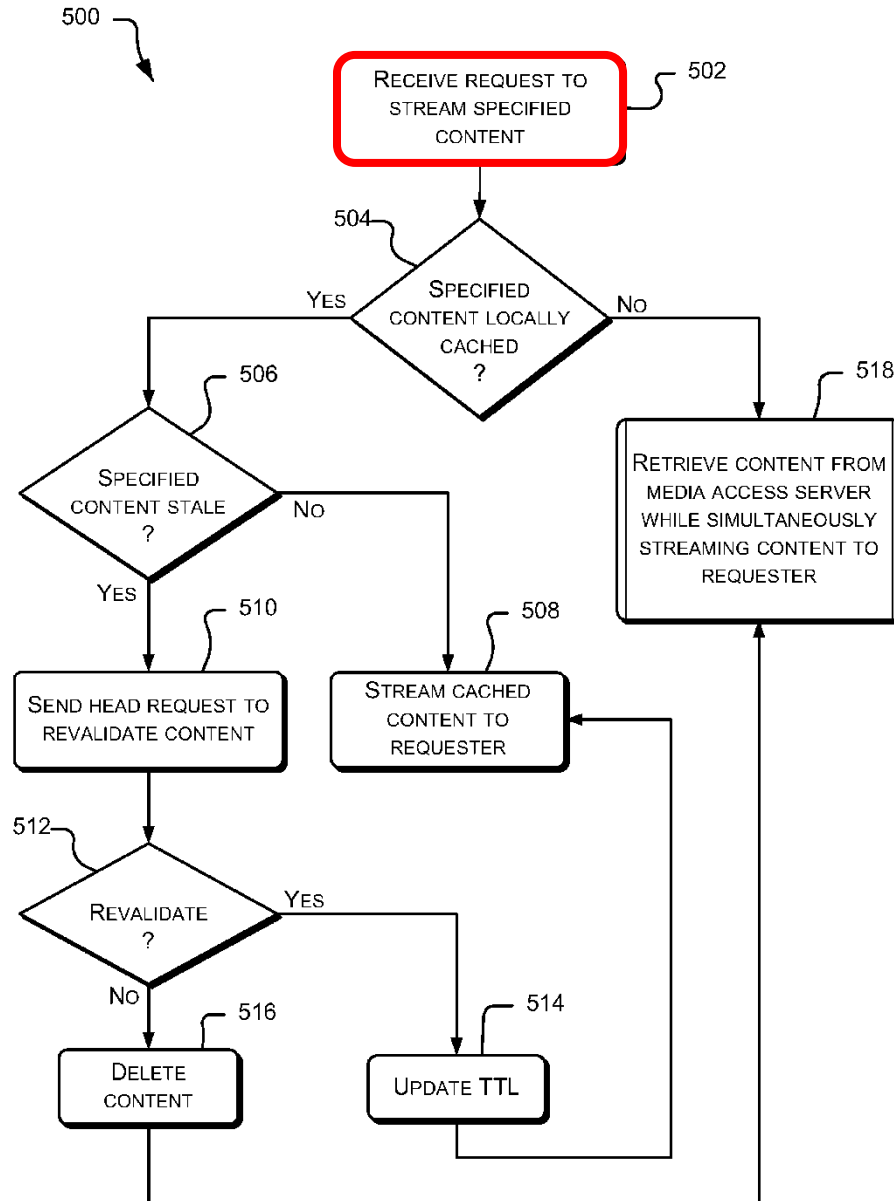


FIG. 5

184. After receiving the request for content at operation 502, “[a] query operation 504 determines if the requested content exists in local cache.” Middleton, [0080]. Middleton’s query operation 504 illustrates *determining...if the content is*

cached locally at the node. This determination is made *in response to* the request for content. *See, e.g.,* Middleton, [0049]-[0051] (describing querying a cache manager 304 to determine whether requested content is in a cache 212, made in response to a request from the media streaming broker 208, which itself makes the request in response to a content request 214 received by a media streaming server 206). Figure 5 of Middleton reflects the same. As illustrated in the annotated Figure 5 that I have provided below, the determination is made whether the requested content exists in the local cache at operation 504 (highlighted blue), which is performed after the request for content is received at operation 502 (highlighted red), and is thus done *in response to* the request for content.

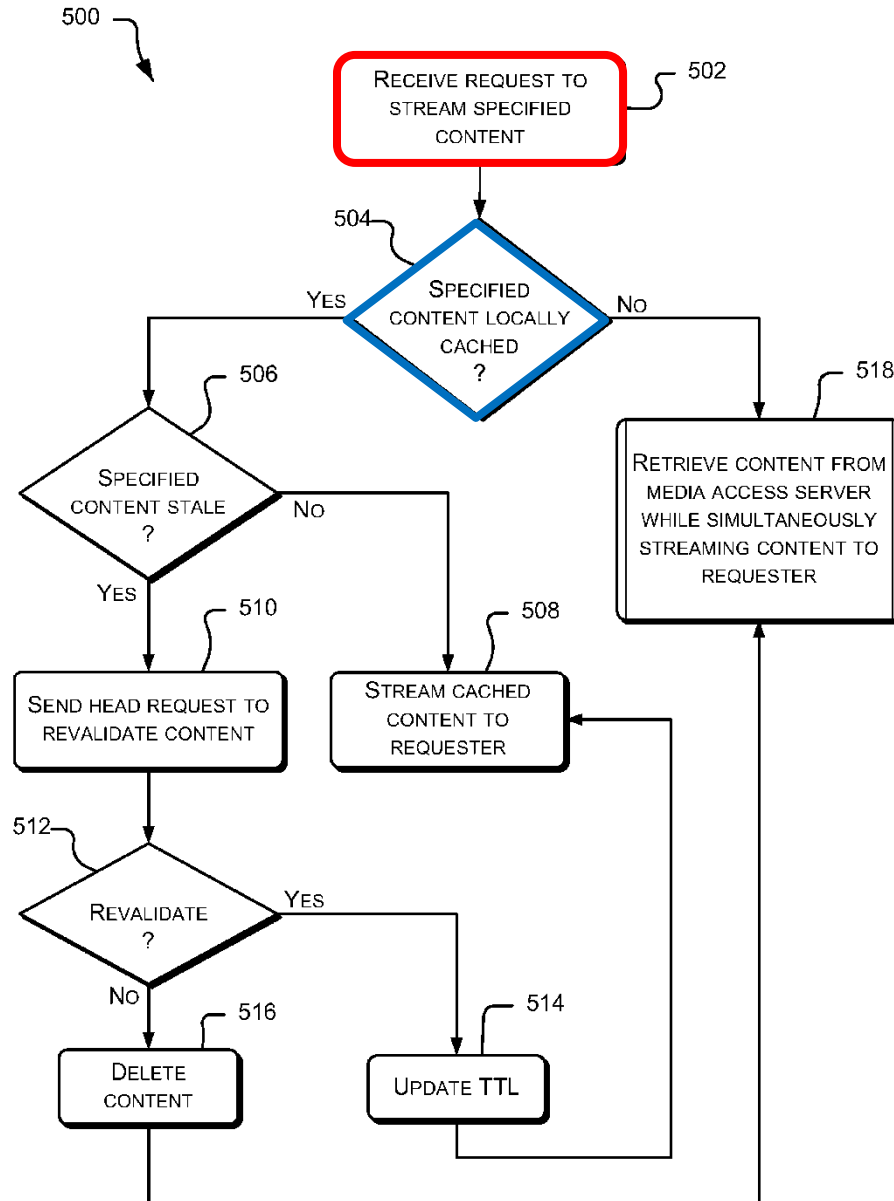


Fig. 5

- d. 1[b]/8[b]: (B) when it is determined [by the node]/[] that said content is cached locally, then, after determining that the content is cached locally:

185. In my opinion, Middleton discloses or suggests these limitations.

186. Specifically, Middleton discloses operations (e.g., query operation 506 and those that follow) of content request handling operation 500 that are performed when query operation 504 determines that requested content exists in the local cache (*when it is determined that said content is cached locally*).

187. As I described above in §IX.A.1.c (Ground 2A, 1[a]/8[a]), Middleton's content request handling operation 500 (illustrated in Figure 5) performs "[a] query operation 504 [that] determines if the requested content exists in local cache." Middleton, [0080]. The content request handling operation 500 performs different operations depending on the determination made at operation 504 (i.e., whether the requested content exists in the local cache). For example, and as I discuss in greater detail below, "[i]f it is determined that the requested content exists in local cache, another query operation 506 determines if the content in local cache is stale." *Id.* On the other hand, "if, in query operation 504 it is determined that the requested content is not in the local cache, the operation 504 branches to retrieving operation 518." *Id.*, [0083]. And as I additionally described above in §IX.A.1.c (Ground 2A, 1[a]/8[a]), the operations content request handling operation 500 are performed *by the node*.

188. Referring to the annotated Figure 5 that I have provided below, the operations of content request handling operation 500 that are performed when the operation 504 (highlighted blue) determines that the requested content exists in the local cache (the "YES" branch in Figure 5, highlighted green) are performed *when*

it is determined that said content is cached locally and after determining that the content is cached locally. And, as I described above, the determination (that said content is cached locally) is made by the node, as additionally recited by 1[b].

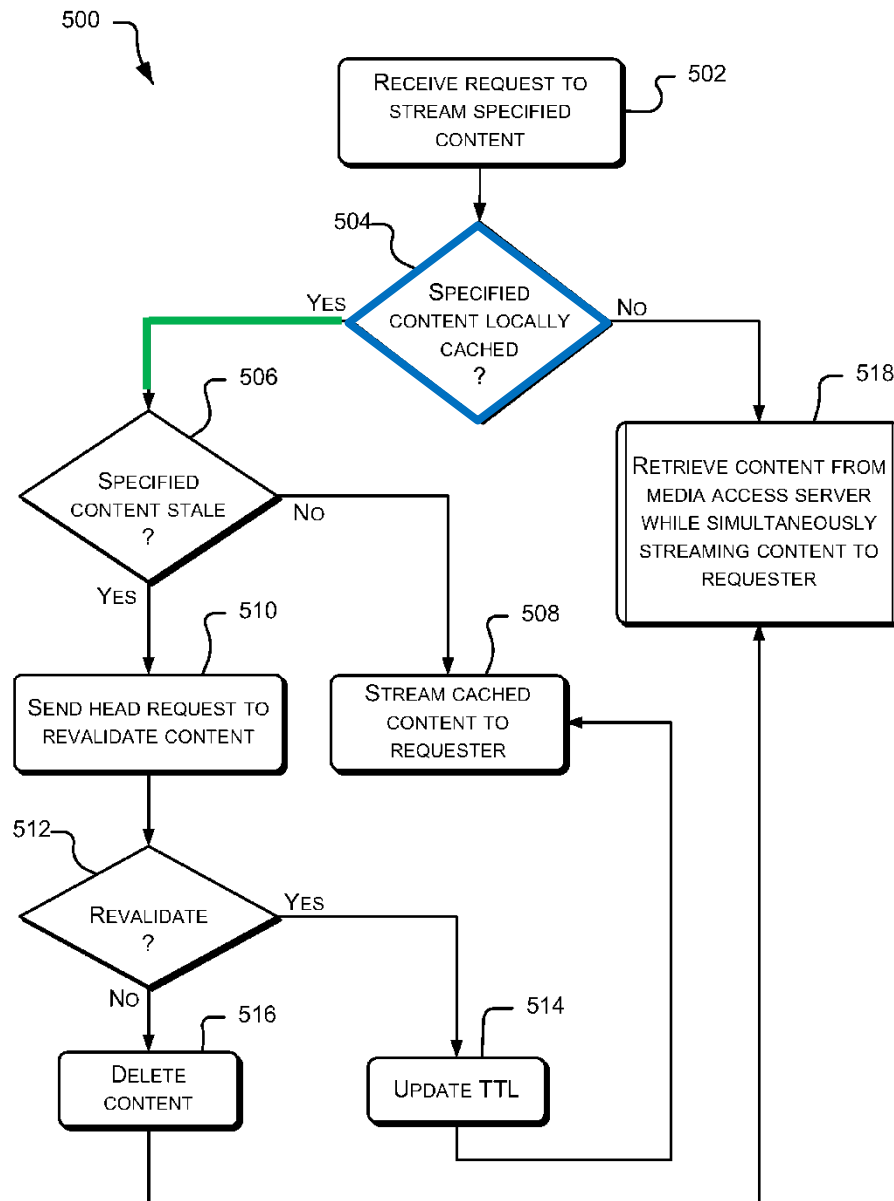


Fig. 5

- e. **1[b][i]/8[b][i]: (B)(1) determining, by the node, a current cache policy associated with the content, wherein determining the current cache policy comprises determining a time period for expiration of content in the cache; and then**

189. In my opinion, Middleton discloses or suggests these limitations.

190. Specifically, Middleton discloses a revalidation flow that attempts to revalidate stale locally cached content, and if successful, obtains an updated TTL value for the requested content (*a current cache policy associated with the content*).

191. As I described above in §IX.A.1.d (Ground 2A, 1[b]/8[b]), if query operation 504 (of Middleton's content request handling operation 500) determines that the requested content exists in the local cache, "another query operation 506 determines if the content in local cache is stale." Middleton, [0080]. Middleton discloses that the "query operation 506 compares the age of the locally cached content to a TTL value associated with the content, and if the age is greater than the TTL value, the content is stale; otherwise the content is not stale." *Id.* In other words, the staleness determination at operation 506 is made based on comparing a TTL value associated with the content to the age of the locally cached content. Since the TTL value associated with the content is used to determine if the content is stale (i.e., whether the locally cached content is too old because it has exceeded its associated "time-to-live" value), the TTL value associated with the content *determin[es] a time period for expiration of content in the cache.*

192. As I describe in greater detail below, Middleton's content request handling operation 500 performs different operations depending on the determination made at operation 506 (i.e., whether the locally cached content is stale). That is, Middleton discloses that "[i]f the locally cached content is determined to be not stale, the operation 506 branches "NO" to streaming operation 508. In streamlining operation 508, the locally cached content is streamed to the requester. On the other hand, if the locally cached content is determined to be stale, the operation 506 branches "YES" to sending operation 510." Middleton, [0081]. In other words, whether to provide the locally cached content to the client (*cache policy*) depends on whether the locally cached content is stale. And, as I described above, whether the locally cached content is stale is determined based on the TTL value associated with the content. Since the TTL value is *associated with the content* and is used to determine whether to provide locally cached content to the client (e.g., based on the age of the locally cached content), the content's TTL value is a *cache policy associated with the content*.

193. Middleton disclose that if the edge server (*node*) "determines that requested content is not available (e.g., is either not locally stored or is stale), the edge server computer 108 takes a remedial action to accommodate the request." Middleton, [0034]. In particular, "[i]f the content is locally stored but is stale, the remedial action involves attempting to revalidate the content." *Id.* As I describe in

greater detail below, the edge server (*node*) attempt to revalidate the content discloses or suggests *determining, by the node, a current cache policy associated with the content.*

194. As I described above in §IX.D.1.c (Ground 2A, 1[a]/8[a]), Middleton discloses that the edge server (*node*) communicates with a media access server (MAS). Middleton, [0047], [0048]. And as I described above, when Middleton's content request handling operation 500 determines that the locally cached content is stale (at operation 506), "the operation 506 branches "YES" to sending operation 510." Middleton, [0081]. Middleton then discloses that "[i]n sending operation 510, a HEAD request is sent to a media access server (MAS) to revalidate the locally cached content. In another query operation 512 [the edge server] checks the response from the MAS to determine whether the locally cached content is revalidated." *Id.*, [0082]. The response from the MAS can include, for example, "information indicating whether the TTL of the content in cache 212 is revalidated or not." *Id.*, [0057]. That is, the TTL value of the content in the cache may not be revalidated. *See, e.g.*, Middleton, [0077]. On the other hand, "[t]he MAS may send a message indicating successful revalidation and a new TTL value." Middleton, [0077]. When the MAS indicates a successful revalidation and provides a new TTL value (e.g., so that the locally cached content is no longer considered stale), the edge server

“updates the TTL value associated with the locally cached content.” *Id.*, [0059], [0082].

195. As illustrated in the annotated Figure 5 that I have provided below, the request to revalidate the locally cached content (operation 510, highlighted green) and determination of whether the revalidation was successful (operation 512, highlighted blue) are both performed after the request for content is received (operation 502, highlighted red). Since the response from the MAS indicating how to treat the locally cached content (e.g., a revalidation failure, such that the locally cached content remains stale, or a new TTL value upon a revalidation success, such that the locally cached content is no longer treated as stale and can be provided to the requester) is received after the request for content is received, the response is *current* as of when the content request is received.

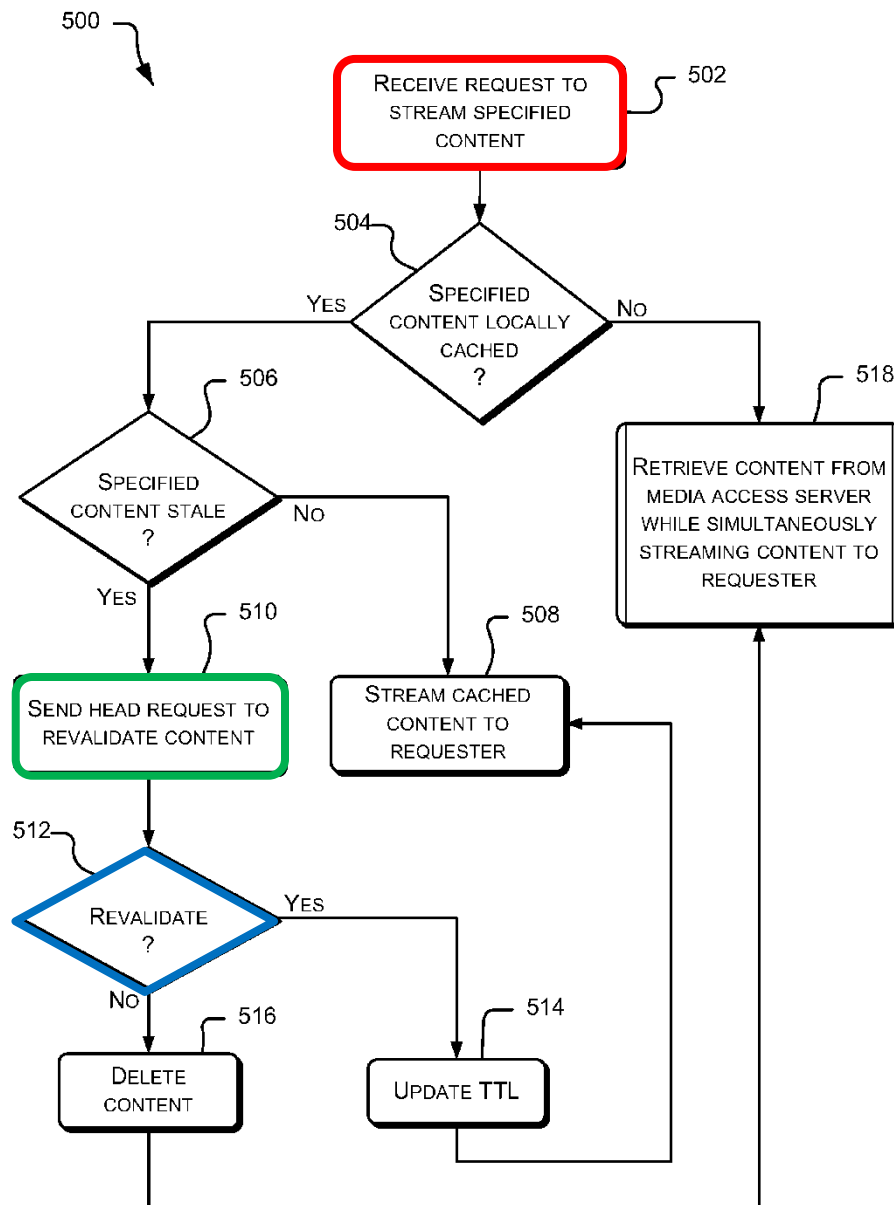


FIG. 5

196. As I described in greater detail below, the edge server (*node*) performs different operations depending on the result of the revalidation attempt. For example, if the MAS response indicates that the revalidation attempt was not successful, the

edge server deletes the locally cached content and retrieves the requested content from the MAS (at operations 516 and 518, respectively). Middleton, [0083]. On the other hand, if the revalidation attempt was successful the edge server updates the TTL value associated with the content (so that the locally cached content is no longer stale) and provides the requested content to the requester (at operations 514 and 508, respectively). *Id.*, [0081], [0082]. Since the edge server (*node*) performs these operations, the information used by the edge server in response to the revalidation attempt (e.g., an indication of revalidation failure or new TTL value) (*current cache policy associated with the content*) is accordingly *determin[ed]*, by the *node*.

- f. 1[b][ii]/8[b][ii]: (B)(2) determining, by the node and based on said current cache policy associated with the content, whether it is acceptable to serve the content that is cached locally;**

197. In my opinion, Middleton discloses or suggests these limitations.

198. Specifically, Middleton discloses checking the response from the MAS, generated in response to a revalidation attempt, to determine whether to provide the locally cached content to the requester (*determining...whether it is acceptable to serve the content that is cached locally*).

199. As I described above in §IX.D.1.e (Ground 2A, 1[b][i]/8[b][i]), Middleton discloses that when an edge server determines that locally cached content is stale (based on the locally cached content's age and a TTL value associated with

the content), the edge server sends a request to a media access server (MAS) to attempt to revalidate the locally cached content. Middleton, [0080]-[0082]. The MAS response indicates whether the content was successfully revalidated (so that the locally cached content is no longer stale) or whether the content was not revalidated. *Id.*, [0082], [0083].

200. The steps I have described above with respect to revalidation are discussed by Middleton in its description of the content request handling operation 500 illustrated in Figure 5. Middleton, [0080]-[0083]. As illustrated in the annotated Figure 5 that I have provided below, the content request handling operation 500 performs different operations depending on the MAS response to the revalidation attempt (at query operation 512, highlighted blue), and whether the MAS response indicates that revalidation was successful (illustrated by the “YES” branch from query operation 512, highlighted green) or that the revalidation was unsuccessful (illustrated by the “NO” branch from query operation 512, highlighted red).

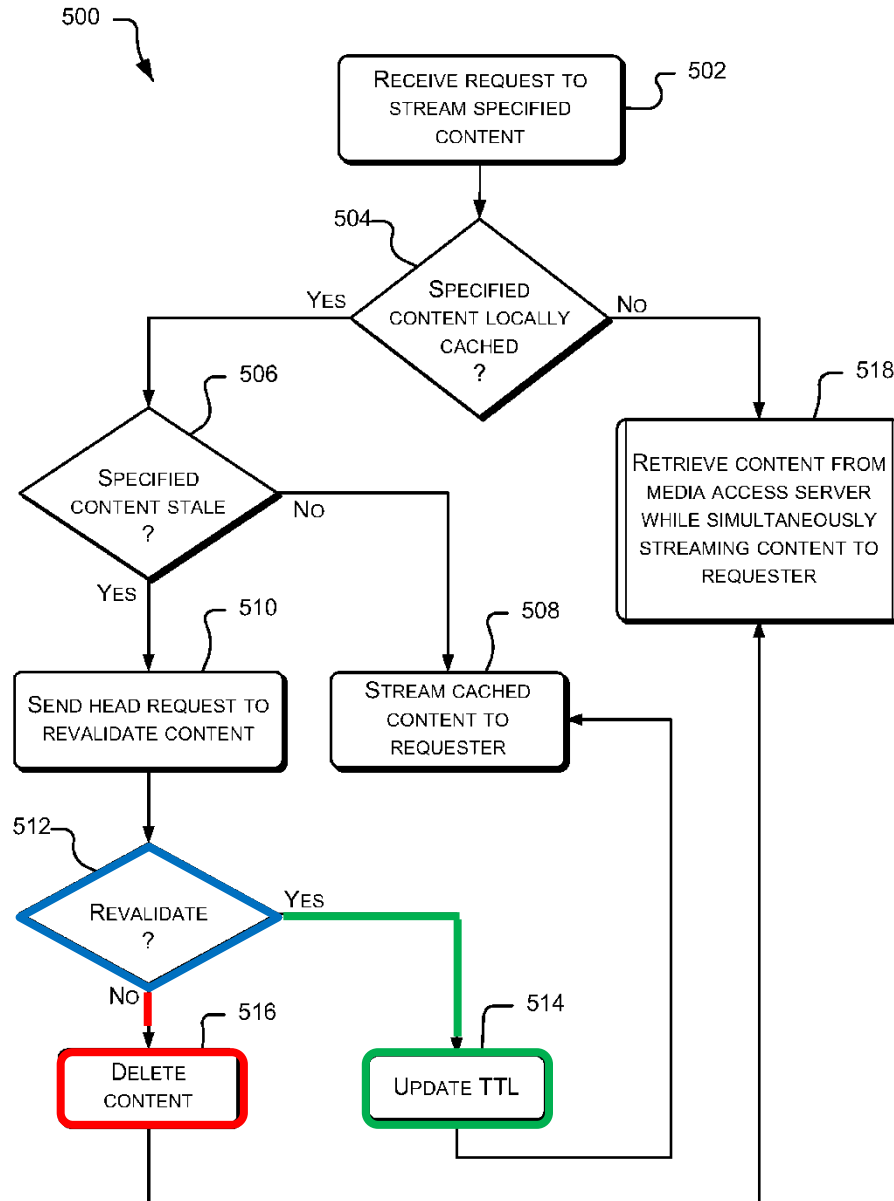


Fig. 5

201. As illustrated in the annotated Figure 5 that I have provided above, when “the response from the MAS indicates that the locally cached content is not revalidated, the operation 512 branches “NO” to deleting operation 516. Deleting

operation 516 deletes the locally cached content.” Middleton, [0083]. On the other hand, and as I describe in greater detail below, “[i]f the content is revalidated, the operation 512 branches “YES” to updating operation 514. Updating operation 514 updates the TTL value associated with the locally cached content, so that the locally cached content is no longer stale.” *Id.*, [0082]. Furthermore, and as I described above in §IX.D.1.e (Ground 2A, 1[b][i]/8[b][i]), the edge server uses the MAS response to the revalidation attempt to determine how to handle the locally cached content in response to the content request (e.g., whether to delete the locally cached content or treat it as no longer stale based on a new TTL value from the MAS). Accordingly, the MAS response used by the cache server is a *current cache policy associated with the content*. Furthermore, when the edge server (*node*) determines whether the revalidation attempt was successful (*determining...whether it is acceptable to serve the content that is cached locally*) based on the MAS response to the revalidation attempt, the determination is made *by the node and based on said current cache policy associated with the content*.

g. 1[b][iii]/8[b][iii]: (B)(3) based on said determining in (B)(2),

202. In my opinion, Middleton discloses or suggests these limitations.

203. Specifically, Middleton discloses performing different operations based on whether an attempt to revalidate locally cached content was successful (*based on said determining in (B)(2)*).

204. Both 1[b][iii] and 8[b][iii] recite *based on said determining in (B)(2)*. (B)(2), as recited by these limitations, refers to the limitations that I refer to as 1[b][ii] and 8[b][ii], respectively. As I described above in §IX.D.1.f (Ground 2A, 1[b][ii]/8[b][ii]), the edge server's determination of whether a revalidation attempt was successful discloses 1[b][ii]/8[b][ii].

205. As illustrated in the annotated Figure 5 that I have provided below, the content request handling operation 500 performs different operations depending on the MAS response to the revalidation attempt (at query operation 512, highlighted blue) (i.e., whether the revalidation attempt was successful). Middleton, [0081], [0082]. If the MAS response indicates the revalidation was successful, content request handling operation 500 proceeds to operation 514 (highlighted green). *Id.*, [0082]. On the other hand, if the MAS response indicates the revalidation was not successful, content request handling operation 500 proceeds to operation 516 (highlighted red). *Id.*, [0083].

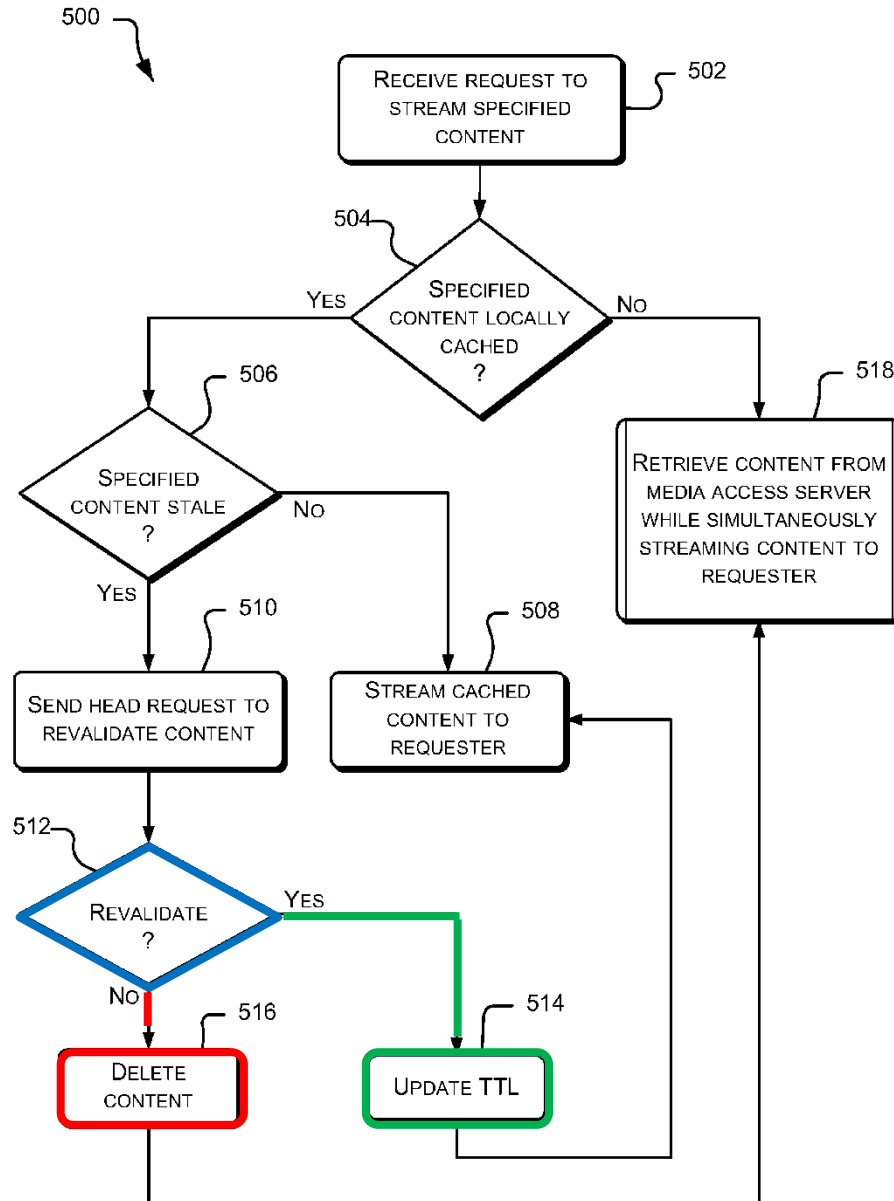


Fig. 5

206. Accordingly, the operations of content request handling operation 500 that are performed after operation 512 (e.g., operation 514 or operation 516) are based on said determining in (B)(2).

- h. 1[b][iii][A]/8[b][iii][A]: (i) when it is not acceptable to serve the content that is cached locally, obtaining, by the node, a new version of the content and then serving, by the node, the new version of the content, and**

207. In my opinion, Middleton discloses or suggests these limitations.

208. Specifically, Middleton discloses that when the revalidation attempt was not successful (*when it is not acceptable to serve the content that is cached locally*), retrieving the requested content from the media access server and providing it to the requester (*obtaining, by the node, a new version of the content and then serving, by the node, the new version of the content*).

209. As I described above in §IX.D.1.f (Ground 2A, 1[b][ii]/8[b][ii]), an edge server (*node*) determines whether it is *acceptable to serve the content that is cached locally* based on whether a revalidation attempt of the locally cached content was successful. When the MAS indicates to the edge server that the locally cached content was not revalidated (*when it is not acceptable to serve the content that is cached locally*), Middleton discloses that “the operation 512 branches “NO” to deleting operation 516. Deleting operation 516 deletes the locally cached content.” Middleton, [0083]. After deleting the locally cached content, “[i]n retrieving operation 518, the requested content is retrieved from the MAS while the content is simultaneously streamed to the requester.” *Id.* For example, Middleton discloses that once portions of the content are received from the MAS and available in the local

cache of the edge server, these cached portions are read from the cache and sent to the requester while the edge server receives other portions of the content. *Id.*, [0063]-[0064]. Accordingly, the edge server receiving new portions of the content from the MAS discloses or suggests *obtaining, by the node, a new version of the content*. Similarly, the edge server reading the portions from the cache to send to the requester discloses or suggests *and then serving, by the node, the new version of the content*.

210. The annotated Figure 5 that I have provided below illustrates the edge server retrieving content from the MAS and streaming the content to the requester at operation 518 (highlighted red) when the edge server determines that the revalidation attempt was unsuccessful at query operation 512 (highlighted blue).

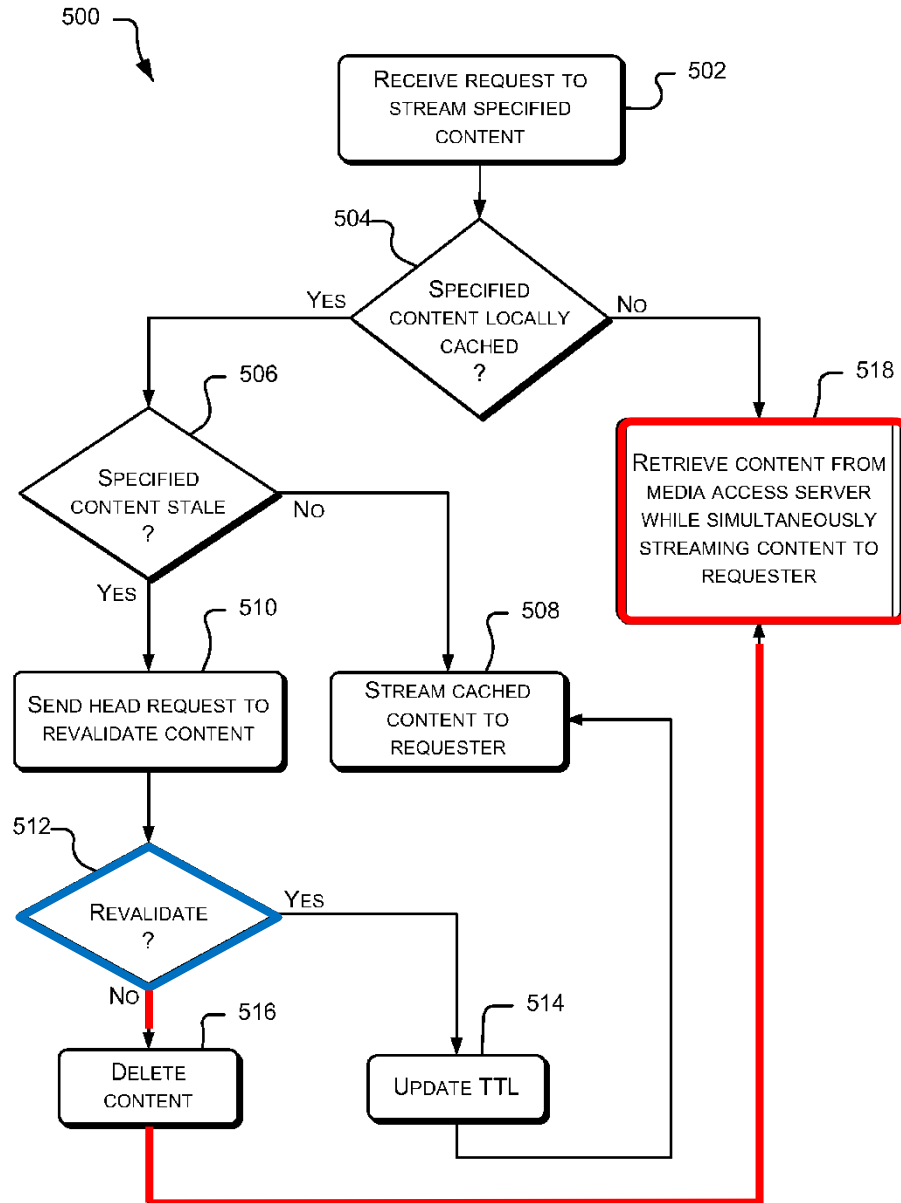


Fig. 5

211. Middleton additionally discloses that its extensible content delivery platform utilizes a “structured event model,” which describes “process[es] for receiving, processing, and servicing requests for content.” Middleton, [0011].

Middleton explains that the structured event model “identifies discrete events within one or more content delivery process[es] at which various actions can be taken through customizable modules or scripts within an extensible content delivery platform.” *Id.*, [0112]. Figure 11 of Middleton, which I have annotated and provided below, illustrates an implementation of the event model. *Id.* As illustrated by the annotated Figure 11, “a content delivery process is initiated when a connection with a user is established at operation 1102,” and then a “request for a content resource is received over the connection in operation 1104.” *Id.*, [0113]. Operation 1102 and operation 1104 are highlighted red in the annotated figure below. If the requested content is not found in the cache (i.e., a cache miss), as represented by operation 1108, “the event model proceeds to operation 1112 at which a Cache Fill request is prepared to request the content resource from another source.” *Id.*, [0113], [0115]. Operation 1108 and operation 1112 are highlighted blue in the annotated figure below. Then, after operations 1114, 1116, 1118 and 1120 are executed and “[w]hen the Cache Fill is complete in operation 1122, the event model 1100 proceeds to operation 1110 indicating that a response to the user's request for content is ready to be sent to the user.” *Id.*, [0115]. “Once the response is ready to be sent to the user at operation 1110 (whether from a Cache Hit in operation 1106 or from a Cache Fill in operations 1112 through 1122), the event model 1100 proceeds to operation 1124 in which the response serving the requested content to the user is initiated.” *Id.*, [0116].

Operation 1122 (cache fill complete), operation 1110 (response ready to send), and operation 1124 (serving the requested content to the user) are highlighted green in the annotated figure below.

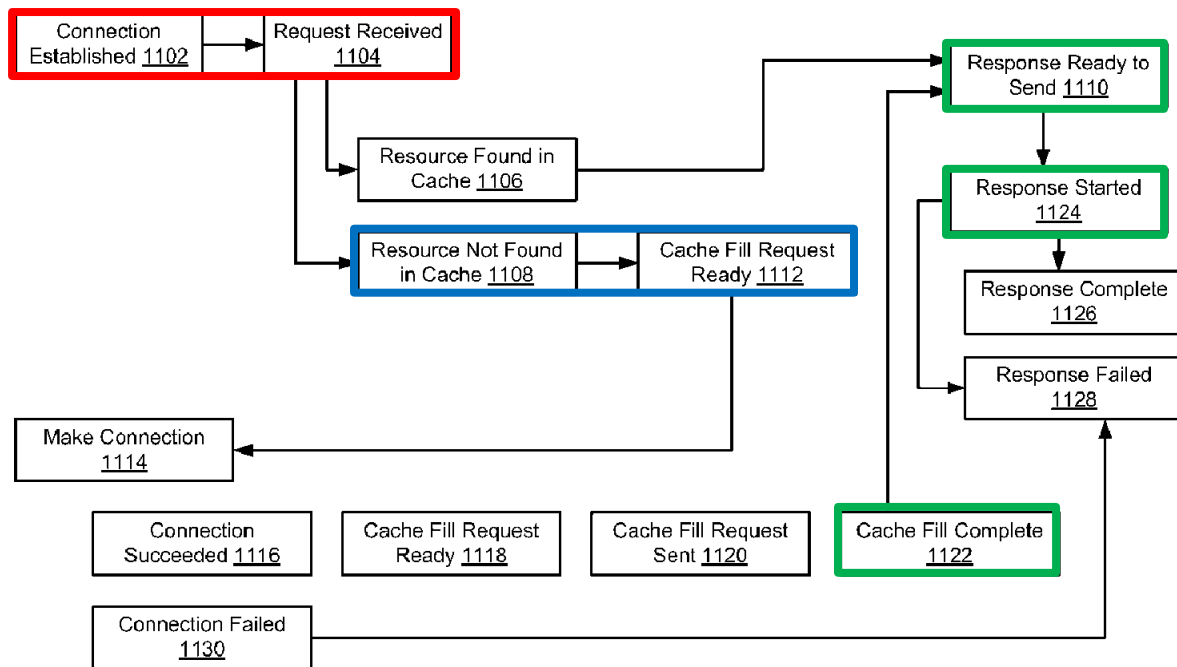


Fig. 11

212. Middleton's event model therefore discloses "discrete events" (Middleton, [0112]) in which requested content is sent to a user (i.e., operations 1110 and 1124) only after the cache fill has completed at a prior event (i.e., operation 1122). This model, in which content is retrieved and cached before it is served to a requester, is also known as "store-and-forward." The "store-and-forward" process of the Middleton extensible content delivery platform's event model additionally

discloses or suggests *obtaining, by the node, a new version of the content and then serving, by the node, the new version of the content.*

- i. **1[b][iii][B]/8[b][iii][B]: (ii) when it is acceptable to serve the content that is cached locally, serving, by the node, the content that is cached locally.**

213. In my opinion, Middleton discloses or suggests these limitations.

214. Specifically, Middleton discloses that when the revalidation attempt was successful (*when it is acceptable to serve the content that is cached locally*), providing the locally cached content to the requester (*serving, by the node, the content that is cached locally*).

215. As I described above in §IX.D.1.f (Ground 2A, 1[b][ii]/8[b][ii]), an edge server (*node*) determines whether it is *acceptable to serve the content that is cached locally* based on whether a revalidation attempt of the locally cached content was successful. When the MAS indicates to the edge server that the locally cached content was revalidated (*when it is acceptable to serve the content that is cached locally*), Middleton discloses that “the operation 512 branches “YES” to updating operation 514. Updating operation 514 updates the TTL value associated with the locally cached content, so that the locally cached content is no longer stale. The locally cached content is then streamed in streaming operation 508.” Middleton, [0082].

216. The annotated Figure 5 that I have provided below illustrates the edge server streaming the cached content to the requester at operation 508 (highlighted green) when the edge server determines that the revalidation attempt was successful at query operation 512 (highlighted blue).

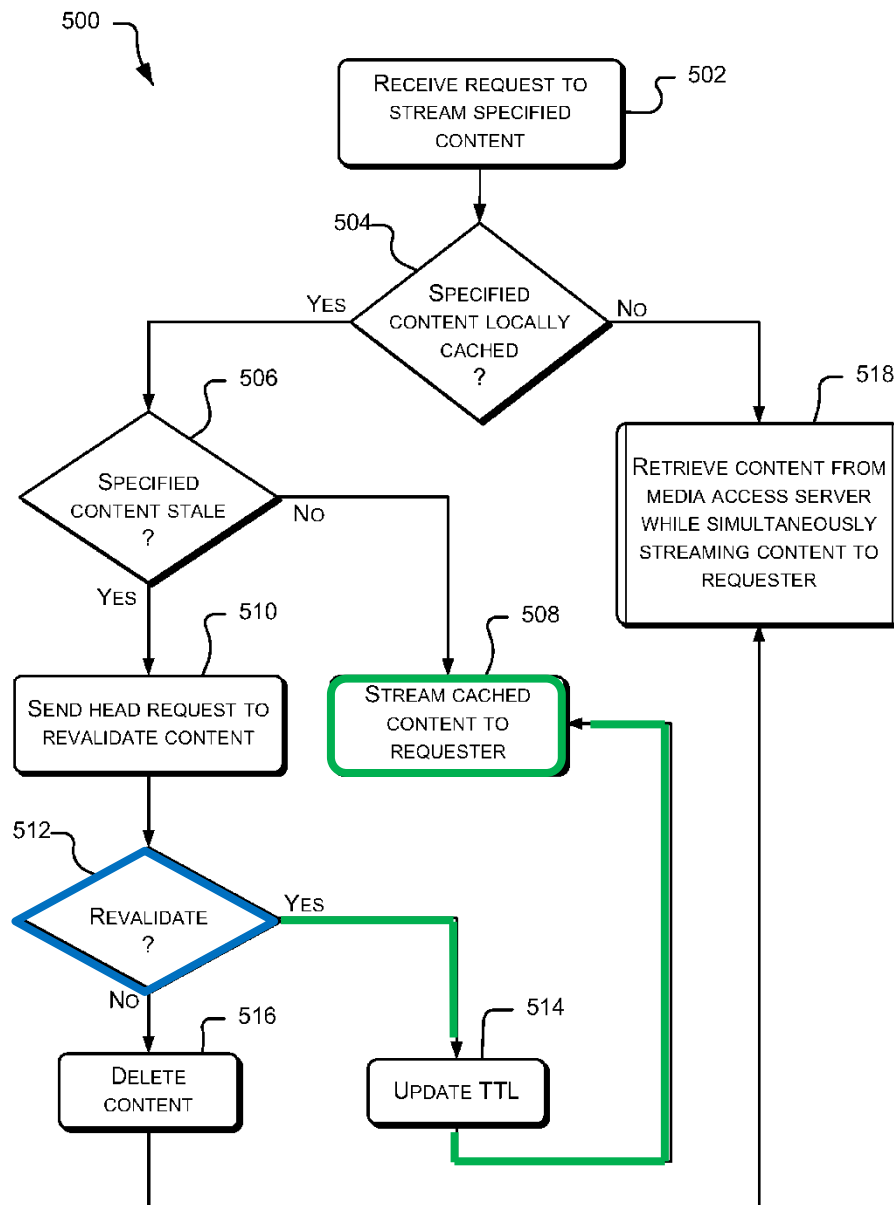


Fig. 5

217. As illustrated by Figure 5, the edge server provides the locally cached content to the requester (*serving, by the node, the content that is cached locally*)

when the edge server determines that the locally cached content was revalidated
(*when it is acceptable to serve the content that is cached locally*).

2. Claims 2, 9

a. 2[pre]: The method as recited in claim 1 further comprising:

218. In my opinion, if the preamble is limiting, Middleton discloses or suggests it.

219. As I described above in §IX.A.1.c (Ground 2A, 1[a]/8[a]), Figures 5-7 of Middleton “are flow charts illustrating processes for handling a request to deliver content. As described below, the extensible content delivery platform is supported in the processes.” Middleton, [0079]. Figure 5, in particular, illustrates a “content request handling operation 500.” *Id.*, [0080]. Middleton’s content request handling operation 500 discloses or suggests *the method as recited in claim 1*.

b. 9[pre]: The computer program product as recited in claim 8 wherein the method further comprises:

220. In my opinion, if the preamble is limiting, Middleton discloses or suggests it.

221. As I described above in §IX.D.1.b (Ground 2A, 8[pre]), Middleton discloses a computer system 1200 (*computer program product*) that includes a “[m]ass storage 1207 [that] can be used to store information and instructions.” Middleton, [0123]-[0125]. The computer system 1200 provides “functionality associated with extensible content delivery application 1250-1 and extensible

content delivery process 1250-2 [that] is discussed above with reference to FIGS. 1-11.” *Id.*, [0131]. Accordingly, the computer system 1200 may perform the content request handling operation 500 disclosed by Middleton and illustrated in Figure 5. Further, the computer system 1200 is the *computer program product as recited in claim 8*, and Middleton’s content request handling operation 500 discloses or suggests what *the method further comprises*.

- c. 2[a]/9[a]: (C) when it is determined that said content is not cached locally, then (C)(1) obtaining the content and then (C)(2) serving the content obtained in (C)(1).**

222. In my opinion, Middleton discloses or suggests these limitations.

223. Specifically, Middleton discloses that when the requested content is not locally cached (*when it is determined that said content is not cached locally*), retrieving the requested content from the media access server and providing it to the requester ((C)(1) *obtaining the content and then (C)(2) serving the content obtained in (C)(1)*).

224. As I described above in §IX.A.1.c (Ground 2A, 1[a]/8[a]), Middleton discloses “[a] query operation 504 [that] determines if the requested content exists in local cache.” Middleton, [0080]. Middleton further discloses that “if, in query operation 504 it is determined that the requested content is not in the local cache” ((C) *when it is determined that said content is not cached locally*), “the operation 504 branches to retrieving operation 518.” *Id.*, [0083]. Then, “[i]n retrieving

operation 518, the requested content is retrieved from the MAS while the content is simultaneously streamed to the requester” ((C)(1) *obtaining the content and then (C)(2) serving the content obtained in (C)(1)*). *Id.*

225. The annotated Figure 5 that I have provided below illustrates the edge server retrieving content from the MAS and streaming the content to the requester at operation 518 (highlighted red) when the edge server determines that the requested content is not in the local cache at operation 504 (highlighted blue).

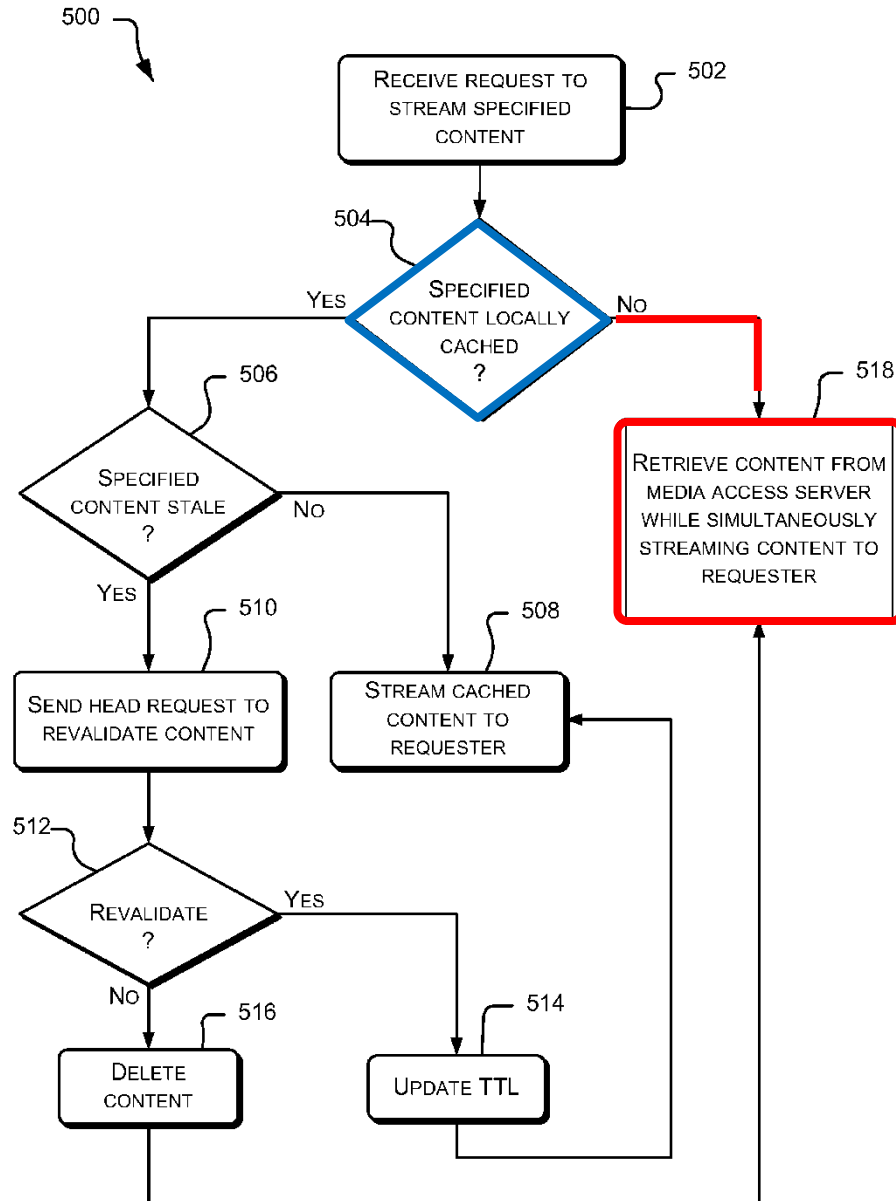


Fig. 5

226. Middleton additionally discloses that its extensible content delivery platform utilizes a “structured event model,” which describes “process[es] for receiving, processing, and servicing requests for content.” Middleton, [0011].

Middleton explains that the structured event model “identifies discrete events within one or more content delivery process[es] at which various actions can be taken through customizable modules or scripts within an extensible content delivery platform.” *Id.*, [0112]. Figure 11 of Middleton, which I have annotated and provided below, illustrates an implementation of the event model. *Id.* As illustrated by the annotated Figure 11, “a content delivery process is initiated when a connection with a user is established at operation 1102,” and then a “request for a content resource is received over the connection in operation 1104.” *Id.*, [0113]. Operation 1102 and operation 1104 are highlighted red in the annotated figure below. If the requested content is not found in the cache (i.e., a cache miss), as represented by operation 1108, “the event model proceeds to operation 1112 at which a Cache Fill request is prepared to request the content resource from another source.” *Id.*, [0113], [0115]. Operation 1108 and operation 1112 are highlighted blue in the annotated figure below. Then, after operations 1114, 1116, 1118 and 1120 are executed and “[w]hen the Cache Fill is complete in operation 1122, the event model 1100 proceeds to operation 1110 indicating that a response to the user's request for content is ready to be sent to the user.” *Id.*, [0115]. “Once the response is ready to be sent to the user at operation 1110 (whether from a Cache Hit in operation 1106 or from a Cache Fill in operations 1112 through 1122), the event model 1100 proceeds to operation 1124 in which the response serving the requested content to the user is initiated.” *Id.*, [0116].

Operation 1122 (cache fill complete), operation 1110 (response ready to send), and operation 1124 (serving the requested content to the user) are highlighted green in the annotated figure below.

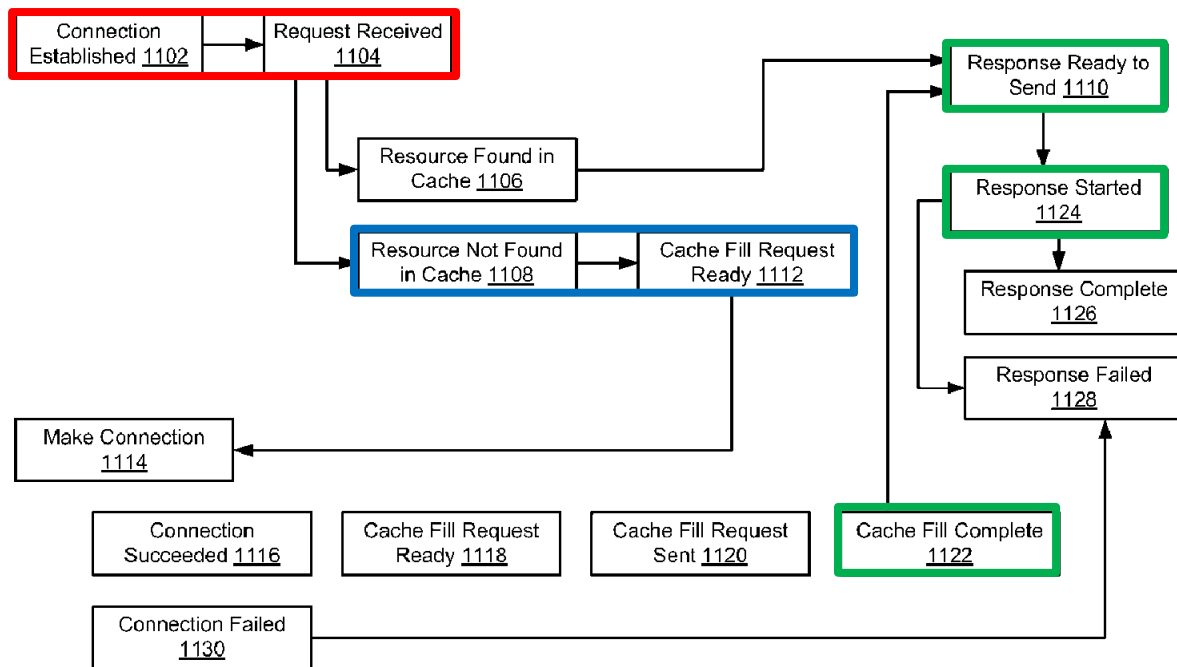


Fig. 11

227. Middleton's event model therefore discloses "discrete events" (Middleton, [0112]) in which requested content is sent to a user (i.e., operations 1110 and 1124) only after the cache fill has completed at a prior event (i.e., operation 1122). This model, in which content is retrieved and cached before it is served to a requester, is also known as "store-and-forward." The "store-and-forward" process of the Middleton extensible content delivery platform's event model additionally

discloses or suggests (C)(1) *obtaining the content and then (C)(2) serving the content obtained in (C)(1).*

- 3. Claims 4, 11: [The method as recited in claim 1]/[The computer program product as recited in claim 8] wherein the current cache policy is determined from a location distinct from the node.**

228. In my opinion, Middleton discloses or suggests these limitations.

229. As I described above in §IX.D.1.e (Ground 2A, 1[b][i]/8[b][i]), Middleton discloses the edge server (*node*) sending a request “to a media access server (MAS) to revalidate the locally cached content.” Middleton, [0082]; *see also id.*, [0054]. And as I described above in §IX.D.1.e (Ground 2A, 1[b][i]/8[b][i]), the result of the revalidation attempt discloses or suggests determining a *current cache policy*.

230. Middleton additionally discloses that the media access server may be a “content distribution server computer 110” or a “content origin server 112.” Middleton, [0040]. Both the content distribution server computer 110 and the content origin server 112 are *distinct from* the edge server 108 (*node*). *See, e.g., id.*, Fig. 1. Accordingly, Middleton discloses or suggests that the *current cache policy is determined from a location distinct from the node*.

- 4. Claim 5, 12: [The method as recited in claim 4]/[The computer program product as recited in claim 11] wherein the location is a database.**

231. In my opinion, Middleton discloses or suggests these limitations.

232. Middleton discloses “a content database 222 accessible to the MAS 204.” Middleton, [0055]. Figure 2 of Middleton, which I have annotated and provided below, illustrates the content database 222 (highlighted blue) that is part of the media access server (MAS) 204 (highlighted red).

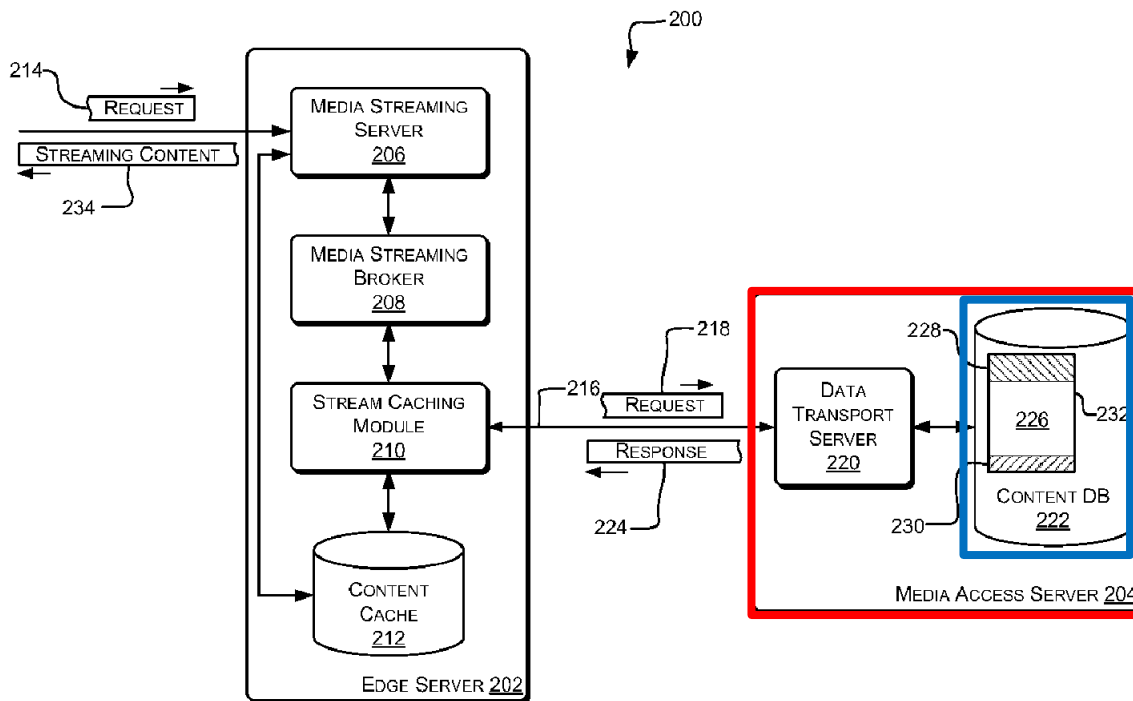


FIG. 2

233. Information in the content database is used by the MAS when responding to requests from the edge server. *See, e.g.*, Middleton, [0055]-[0057]. For example, Middleton discloses that “[i]f the request 218 to the MAS 204 is a HEAD request, and the content is found in the database 222, the response 224 typically includes an HTTP 200 code. The response 224 to a HEAD request also includes information indicating whether the TTL of the content in cache 212 is

revalidated or not.” *Id.*, [0057]. As I described above in §IX.D.1.e (Ground 2A, 1[b][i]/8[b][i]), the edge user sends a HEAD request to the MAS when attempting to revalidate the locally cached content. In other words, the MAS response to the revalidation attempt is based on information found in the content database 222, such as TTL information. Accordingly, the *location* based on which the MAS responds to a revalidation request *is a database*.

5. Claims 6, 13: The method as recited in claim 1]/[The computer program product as recited in claim 8] wherein the cache policy relates to caching content within the CDN.

234. In my opinion, Middleton discloses or suggests these limitations.

235. As I described above in §IX.D.1.e (Ground 2A, 1[b][i]/8[b][i]) and §IX.D.1.f (Ground 2A, 1[b][ii]/8[b][ii]), Middleton discloses the edge server “updat[ing] the TTL value associated with the locally cached content, so that the locally cached content is no longer stale[,]” based on the MAS response to the revalidation attempt (and new TTL value therein) (*cache policy*). Middleton, [0082]; *see also id.*, [0077]. Middleton further discloses that the edge server, which is *within the CDN*, uses the new TTL value to determine whether the locally-cached content is stale. *Id.*, [0080]. For example, the updated TTL value would be used the next time the edge server receives a request for the same content. Accordingly, the TTL value *relates to caching content within the CDN*.

E. Ground 2B: Obviousness over Middleton and MS-MSSO

1. Motivation to Combine Middleton and MS-MSSO

236. In my opinion, a POSITA would have been motivated to modify the content delivery system of Middleton to utilize the HTTP Streaming Protocol taught by MS-MSSO.

237. As explained above, Middleton identified the problem that while “[c]ache servers can be feature rich,” their features were often “complex, sometimes interdependent, and often inconsistent in style and configuration.” Middleton, [0006]. Middleton explained that “[f]eatures of traditional cache servers have been historically challenging to describe, document, and train users.” Middleton, [0006]. In other words, Middleton identified that prior art solutions suffered from the shortcoming of not allowing users to adequately access existing cache server features to specify how content should be cached.

238. To address this perceived problem, Middleton taught an “extensible content delivery platform” in which “[r]ich and well-formed metadata and ‘metacode’ can be associated with content and/or requests in a consistent and programmatic manner.” Middleton, [0007]. Middleton’s “extensible content delivery platform also provides for flexible implementation (e.g., enriched extensibility, customization, and application integration potential) without core code

changes,” and is provided by an edge server with an “event-based application programming interface (API).” Middleton, [0010], [0035].

239. Middleton described that its cache server may be a Windows Media Server, a Microsoft product that was commercially available at the time. Middleton described this implementation as follows: “[t]he edge computer server 108 is configured with media streaming server software, such as...Windows Media ServerTM (WMS). As such, if the requested content is found to be locally stored on the edge computer server 108 and the cached content is not stale, the media streaming software can stream the requested content to the requester.” Middleton, [0033]. Middleton further explained that during operation, a request “is initially received by the media streaming server. The media streaming server 206 could be a...Windows Media Server (WMS).... The media streaming server 206 is configured to communicate data with a content requester using a data streaming protocol.” *Id.*, [0048].

240. A POSITA, aware of the benefits of Middleton’s extensible content delivery platform, which could be used with existing cache servers including the Windows Media Server, would have looked to MS-MSSO for further details about how the Windows Media Server operated and what protocols and functions were supported. And a POSITA would have further recognized that the Windows Media HTTP Streaming Protocol, used by Windows Media Servers, provides additional

advantages when, for example, “the multimedia data needs to pass through a proxy or a firewall.” MS-MSSO, 47. For these reasons, a POSITA, aware of the benefits both of an extensible content delivery platform and of optimizing data transmission in various network environments, including those involving proxies and firewalls, would have been motivated to further improve upon Middleton (extensible content delivery platform including a Windows Media Server) to utilize the Windows Media HTTP Streaming Protocol, as taught by MS-MSSO. This combination would have involved applying combining prior art elements (MS-MSSO’s Windows Media HTTP Streaming Protocol, and Middleton’s platform incorporating a Windows Media Server) according to known methods (deploying a Windows Media Server as an edge server as taught by Middleton, and utilizing the Windows Media HTTP Protocol as taught my MS-MSSO) to yield predictable results (improved caching and network performance).

241. A POSITA would have had a reasonable expectation of success combining Middleton with MS-MSSO in this manner. Middleton already taught an edge server that could be implemented as a Windows Media Server. Middleton, [0033], [0048]. A POSITA would have recognized that the Windows Media Server taught in Middleton could be readily modified to utilize the Windows Media HTTP Streaming Protocol, and the combination would yield predictable results. This combination would have been well within the capabilities of a POSITA because

Middleton describes implementing the invention using a Windows Media Server as its cache server, and MS-MSSO describes the use of various protocols within such a Windows Media Server, including the Windows Media HTTP Streaming Protocol. Additionally, the resulting system's elements would perform functions performed prior to combination—Middleton's platform, utilizing a Windows Media Server, would provide improved access existing to cache server features, and utilizing the Windows Media HTTP Streaming Protocol as described in MS-MSSO would improve network performance.

2. Claims 3, 10: [The method as recited in claim 1]/[The computer program product as recited in claim 8] wherein the request comprises an HTTP request and wherein the content comprises one or more resources.

242. In my opinion, Middleton discloses or suggests these limitations.

243. Middleton discloses that the edge server uses a data transport protocol, such as HTTP, to communicate with other devices of the content delivery network. *See, e.g.*, Middleton, [0044], [0045], [0078]. For example, Middleton discloses that channels 116, between the edge server 108 and other servers of the content delivery network, “are data transport, meaning the channels 116 carry data using a data transport protocol, such as HTTP.” Middleton, [0045]. Similarly, the revalidation of content, which I described above, uses HTTP headers. *Id.*, [0078]. Middleton further discloses that “[i]n one embodiment of a content delivery network servicing HTTP/S

requests, for example, events in the process comprise a Receive Request (e.g., an HTTP request received by the content delivery network).” *Id.*, [0118]. Accordingly, the *request* received by the content delivery network *comprises an HTTP request*.

244. Middleton also discloses that the content provided by the content delivery network “may be any of numerous types of content, including video, audio, images, text, multimedia, or any other type of media.” Middleton, [0029]. The types of media disclosed by Middleton include different types of resources (e.g., audio data, video data, images, etc.). Accordingly, requested content *comprises one or more resources*.

245. To the extent PO argues that Middleton alone does not disclose or suggest these limitations, in my opinion the combination of Middleton and MS-MSSO does.

246. Middleton discloses that the edge server may be “configured with media streaming server software, such as...Windows Media Server™ (WMS).” Middleton, [0033]. And as I described above in §VI.D (MS-MSSO overview), Window Media Servers utilized the Windows Media HTTP Streaming Protocol, which “depends on HTTP for the transfer of all protocol messages, including the transfer of the multimedia data.” MS-MSSO, 47. The MS-MSSO explained that the use of Windows Media HTTP Streaming Protocol was advantageous when, for example, “the multimedia data needs to pass through a proxy or a firewall.” *Id.*

Accordingly, it would have been obvious in the system that combined Middleton and the MS-MSSO for the cache server configured as a Window Media Server (as taught by Middleton) to utilize the Windows Media HTTP Streaming Protocol (as taught by the MS-MSSO), such that requests to the cache server *comprises an HTTP request* for this additional reason.

F. Ground 2C: Obviousness over Middleton and Blumofe

1. Motivation to Combine Middleton and Blumofe

247. In my opinion, A POSITA would have been motivated to modify the content delivery system of Middleton to add an extender as taught by Blumofe.

248. As explained above, Middleton identified the problem that while “[c]ache servers can be feature rich,” their features were often “complex, sometimes interdependent, and often inconsistent in style and configuration.” Middleton, [0006]. Middleton explained that “[f]eatures of traditional cache servers have been historically challenging to describe, document, and train users.” Middleton, [0006]. In other words, Middleton identified that prior art solutions suffered from the shortcoming of not allowing users to adequately access existing cache server features to specify how content should be cached.

249. To address this perceived problem, Middleton taught an “extensible content delivery platform” in which “[r]ich and well-formed metadata and ‘metacode’ can be associated with content and/or requests in a consistent and

programmatic manner.” Middleton, [0007]. Middleton’s “extensible content delivery platform also provides for flexible implementation (e.g., enriched extensibility, customization, and application integration potential) without core code changes,” and is provided by an edge server with an “event-based application programming interface (API).” Middleton, [0010], [0035].

250. Middleton explained that it was desirable to position cache servers so that they are close to users. See Middleton, [0031] (noting it is advantageous for cached content to be “available in a more geographically or logically proximate location to the end user” because this “could reduce content download time to the user computer 104.”). Middleton also taught that the CDN could “include one or more networks” and “the user may use any of numerous different types of computing devices to access the network 102, including, but not limited to, a laptop computer, a handheld computer, a personal digital assistant (PDA), or a cell phone.” Middleton, [0027]-[0028]. But Middleton was silent as to how the invention could be used with wireless devices such as PDAs or cell phones, or how information could be cached in a location that is close to a user who is using a mobile network such as a 3G network.

251. As described above, Blumofe taught “extend[ing] [a CDN] into one or more mobile or wireline networks that cannot or do not otherwise support fully-managed CDN edge servers.” Blumofe, Abstract. Blumofe explained that

“[b]ecause of the way [these] networks are architected and operated, it is not always possible or practical to deploy fully-managed CDN servers therein (i.e., close to end-users).” Blumofe, [0007]. Blumofe observed that in 3G wireless mobile networks, for example, the gateways used to connect wireless end-users to the internet, where CDN servers could be deployed, could be thousands of miles and hundreds of milliseconds away from the end-users. Blumofe, [0007].

252. To address these perceived problems, Blumofe described a CDN Extender, which could be implemented in network environments that do not support CDN services (such as a wireless mobile network), and operated as a cache beyond the edge of the CDN. Blumofe, [0025], [0027]; see also Blumofe, Fig. 3. Blumofe described that the “Extender” could be deployed on the network “as a passive web caching proxy that is beyond the edge of the CDN but that serves CDN-provisioned content under the control of the CDN.” Blumofe, [0009]. The CDN controlled the Extender, and caching policies were supplied by the CDN to the Extender, i.e., caching policies within the CDN relate to caching content outside of the CDN. Blumofe, [0036] (“As noted above, the Extender caches and serves the CDN customer content based on information it receives, in terms of content and cache-control instructions, from the CDN edge servers.”). This solution enabled serving of “CDN-provisioned content under the control of the CDN” where it would otherwise

not be “possible or practical to deploy fully-managed CDN servers.” Blumofe, [0007]-[0009].

253. A POSITA, aware of the benefits both of Middleton’s extensible content delivery platform and of providing caching functionality close to end users in various network environments, would have been motivated to further improve upon Middleton (an extensible content delivery platform) to allow for further caching resources within a wireless network. And a POSITA would have further recognized that an extender as taught in Blumofe would provide this benefit to Middleton.

254. Both references were directed to content delivery networks. Middleton, [0002]; Blumofe, [0005]. Both described shortcomings with conventional CDN architectures and were directed to solving these problems by managing caches that are deployed between origin servers and end users. Middleton, [0031], Blumofe, [0009]. Both Middleton and Blumofe described methods for the content owner to specify the methods used for caching content. Middleton, [0010], [0035], Blumofe, [0030].

255. An extender as taught in Blumofe was designed to work in combination with an existing CDN, such as the CDN of Middleton. For example, Blumofe explains that the Extender “is deployed in the mobile or wireline network, preferably as a passive web caching proxy that is beyond the edge of the CDN but that serves

CDN-provisioned content” and that the Extender “preferably delivers all forms of CDN content (i.e., content that has been off-loaded by content providers for delivery by the CDN service provider, i.e., the entity managing the CDN] over the CDN).” Blumofe, [0009], [0029].

256. A POSITA would have found it obvious to combine Middleton, which described a method of managing cache servers within a CDN (*see generally supra*), with an extender as taught in Blumofe, which described managing caches both within the CDN and outside the CDN (e.g., within the Extender). That is, a POSITA would have found it obvious to utilize an extender as taught in Blumofe with the CDN taught by Middleton. A POSITA would have recognized the benefits this combination provides. For example, using an extender as taught in Blumofe would have allowed Middleton’s extensible content delivery platform to be used in conjunction with other network environments (e.g., wireless networks) where it otherwise could not have been deployed. This combined system would likewise have allowed content to be cached closer to the user, providing improved network performance. Moreover, using an extender as taught in Blumofe in combination with Middleton’s extensible content delivery platform would enable cache policies to be enforced consistently both inside and outside the CDN.

257. This combination would have involved applying a known technique (caching content in wireless networks) to improve similar devices (CDNs) in the

same way (deploying caches in other network environments and/or closer to the user). Further, this would have involved combining prior art elements (an extender as taught in Blumofe and Middleton's CDN) according to known methods (providing cache servers between origin servers and end users) to yield predictable results (improved network performance).

258. A POSITA would have had a reasonable expectation of success combining Middleton with an extender as taught in Blumofe in this manner. Middleton already taught a cache server that could be used in conjunction with various networks, and that could be used to deliver content to wireless devices such as PDAs and cell phones. Middleton, [0027]-[0028]. A POSITA would have recognized that an extender as taught in Blumofe could be readily incorporated into Middleton's system, and the combination would yield predictable results. This combination would be well within the capabilities of a POSITA because an extender as taught in Blumofe was already designed to work with an existing CDN. Blumofe, [0009] ("[t]he CDN is extended into one or more mobile or wireline networks that cannot or do not otherwise support fully-managed CDN edge servers . . . [the] 'Extender' is deployed in the mobile or wireline network, preferably as a passive web caching proxy that is beyond the edge of the CDN but that serves CDN-provisioned content."); Blumofe, [0029] (the Extender "preferably delivers all forms of CDN content (i.e., content that has been off-loaded by content providers for

delivery by the CDNSP over the CDN).” Middleton and Blumofe describe similar CDN environments and communications between origin servers, caches, and clients, such that a POSITA would have found it straightforward to modify Middleton’s CDN to implement an extender as taught in Blumofe. Additionally, the resulting system’s elements would perform functions performed prior to combination—Middleton’s CDN would allow users to access cache server features to specify how content should be cached, and the extender would enable content to be cached in in wireless networks and/or closer to the user.

259. A POSITA would have understood that in such a combination, the CDN would send the cache control information to the extender so that the extender would perform according to the CDN customer’s rules, and the cache policy that configures the cache server within the CDN (e.g., with respect to whether cached content has expired or is an acceptable version) would also be used to configure the extender outside the CDN. Blumofe, [0030] (“To deliver CDN content, the Extender acts under the control of the CDN. This control is necessary because, typically, it is desired (by the CDN customer) that the correct content be delivered according to customer metadata rules”), [0036] (“the Extender caches and serves the CDN customer content based on information it receives, in terms of content and instructions, from the CDN edge servers”), [0037] (“When the CDN edge server responds to an Extender, preferably it communicates some or all available cache

.control information in one or more HTTP response headers. This cache control information includes, for example, time-to-live (TTL) expiration times...”). The combination would have been well within the capabilities of a POSITA because it would have merely required modifying the CDN of Middleton to include an extender as taught by Blumofe.

2. Claims 7, 14: [The method as recited in claim 6]/[The computer program product as recited in claim 13] wherein the cache policy also relates to caching content outside the CDN.

260. In my opinion, the combination of Middleton and Blumofe discloses or suggests these limitations.

261. As I described above in §VI.C (Blumofe overview), Blumofe discloses a CDN Extender that operates as a cache beyond the edge of a CDN. *See, e.g.*, Blumofe, [0027], [0050]. Accordingly, in the combined system of Middleton and Blumofe, the extender is *outside the CDN*, and any content cached by the extender in the combined system is *cached content outside the CDN*.

262. As I additionally described above in §VI.C (Blumofe overview), Blumofe discloses that when the extender receives an end-user request for content that is not in its cache or is present in the cache but stale, the extender sends a GET request to the CDN to fetch content. Blumofe, [0033]. Similarly, Blumofe discloses that if the requested content is already in the extender’s cache, the extender sends an if-modified-since (IMS) request to an edge server of the CDN. Blumofe, [0033]. In

response to the requests, the CDN edge server provides cache control information, such as TTL expiration times. Blumofe, [0036], [0037].

263. The cache control information disclosed by Blumofe, and returned by the CDN, controls the expiration time of the cached content on the extender of the combined Middleton and Blumofe system. Accordingly, the cache control information *relates to caching content outside the CDN*.

264. Additionally, Blumofe describes that the CDN sends the cache control information to the extender so that the extender would perform according to the CDN customer's rules. Blumofe, [0030], [0036]. In the combined system of Middleton and Blumofe, it would have been obvious that the *cache policy* that configures the edge server within the CDN (e.g., with respect to whether a revalidation attempt was successful, and if so, the new TTL value for the content) would have also been used to configure the extender outside the CDN. That is, after the extender of the combined system sends a GET or IMS request to the edge server (which it may do for the reasons I described above), it would have been obvious that when the edge server performs a revalidation attempt (as taught by Middleton) to utilize the TTL value obtained via the revalidation attempt when responding to the extender with cache control information (as taught by Blumofe). Otherwise (e.g., if the edge server did not provide the updated TTL value to the extender), the extender would operate differently than the edge server (e.g., the edge server would have one TTL value for

a piece of content, and the extender another). Having different TTL values at the edge server (that performed the revalidation attempt) and the extender (that sent the GET or IMS request to the edge server) would be undesirable because it could result in serving stale content or incorrectly treating content as stale, and would contradict Blumofe's teaching that "the Extender acts under the control of the CDN" so that "the correct content [is] delivered according to customer metadata rules." Blumofe, [0030], *see also id.*, [0054] ("Preferably, an Extender serves a response from its cache if the CDN edge server to which it is directed would have served the same response to the client."). Accordingly, the *cache policy* (as I described it above in §IX.D.1.e (Ground 2A, 1[b][i]/8[b][i])) *also relates to caching content outside the CDN*.

X. SUMMARY AND ADDITIONAL REMARKS

265. It is my opinion that Grounds 1 and 2 each renders the Challenged Claims unpatentable as obvious, as I explained in the paragraphs preceding.

266. I currently hold the opinions expressed in this declaration based on the information available to me at this time. I may, however, form additional or new opinions should additional information or insights be made available, or upon further consideration.