US010154074B1

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

(10) **Patent No.: US 10,154,074 B1**
(45) **Date of Patent: *Dec. 11, 2018**

(54) **REMEDIATION OF THE IMPACT OF DETECTED SYNCHRONIZED DATA REQUESTS IN A CONTENT DELIVERY NETWORK**

(71) Applicant: **Conviva Inc.**, Foster City, CA (US)

(72) Inventors: **Ion Stoica**, Piedmont, CA (US); **Hui Zhang**, Pittsburgh, PA (US); **Aditya Ravikumar Ganjam**, San Francisco, CA (US)

(73) Assignee: **Conviva Inc.**, Foster City, CA (US)

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

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **14/987,299**

(22) Filed: **Jan. 4, 2016**

**Related U.S. Application Data**

(63) Continuation of application No. 13/181,391, filed on Jul. 12, 2011, now Pat. No. 9,264,780, which is a continuation-in-part of application No. 12/780,790, filed on May 14, 2010, now Pat. No. 8,489,923, which is a continuation-in-part of application No.
(Continued)

(51) **Int. Cl.**

| | |
|---|---|
| ***H04L 29/02*** | (2006.01) |
| ***H04L 29/06*** | (2006.01) |
| ***H04L 29/08*** | (2006.01) |
| ***H04L 12/26*** | (2006.01) |

(52) **U.S. Cl.**
CPC ..... ***H04L 65/4084*** (2013.01); ***H04L 43/04*** (2013.01); ***H04L 67/1008*** (2013.01); ***H04L 67/16*** (2013.01); *H04L 67/1002* (2013.01)

(58) **Field of Classification Search**
CPC ..... G06F 11/3433; G06F 11/3438; G06F 11/3452; H04N 21/2343; H04N 21/234381; H04N 21/231; H04L 67/1002; H04L 67/1029; H04L 67/1031; H04L 67/1036; H04L 47/125; H04L 47/2425; H04L 47/2433; H04L 63/10; H04L 63/102
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,987,621 A 11/1999 Duso
6,026,077 A 2/2000 Iwata
(Continued)

OTHER PUBLICATIONS

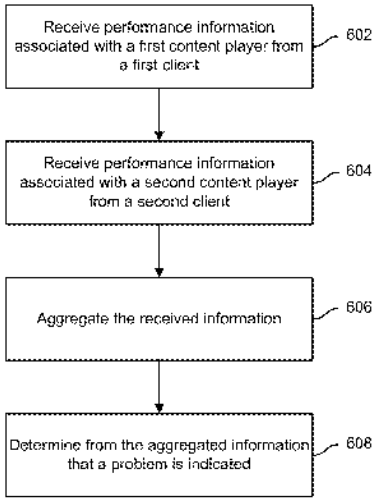
"Relay Nodes in Wireless Sensor Networks: A Survey"—Ataul Bari, University of Windsor, Nov. 2005, http://richard.myweb.cs.uwindsor.ca/cs510/survey_bari.pdf.
(Continued)

*Primary Examiner* Joseph O Schell
(74) *Attorney, Agent, or Firm* — Van Pelt, Yi & James LLP

(57) **ABSTRACT**

Managing synchronized data requests is disclosed. Examples of synchronized data requests include viewer-induced synchronization, failure-induced synchronization, and player-induced synchronization. Information indicative of a potential occurrence of an amount of synchronized requests for audiovisual content that has a potential to have a detrimental impact on one or more components within a content delivery network is obtained. Subsequent to obtaining the information, one or more remediation actions are automatically caused to occur.

**35 Claims, 11 Drawing Sheets**



**Related U.S. Application Data**

11/897,957, filed on Aug. 30, 2007, now Pat. No. 9,407,494.

(60) Provisional application No. 60/859,428, filed on Nov. 15, 2006, provisional application No. 61/227,066, filed on Mar. 10, 2010, provisional application No. 61/363,580, filed on Jul. 12, 2010.

(56) **References Cited**

U.S. PATENT DOCUMENTS


| | | | | |
|---|---|---|---|---|
| 6,279,039 | B1 * | 8/2001 | Bhat | H04L 69/329 709/226 |
| 6,377,996 | B1 | 4/2002 | Lumelsky | |
| 6,438,592 | B1 | 8/2002 | Killian | |
| 6,470,389 | B1 | 10/2002 | Chung | |
| 6,711,622 | B1 | 3/2004 | Fuller | |
| 6,836,691 | B1 | 12/2004 | Stirton | |
| 6,892,236 | B1 | 5/2005 | Conrad | |
| 6,892,307 | B1 | 5/2005 | Wood | |
| 6,906,743 | B1 | 6/2005 | Maurer | |
| 7,006,666 | B2 | 2/2006 | Montgomery | |
| 7,020,082 | B2 | 3/2006 | Bhagavath | |
| 7,024,452 | B1 | 4/2006 | O'Connell, Jr. | |
| 7,024,468 | B1 | 4/2006 | Meyer | |
| 7,039,694 | B2 | 5/2006 | Kampe | |
| 7,139,834 | B1 | 11/2006 | Albanese | |
| 7,159,234 | B1 | 1/2007 | Murphy | |
| 7,222,190 | B2 | 5/2007 | Klinker | |
| 7,356,341 | B2 | 4/2008 | Nanda | |
| 7,367,044 | B2 | 4/2008 | Fowler | |
| 7,373,415 | B1 | 5/2008 | Deshan | |
| 7,490,136 | B2 | 2/2009 | Suzuki | |
| 7,574,488 | B2 | 8/2009 | Matsubara | |
| 7,599,698 | B2 | 10/2009 | Cheng | |
| 7,627,872 | B2 | 12/2009 | Hebeler | |
| 7,668,914 | B2 | 2/2010 | Parker et al. | |
| 7,689,485 | B2 | 3/2010 | Kanekar | |
| 7,904,580 | B2 | 3/2011 | Mandera et al. | |
| 7,941,823 | B2 | 5/2011 | Hasek | |
| 8,135,855 | B2 | 3/2012 | Sitaraman | |
| 8,230,105 | B2 | 7/2012 | Melnyk et al. | |
| 8,370,887 | B2 | 2/2013 | Virdi | |
| 8,387,094 | B1 | 2/2013 | Ho | |
| 8,484,319 | B2 | 7/2013 | Wein | |
| 8,489,923 | B1 | 7/2013 | Lakshminarayanan | |
| 8,589,473 | B2 | 11/2013 | Bruss | |
| 8,677,428 | B2 | 3/2014 | Lewis | |
| 8,683,066 | B2 | 3/2014 | Hurst | |
| 8,751,679 | B2 | 6/2014 | McHugh | |
| 8,874,964 | B1 | 10/2014 | Lakshminarayanan | |
| 8,898,338 | B1 | 11/2014 | McGowan | |
| 8,924,996 | B2 | 12/2014 | Shafiee | |
| 8,930,991 | B2 | 1/2015 | Philpott | |
| 8,943,170 | B2 | 1/2015 | Li | |
| 8,954,491 | B1 | 2/2015 | Medved | |
| 9,100,288 | B1 | 8/2015 | Ganjam | |
| 9,456,015 | B2 | 9/2016 | Chen | |
| 9,549,043 | B1 | 1/2017 | Stoica | |
| 9,613,042 | B1 | 4/2017 | Joseph | |
| 9,819,566 | B1 | 11/2017 | Ganjam | |
| 2002/0002708 | A1 | 1/2002 | Arye | |
| 2002/0082730 | A1 | 6/2002 | Capps et al. | |
| 2002/0095400 | A1 * | 7/2002 | Johnson | H04L 41/147 |
| 2002/0141343 | A1 | 10/2002 | Bays | |
| 2002/0143798 | A1 | 10/2002 | Lisiecki | |
| 2002/0175934 | A1 | 11/2002 | Hand | |
| 2002/0183972 | A1 | 12/2002 | Enck | |
| 2002/0198984 | A1 | 12/2002 | Goldstein et al. | |
| 2003/0046396 | A1 * | 3/2003 | Richter | G06F 9/505 709/226 |
| 2003/0050966 | A1 | 3/2003 | Dutta | |
| 2003/0061305 | A1 | 3/2003 | Copley et al. | |
| 2003/0061356 | A1 | 3/2003 | Jason, Jr. | |
| 2003/0065739 | A1 | 4/2003 | Shnier | |
| 2003/0105850 | A1 | 6/2003 | Lean | |
| 2003/0140180 | A1 | 7/2003 | Brown | |
| 2003/0204613 | A1 | 10/2003 | Hudson | |
| 2004/0010544 | A1 | 1/2004 | Slater | |
| 2004/0019675 | A1 | 1/2004 | Hebeler | |
| 2004/0047354 | A1 * | 3/2004 | Slater | H04L 41/0896 370/400 |
| 2004/0057420 | A1 | 3/2004 | Curcio | |
| 2004/0093155 | A1 | 5/2004 | Simonds | |
| 2004/0107387 | A1 | 6/2004 | Larsson et al. | |
| 2004/0128682 | A1 | 7/2004 | Liga | |
| 2004/0133471 | A1 | 7/2004 | Pisaris-Henderson et al. | |
| 2004/0158643 | A1 | 8/2004 | Suzuki | |
| 2004/0162901 | A1 * | 8/2004 | Mangipudi | H04L 69/329 709/225 |
| 2004/0187159 | A1 | 9/2004 | Gaydos | |
| 2004/0193716 | A1 | 9/2004 | McConnell | |
| 2004/0267691 | A1 | 12/2004 | Vasudeva | |
| 2005/0010915 | A1 | 1/2005 | Chen | |
| 2005/0021715 | A1 | 1/2005 | Dugatkin | |
| 2005/0060158 | A1 | 3/2005 | Endo et al. | |
| 2005/0120131 | A1 * | 6/2005 | Allen | H04L 12/5695 709/233 |
| 2005/0183120 | A1 | 8/2005 | Jain | |
| 2005/0251835 | A1 | 11/2005 | Scott et al. | |
| 2005/0278259 | A1 | 12/2005 | Gunaseelan | |
| 2006/0059248 | A1 | 3/2006 | Ikeda | |
| 2006/0135172 | A1 * | 6/2006 | Dronne | H04L 47/10 455/452.2 |
| 2006/0143350 | A1 | 6/2006 | Miloushev | |
| 2006/0206539 | A1 | 9/2006 | Thompson | |
| 2006/0285489 | A1 | 12/2006 | Francisco | |
| 2007/0025381 | A1 | 2/2007 | Feng | |
| 2007/0041584 | A1 | 2/2007 | O'Connor | |
| 2007/0101202 | A1 * | 5/2007 | Garbow | G06F 11/008 714/47.2 |
| 2007/0136311 | A1 * | 6/2007 | Kasten | H04L 67/1008 |
| 2007/0140113 | A1 * | 6/2007 | Gemelos | H04L 12/5695 370/229 |
| 2007/0198413 | A1 | 8/2007 | Nagao | |
| 2007/0204011 | A1 | 8/2007 | Shaver | |
| 2007/0250560 | A1 | 10/2007 | Wein | |
| 2007/0286351 | A1 | 12/2007 | Ethier et al. | |
| 2007/0288638 | A1 | 12/2007 | Vuong | |
| 2008/0037438 | A1 | 2/2008 | Twiss | |
| 2008/0063195 | A1 | 3/2008 | Li | |
| 2008/0096562 | A1 | 4/2008 | Wu | |
| 2008/0151821 | A1 | 6/2008 | Cho | |
| 2008/0155586 | A1 | 6/2008 | Yang et al. | |
| 2008/0195461 | A1 | 8/2008 | Li | |
| 2008/0215718 | A1 | 9/2008 | Stolorz | |
| 2008/0247326 | A1 | 10/2008 | Cormier | |
| 2008/0263180 | A1 | 10/2008 | Hurst | |
| 2009/0019503 | A1 | 1/2009 | Vorbau | |
| 2009/0059812 | A1 | 3/2009 | Chinnaswamy | |
| 2009/0164656 | A1 | 6/2009 | Guan | |
| 2009/0172200 | A1 | 7/2009 | Morrison et al. | |
| 2009/0187956 | A1 | 7/2009 | Sommer | |
| 2009/0248872 | A1 | 10/2009 | Luzzatti | |
| 2009/0271101 | A1 | 10/2009 | Relyea | |
| 2009/0327489 | A1 | 12/2009 | Swildens | |
| 2009/0328124 | A1 | 12/2009 | Khouzam et al. | |
| 2010/0043014 | A1 | 2/2010 | Hebeler | |
| 2010/0080290 | A1 | 4/2010 | Mehrotra | |
| 2010/0114562 | A1 | 5/2010 | Hutchinson | |
| 2010/0125675 | A1 | 5/2010 | Richardson | |
| 2010/0241701 | A1 | 9/2010 | Lester | |
| 2010/0302002 | A1 | 12/2010 | Guo | |
| 2010/0306368 | A1 | 12/2010 | Gagliardi | |
| 2011/0047413 | A1 | 2/2011 | McGill | |
| 2011/0082946 | A1 | 4/2011 | Gopalakrishnan | |
| 2011/0196943 | A1 | 8/2011 | Bornstein | |
| 2011/0296048 | A1 | 12/2011 | Knox | |
| 2011/0314130 | A1 | 12/2011 | Strasman | |
| 2012/0007866 | A1 | 1/2012 | Tahan | |
| 2012/0047542 | A1 | 2/2012 | Lewis | |
| 2012/0093098 | A1 | 4/2012 | Charbit | |
| 2012/0110167 | A1 | 5/2012 | Joch | |
| 2012/0124179 | A1 | 5/2012 | Cappio | |

(56) **References Cited**

U.S. PATENT DOCUMENTS

| | | | |
|---|---|---|---|
| 2012/0178426 | A1 | 7/2012 | Filipov |
| 2012/0198492 | A1 | 8/2012 | Dhruv |
| 2012/0204068 | A1 | 8/2012 | Ye et al. |
| 2012/0209717 | A1 | 8/2012 | Henry |
| 2012/0226734 | A1 | 9/2012 | Poese |
| 2012/0240176 | A1 | 9/2012 | Ma |
| 2012/0278496 | A1 | 11/2012 | Hsu |
| 2013/0094445 | A1 | 4/2013 | De Foy |
| 2013/0132605 | A1 | 5/2013 | Kocks |
| 2013/0142129 | A1 | 6/2013 | Rinne |
| 2013/0305299 | A1 | 11/2013 | Bergstrom |
| 2014/0108671 | A1 | 4/2014 | Watson |
| 2014/0149557 | A1 | 5/2014 | Lohmar |
| 2014/0150046 | A1 | 5/2014 | Epstein |
| 2014/0198641 | A1 | 7/2014 | Perkuhn |
| 2014/0245359 | A1 | 8/2014 | De Foy |
| 2014/0330980 | A1 | 11/2014 | Richardson |
| 2015/0026239 | A1 | 1/2015 | Hofmann |
| 2015/0095704 | A1 | 4/2015 | Sokolik |
| 2016/0234293 | A1 | 8/2016 | Berger |

OTHER PUBLICATIONS

Elo et al., "Virtual URLs for Browsing & Searching Large Information Spaces", WebNet Journal, pp. 38-43. p. 66, Jan.-Mar. 1999.
"Firefly-Inspired Heartbeat Synchronization in Overlay Networks"—Binci et al., University of Bologna, Italy, Feb. 2007 http://www.cs.unibo.it/babaoglu/courses/cas06-07/papers/pdf/fireflies.pdf.

* cited by examiner

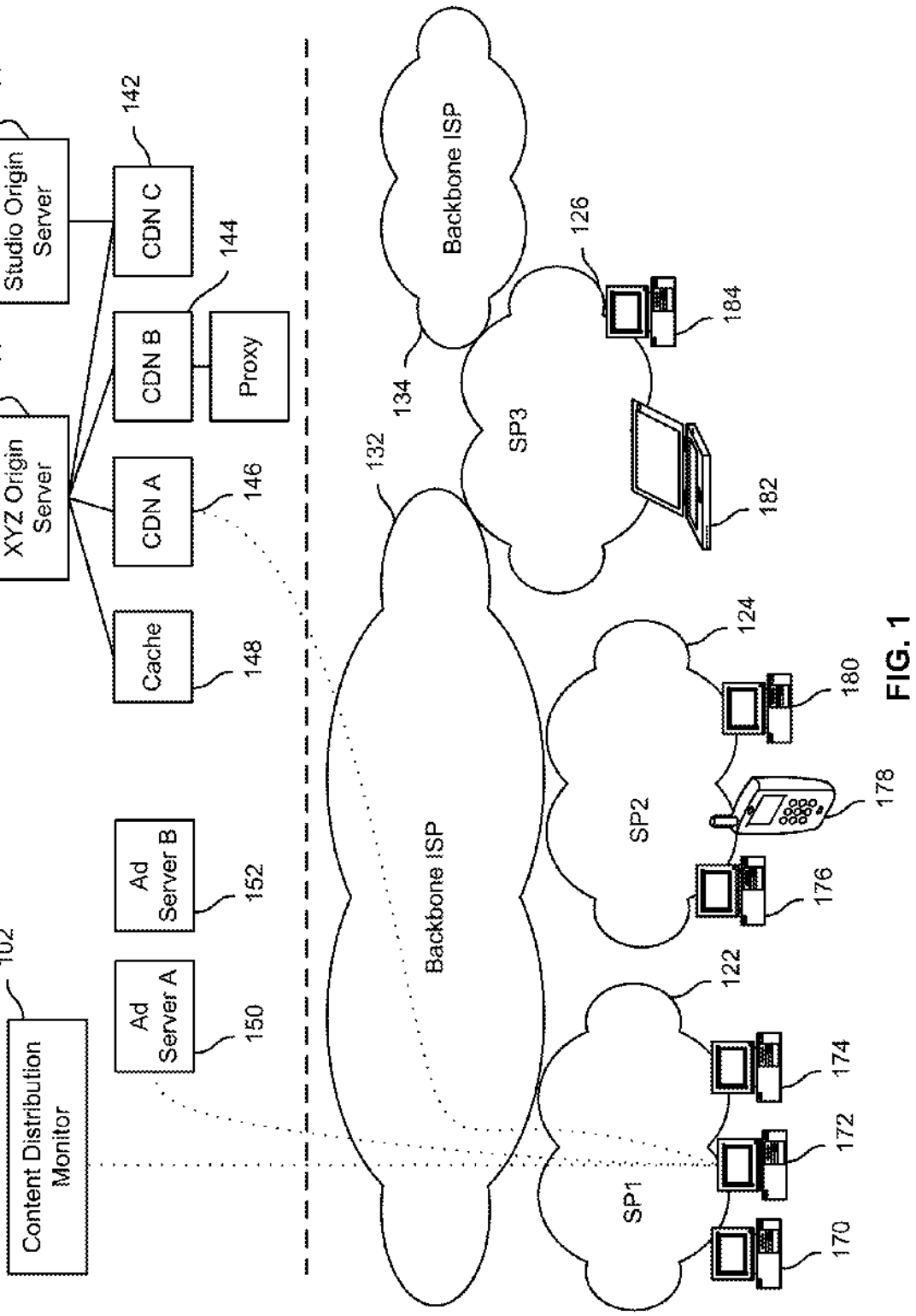

Content Distribution Monitor
102
Ad Server A
150
Ad Server B
152
Cache
148
CDN A
146
CDN B
144
CDN C
142
Proxy
XYZ Origin Server
198
Studio Origin Server
196
Backbone ISP
132
Backbone ISP
134
SP1
122
SP2
124
SP3
126
170
172
174
176
178
180
182
184

FIG. 1

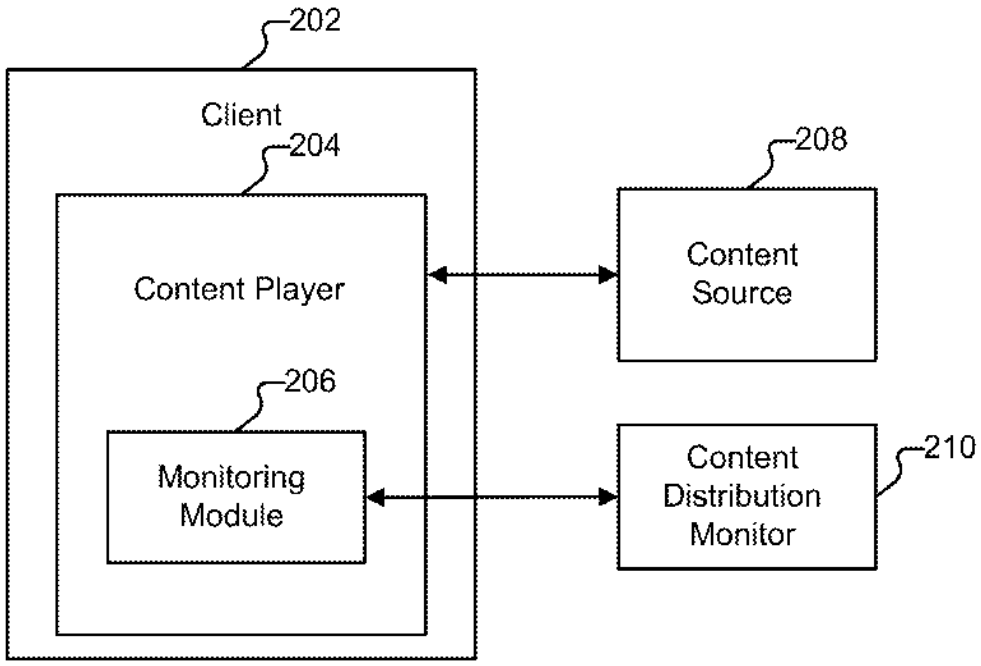

202
Client
204
Content Player
206
Monitoring Module
208
Content Source
210
Content Distribution Monitor

FIG. 2A

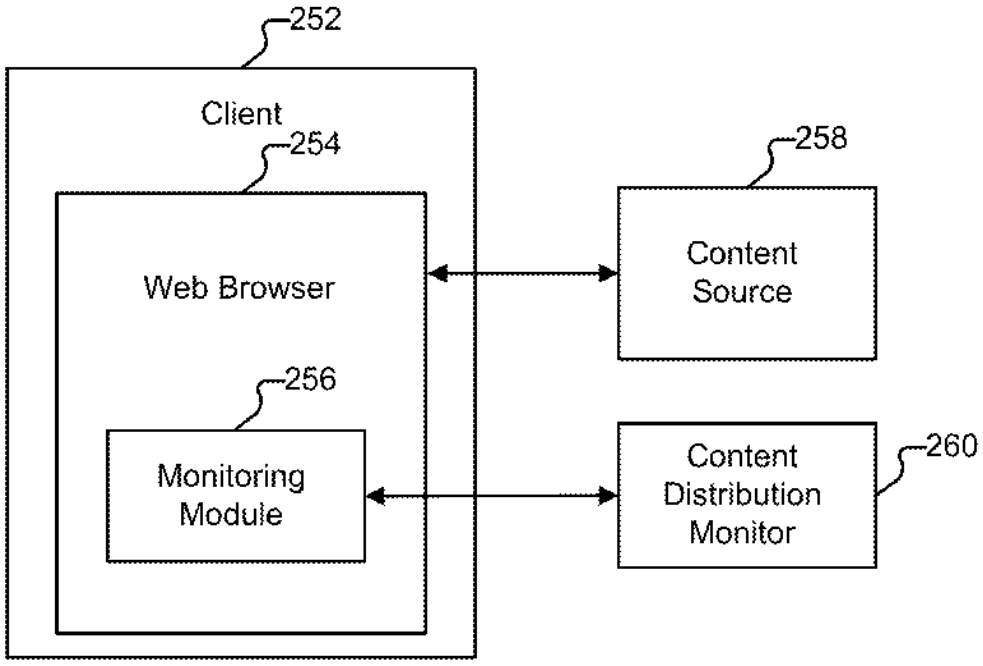

252
Client
254
Web Browser
256
Monitoring Module
258
Content Source
260
Content Distribution Monitor

FIG. 2B

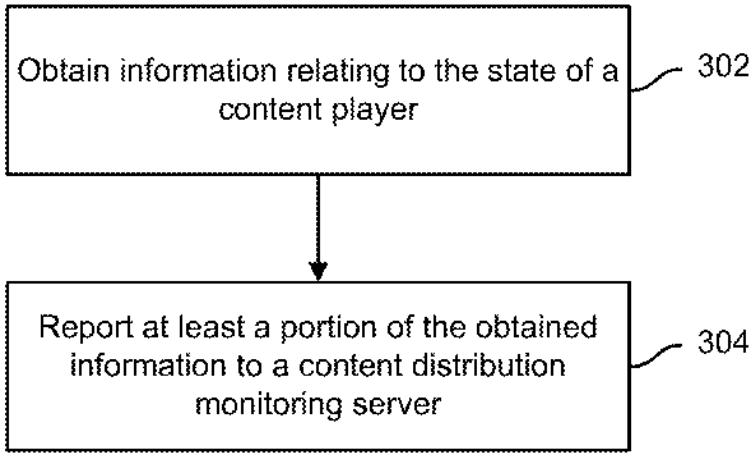

Obtain information relating to the state of a content player
302
Report at least a portion of the obtained information to a content distribution monitoring server
304

FIG. 3

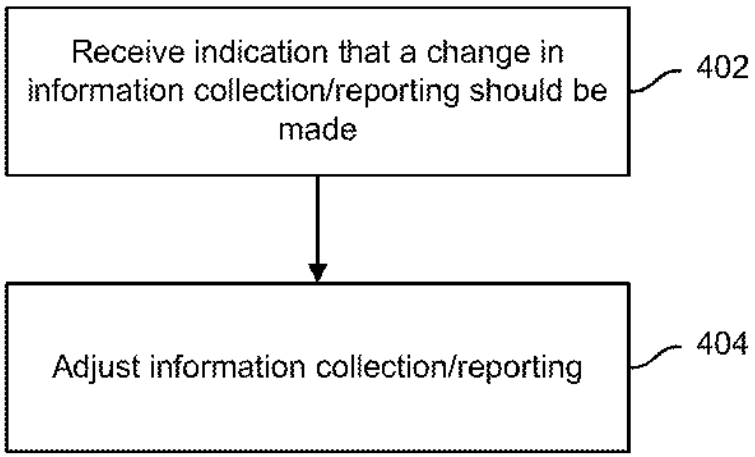

Receive indication that a change in information collection/reporting should be made
402
Adjust information collection/reporting
404

FIG. 4

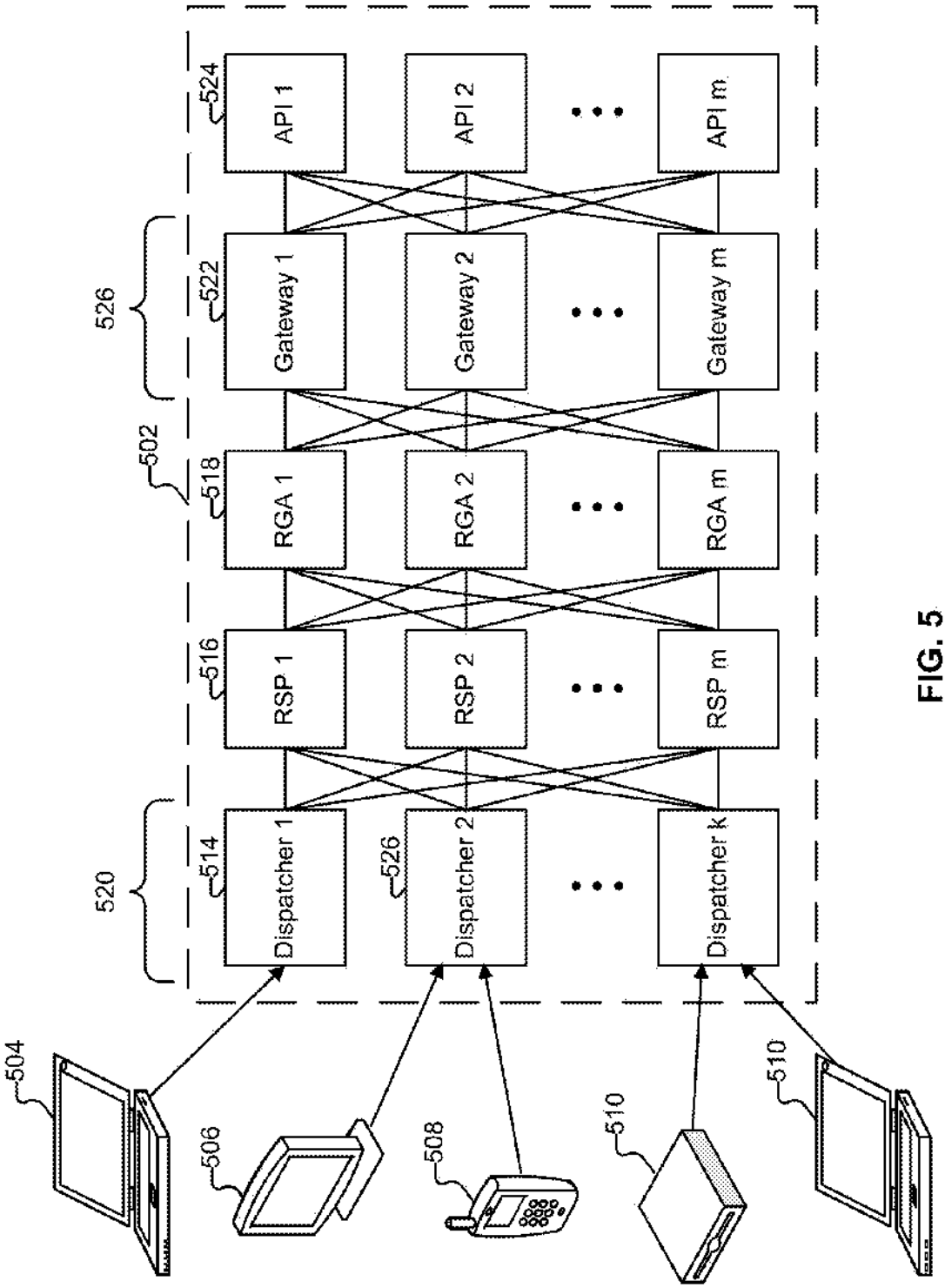

520
526
502
526
524
522
518
516
514
API 1
API 2
API m
Gateway 1
Gateway 2
Gateway m
RGA 1
RGA 2
RGA m
RSP 1
RSP 2
RSP m
Dispatcher 1
Dispatcher 2
Dispatcher k
504
506
508
510
510

FIG. 5

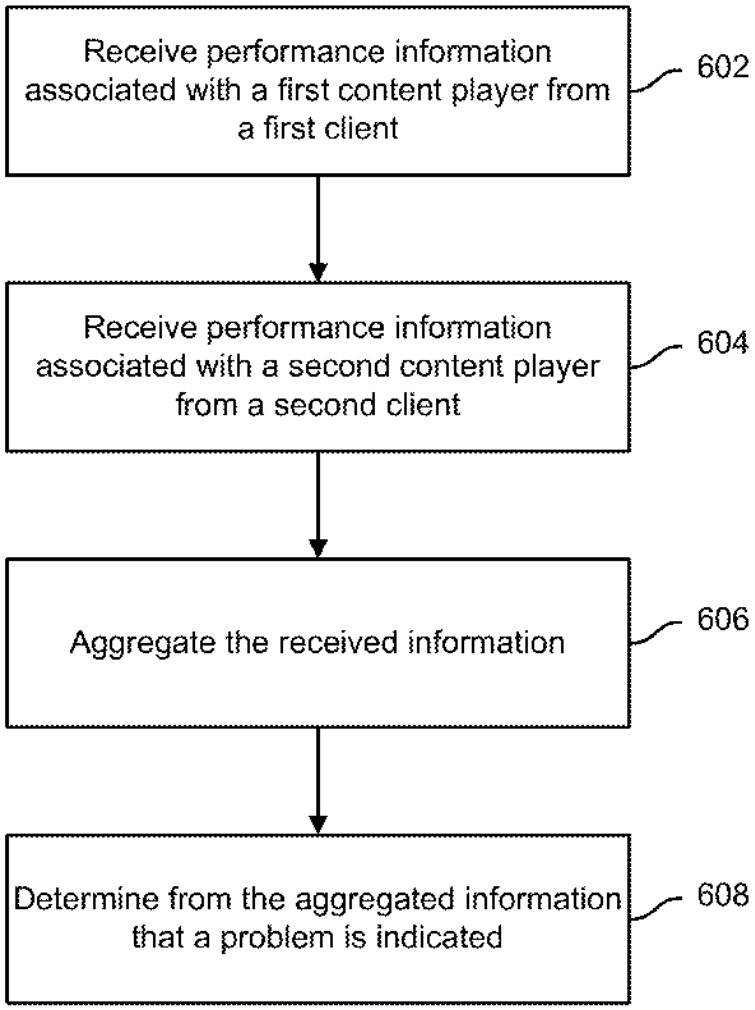

Receive performance information associated with a first content player from a first client
602
Receive performance information associated with a second content player from a second client
604
Aggregate the received information
606
Determine from the aggregated information that a problem is indicated
608

FIG. 6

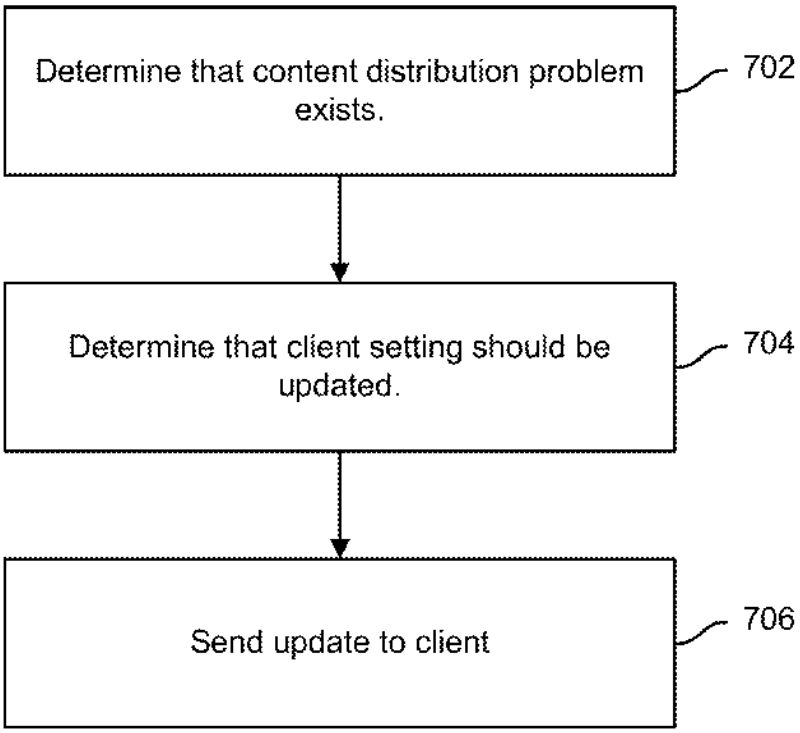

Determine that content distribution problem exists.
702
Determine that client setting should be updated.
704
Send update to client
706

FIG. 7

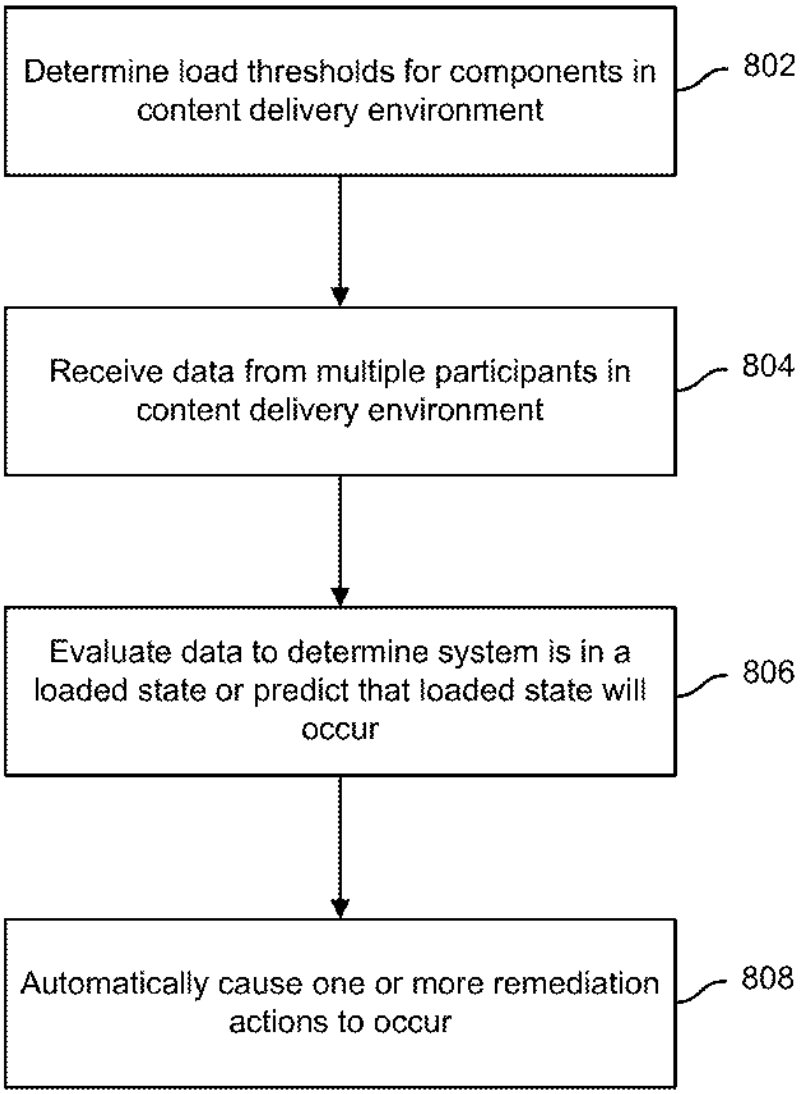

Determine load thresholds for components in content delivery environment
802
Receive data from multiple participants in content delivery environment
804
Evaluate data to determine system is in a loaded state or predict that loaded state will occur
806
Automatically cause one or more remediation actions to occur
808

FIG. 8

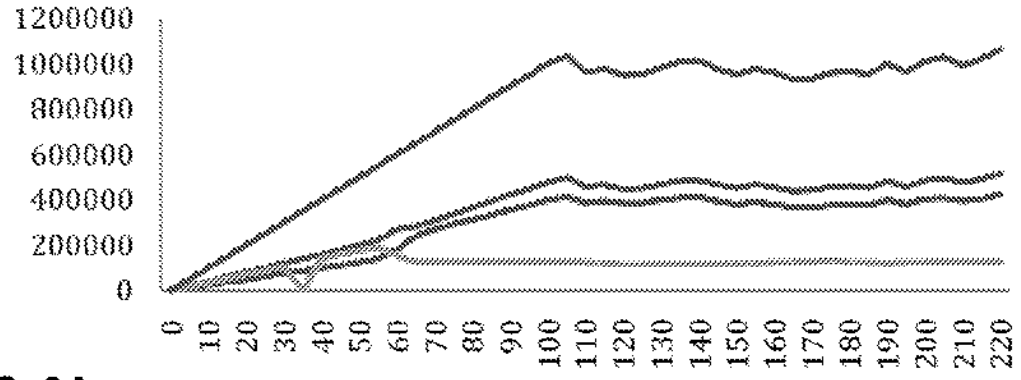
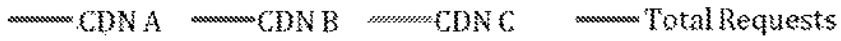



FIG. 9A



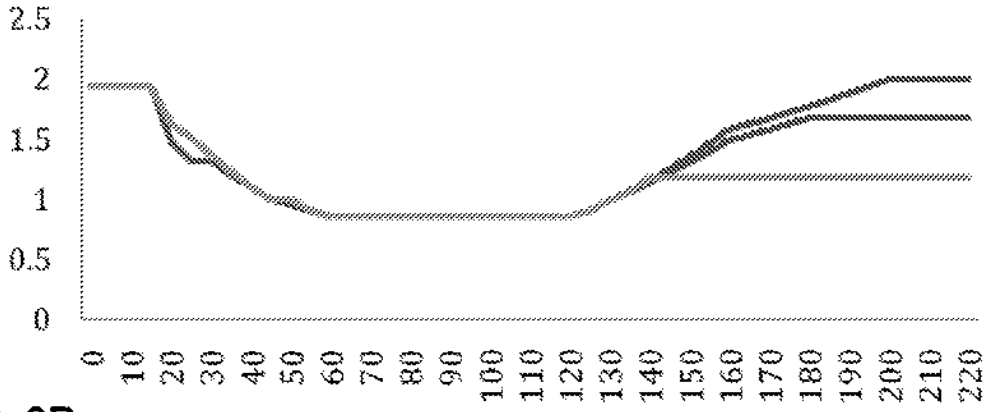




FIG. 9B

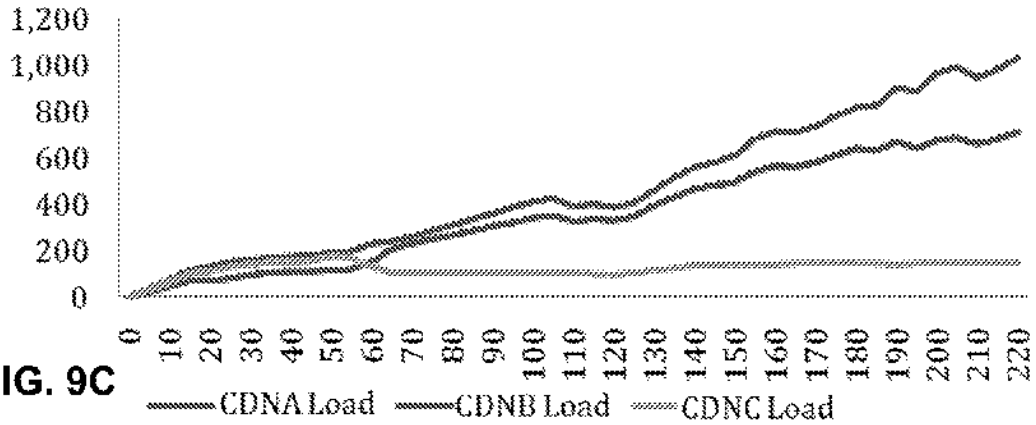
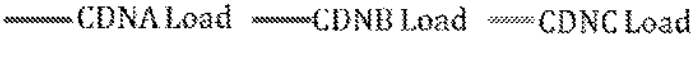




FIG. 9C



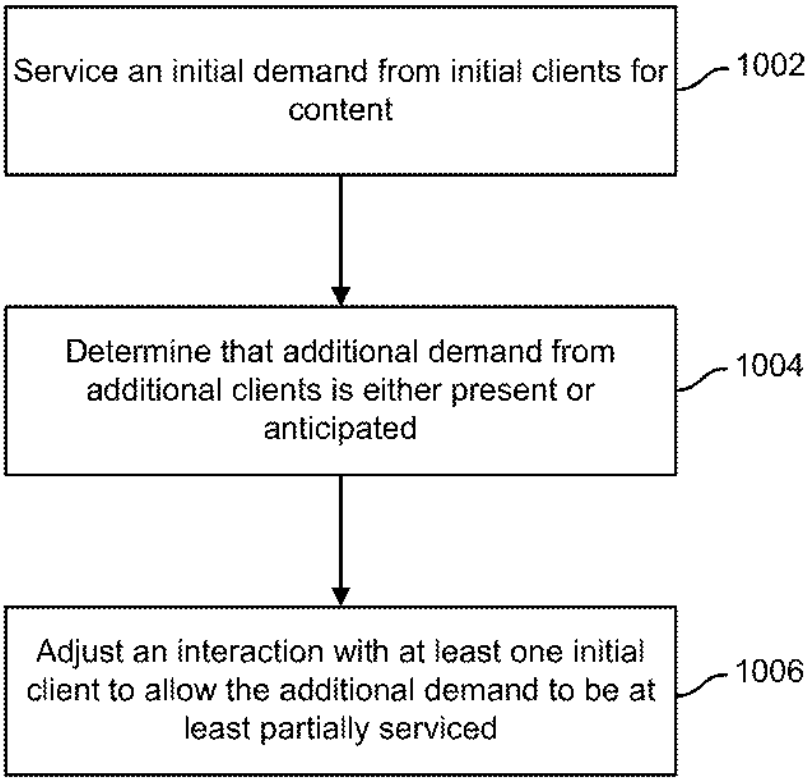

Service an initial demand from initial clients for content
1002
Determine that additional demand from additional clients is either present or anticipated
1004
Adjust an interaction with at least one initial client to allow the additional demand to be at least partially serviced
1006

FIG. 10

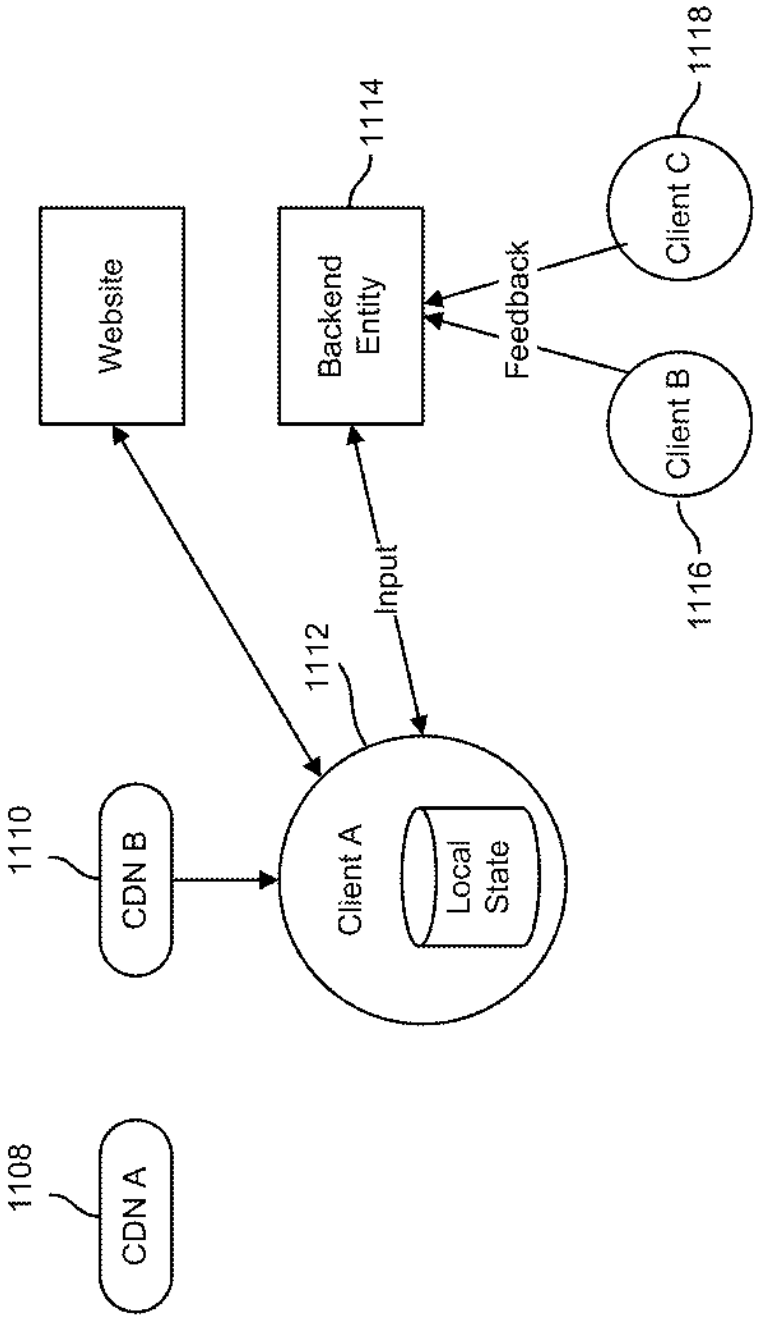

1108
CDN A
1110
CDN B
Client A
Local State
1112
Input
Website
Backend Entity
1114
Feedback
Client B
1116
Client C
1118

FIG. 11

# REMEDIATION OF THE IMPACT OF DETECTED SYNCHRONIZED DATA REQUESTS IN A CONTENT DELIVERY NETWORK

## CROSS REFERENCE TO OTHER APPLICATIONS

This application is a continuation of co-pending U.S. patent application Ser. No. 13/181,391, entitled MANAGING SYNCHRONIZED DATA REQUESTS IN A CONTENT DELIVERY NETWORK filed Jul. 12, 2011, which is a continuation in part of U.S. patent application Ser. No. 12/780,790, now U.S. Pat. No. 8,489,923, entitled DETECTING PROBLEMS IN CONTENT DISTRIBUTION filed May 14, 2010, which is a continuation in part of co-pending U.S. patent application Ser. No. 11/897,957 entitled REASSIGNING SOURCE PEERS filed Aug. 30, 2007, which claims priority to U.S. Provisional Patent Application No. 60/859,428 entitled CONTENT DISTRIBUTION filed Nov. 15, 2006. U.S. patent application Ser. No. 12/780,790 also claims priority to U.S. Provisional Patent Application No. 61/227,066 entitled REAL-TIME TELEMETRY FOR CONTENT filed Jul. 20, 2009 and to U.S. Provisional Patent Application No. 61/339,925 entitled TELEMETRY FOR CONTENT filed Mar. 10, 2010.

U.S. patent application Ser. No. 13/181,391 also claims priority to U.S. Provisional Patent Application No. 61/363,580 entitled AUTOMATED CONTROL OF SYNCHRONIZED REQUESTS ACROSS MULTIPLE CDNS filed Jul. 12, 2010.

All of the aforementioned applications are incorporated herein by reference for all purposes.

## BACKGROUND OF THE INVENTION

Users are increasingly using networks such as the Internet to access content, such as video files and live streaming/video on demand content, via client machines. As demand for such content increases, there are challenges in distributing that content efficiently and with high quality. As one example, certain types of content, such as broadcasts of live events, may receive a large number of concurrent requests for that content. Factors such as the sudden, simultaneous demand for content, and the sheer number of potential consumers, can overwhelm the infrastructure used to provide the content to consumers. Even systems that might otherwise be able to accommodate large groups of viewers may not be able to handle particularly sharp increases in load (such as at the start of a game) gracefully. Improvements in managing synchronized data requests are therefore desirable.

## BRIEF DESCRIPTION OF THE DRAWINGS

Various embodiments of the invention are disclosed in the following detailed description and the accompanying drawings.

FIG. **1** is an illustration of an environment in which content is distributed.

FIG. **2A** illustrates an embodiment of a client.

FIG. **2B** illustrates an embodiment of a client.

FIG. **3** illustrates an example of a process for monitoring the performance of a content player.

FIG. **4** illustrates an example of a process for dynamically adjusting a heartbeat.



FIG. **5** is an illustration of an environment in which status information is received and processed.

FIG. **6** illustrates an example of a process for detecting a problem in a content distribution.

FIG. **7** illustrates an example of a process for correcting a problem in a content distribution.

FIG. **8** illustrates an example of a process for managing synchronized data requests.

FIG. **9A** is a graph that illustrates the number of concurrent viewers over time across three CDNs.

FIG. **9B** is a graph that illustrates the average bitrate for viewers across each of the three CDNs.

FIG. **9C** is a graph that illustrates the total traffic pushed by each of the three CDNs.

FIG. **10** illustrates an example of a process for allocating resources in a content delivery environment.

FIG. **11** illustrates an example of an environment in which content is distributed.

## DETAILED DESCRIPTION

The invention can be implemented in numerous ways, including as a process; an apparatus; a system; a composition of matter; a computer program product embodied on a computer readable storage medium; and/or a processor, such as a processor configured to execute instructions stored on and/or provided by a memory coupled to the processor. In this specification, these implementations, or any other form that the invention may take, may be referred to as techniques. In general, the order of the steps of disclosed processes may be altered within the scope of the invention. Unless stated otherwise, a component such as a processor or a memory described as being configured to perform a task may be implemented as a general component that is temporarily configured to perform the task at a given time or a specific component that is manufactured to perform the task. As used herein, the term 'processor' refers to one or more devices, circuits, and/or processing cores configured to process data, such as computer program instructions.

A detailed description of one or more embodiments of the invention is provided below along with accompanying figures that illustrate the principles of the invention. The invention is described in connection with such embodiments, but the invention is not limited to any embodiment. The scope of the invention is limited only by the claims and the invention encompasses numerous alternatives, modifications and equivalents. Numerous specific details are set forth in the following description in order to provide a thorough understanding of the invention. These details are provided for the purpose of example and the invention may be practiced according to the claims without some or all of these specific details. For the purpose of clarity, technical material that is known in the technical fields related to the invention has not been described in detail so that the invention is not unnecessarily obscured.

FIG. **1** is an illustration of an environment in which content is distributed. In the example shown, clients **170-184** are used to access content, such as audiovisual content (e.g., movies, songs, television shows, sporting events, games, images, etc.) that is owned by content owners. The content is stored (or captured) at origin servers **196-198**, then distributed via other servers, caches, content distribution networks (CDNs), proxies, etc. (collectively, "content sources"). Content sources employ a variety of technologies and include HTTP, Adobe Flash Media, and Microsoft

Internet Information Service servers. In some embodiments content is also distributed by clients (e.g., using peer-to-peer techniques).

Examples of clients include personal computers (**170**), laptops (**182**), cellular phones/personal digital assistants (**178**), and other types of information appliances (not shown) such as set-top boxes, game consoles, broadband routers, file servers, video servers, and digital video recorders, as applicable. The clients shown are used by subscribers to various Internet service providers (ISPs). For example, clients **170**, **172**, and **174** are subscribed to SP1 (**122**), while clients **176**, **178**, and **180** are subscribed to SP2 (**124**), and clients **182** and **184** are subscribed to SP3 (**126**).

In the example shown, a movie studio ("Studio") has contracted with content distributor **142** to provide downloadable copies of its films in exchange for a fee. Similarly, a television network ("XYZ") has contracted with content distributors **142-146** to provide viewers with access to live streams of its broadcasts as well as streams of television show episodes and sporting events. In some cases, the content distributor is owned/operated by the content owner.

Content distributor **142** has a data center that is provided with network access by backbone ISP **132**. Though represented here by a single node (also referred to herein as a "CDN node"), content distributor **142** may typically have multiple data centers (not shown) and may make use of multiple backbone or other ISPs. Content distributor **144** has a data center that is provided with network access by backbone ISP **134**. Advertisements are served to various clients via ad servers **150-152**.

Suppose a user of client **172** (hereinafter "Alice") would like to watch a live soccer game owned by XYZ. Client **172** includes a web browser application. Alice uses the web browser application to navigate to a portal owned by XYZ, such as "http://xyztvnetwork.com/livegames." Her request for the game is directed to a CDN node that is closest to her. In this case, CDN **146** is the fewest hops away from her client. Her client then begins streaming the content from CDN **146**, which is in turn rendered in her browser (e.g., via a Flash or Silverlight player). Advertisements, associated with the portal, are served to her by ad server **150**.

In addition to CDN **146** and ad server **150**, Alice's client is also in communication with content distribution monitor **102**. As will be described in more detail below, client **172** provides status information (also referred to herein as a "heartbeat"), on a recurring basis, to content distribution monitor **102**. The status information includes a variety of telemetry data such as information that captures the quality of the user experience (e.g., video stream quality), and information pertaining to user behavior. Examples of quality metrics include: failed connection attempts, the length of time it takes for the soccer game video to start playing, the number of buffering events (if any), the length of buffering events, the number of frames per second rendered by the video player, and premature connection terminations. Examples of user behavior include: starting and stopping playing a video or audio stream, seeking within the stream, switching the player to full screen mode, minimizing/restoring the player, a change in the volume level of the player, and clicking on an advertisement.

As other users of clients **170-184** request content, their respective players similarly obtain content from content sources such as CDN **144**, communicate status information (also referred to herein as telemetry information) to content distribution monitor **102**, and receive commands. Such players may be browser-based as with Alice's, or they may be standalone applications, as applicable. In various embodiments, all clients in the environment provide status information to content distribution monitor **102**.

As will be described in more detail below, different clients may provide content distribution monitor **102** with different levels of detail, and may also do so with differing frequency. For example, client **178** is a smartphone with less powerful hardware than client **172** and more limited bandwidth. It is configured to provide less information to content distribution monitor **102** than client **172** and also does so less frequently than client **172**.

Content distribution monitor **102** collects and processes the information received from Alice's client along with other clients. The collected information is made available in real-time to control entities/operators and can be used to detect and remedy problems in the content distribution. Examples of such problems include excessive buffering, freezing, and frame skipping. Additional information pertaining to delivery resources (e.g., CDN **142**) and network providers (e.g., ISP **126**) is also made available, as is other information pertaining to clients such as demographic information.

In the example shown in FIG. **1**, a single content distribution monitor **102** is used. Portions of content distribution monitor **102** may be provided by and/or replicated across various other modules or infrastructure depending, for example, on factors such as scalability and availability (reducing the likelihood of having a single point of failure), and the techniques described herein may be adapted accordingly. In some embodiments content distribution monitor **102** is implemented across a set of machines distributed among several data centers. A Resilience Service Layer (RSL) can also be used to ensure that the monitoring service is not disrupted when/if a subset of machines fail or a subset of data centers hosting the content distribution monitor are disconnected from the Internet. In some embodiments entities such as the operator of CDN **142** run their own content distribution monitors **102**. In such cases, the content distribution monitor may provide a subset of the functionality described herein, such as by detecting, predicting, and/or remediating content delivery problems with respect to the servers that comprise the CDN, and without collecting heartbeat information from clients or information related to the servers of other CDNs.

## Examples of Client Architecture

In various embodiments, the collection of status information and the reporting of that information to the content distribution manager are performed by a "monitoring module" included in the client. The monitoring module can also receive commands, such as from content distribution monitor **102**.

FIG. **2A** illustrates an embodiment of a client. In the example shown, client **202** includes a content player application **204** which in turn incorporates monitoring module **206**. Content is received by the player from content source **208**. Monitoring module **206** is in communication with content distribution monitor **210**.

In the example shown, module **206** is implemented in ActionScript and deployed as an SWF library that is dynamically loaded by player **204**. In Flash, the NetStream( ) class is mainly responsible for streaming, buffering, and playing the video. The monitor is implemented as an element in the content player which wraps the video element, and provides the same interface as the video element, as described in more detail below.

Module **206** can also implemented using other approaches, such as in the .NET platform for Silverlight and deployed as a DLL library that is dynamically loaded by player application **204**. In Silverlight, the MediaElement( ) class is mainly responsible for streaming, buffering, and playing the video. The monitor is implemented as an element in the content player which wraps the video element, and provides the same interface as the video element, as described in more detail below.

Examples of some of the information collected by monitoring module **206** include the following, expressed in key-value pairs:

(player_state, "buffering"): The stream is buffering.

(buffer_length, 5 s): The current buffer length is five seconds.

(join_time, 3 s): The join time was 3 sec.

(frame_per_second, 30): The number of frames per second is 30.

(player-mode, "Full Screen"): The player is running in full-screen mode.

FIG. **2B** illustrates an embodiment of a client. In the example shown, client **252** includes a web browser application which in turn incorporates monitoring module **256**. Content is received by the player from content source **258**. Monitoring module **256** is in communication with content distribution monitor **260**. In the example shown, module **256** is implemented in JavaScript. The monitor periodically collects information about the current web page and the browser. Examples of some of the information collected by monitoring module **256** include the following, expressed in key, value pairs:

(browser_minimized, "yes"): The browser window is minimized.

(tab_visible, "no"): The tab containing the web page is not visible.

(pointer_pos, "x, y"): The position of the pointer on the web page.

(banner_display", "ACMECars"): The banner ad on the web page is for ACME Cars.

As explained above, the monitoring module can be implemented in a variety of ways. For example, the monitoring module can be included in the client's player by the author of the player. The monitoring module can also extend the functionality of an existing player, such as by being implemented as a plugin, library, or standalone application (e.g., a helper application). Various examples of techniques for integrating a monitoring module with an existing player will now be described.

In one embodiment, the monitoring module is a wrapper around the lowest level streamer module (e.g., "NetStream" in Flash, or "MediaElement" in Silverlight). The player uses the wrapper, which provides the same or enhanced version of the API provided by the streamer module, to stream video. Logic in the wrapper captures the appropriate telemetry data.

Example. "ConvivaNetStream" extends and wraps "NetStream":

```
var ns:NetStream  new ConvivaNetStream( );
ns.play(<stream>);
```

In a second embodiment, the monitoring module exists as a side attachment and is not present in the code path for streaming to occur. The player passes the streamer module to the monitoring module either directly or through a proxy module. The monitoring module reads properties and listens for events from the streamer module or proxy module to collect data. The proxy module method prevents the monitoring module from interfering with the streaming.

Example 1.

```
var ns:NetStream=new NetStream( ):
ns.play(<stream>):
LivePass.createMonitoringSession(ns);
```

Example 2. "NetStreamProxy" wraps "NetStream," but prohibits any calls that may adversely impact "NetStream":

```
var ns:NetStream=new NetStream( ):
var nsProxy:NetStreamProxy  new NetStreamProxy(ns);
ns.play(<stream>);
LivePass.createMonitoringSession(nsProxy);
```

In a third embodiment, the monitoring module is built into the streamer module. The player uses the streamer module. The monitoring module within collects data.

Example:

```
var ns:NetStream=new NetStream( ):
ns.play(<stream>);
```

Additional Status Information Examples

As mentioned above, clients downloading/streaming content from content providers are also in communication with content distribution monitor **102** on a recurring, time-driven basis. Each heartbeat contains a snapshot of the session state at the time the heartbeat was sent out and a summary of the events and measurements since the last heartbeat or since the start of the session, as applicable. Examples of information that can be included in a heartbeat (though need not be present in every heartbeat) include the following:

version: The version of the heartbeat.

clientID: A unique identifier associated with the client's monitoring module (described in more detail below) when it is downloaded/installed on the client.

clientVersion: The version of the monitoring module.

customerID: An identifier associated with an entity such as XYZ or Studio.

sessionID: A unique identifier associated with the content viewing session.

objectID: A unique identifier associated with the content being streamed.

In Alice's case, this is the soccer game.

currentResource: The source from which the content is currently being obtained. In Alice's case, this is CDN **146**.

candiadateResourceList: A list of potential sources from which the player can obtain the content. In Alice's case, this could include CDN **144**, CDN **142**, cache **148**, etc. The "currentResource" is one resource in the "candidateResourceList."

resourceUsage: The number of bytes loaded/streamed since the start of the session.

currentBufferSize: The current buffer size.

minBufferSize: The minimum buffer size during the previous heartbeat interval.

maxBufferSize: The maximum buffer size during the previous heartbeat interval.

numBufferEmptyEvents: The number of buffering empty events since previous heartbeat.

currentBitrate: The bitrate at which the content is currently streamed.

playheadTime: For video-on-demand content (i.e., not a live event), this is the time offset of the stream file.

currentlyPlaying: For live content, this is the time offset of the stream since the player started.

joinTime: The amount of time that it took the player to enter the player state for the session.

averageFPS: The average rendering frame per second (FPS) at which the content was rendered since the last heartbeat.

encodedFPS: The FPS at which the content was encoded.

averageFPS: Can be lower than the "encodedFPS" if the client's CPU is unable to render the content at the "encodedFPS" or if the client cannot stream the content fast enough.

totalPlayTime: The total play time since the beginning of the session.

totalBufferingTime: The total buffering time since the beginning of the session.

totalPauseTime: The total pause time since the beginning of the session.

totalSleepTime: The total sleep time since the beginning of the session. The client is in a sleep state if the client is suspended.

sessionTime: The total time elapsed since the session started.

currentState: The state of the player when the heartbeat was sent. Examples of player states include: play, pause, buffering, joining, and seeking.

numRateSwitches: The number of bitrate switches since the beginning of the session.

numResourceSwitches: The number of resource switches since the beginning of the session.

rateSwitchingEvent.time: The time at which a bitrate switch was attempted. measured from the start of the session.

rateSwitchingEvent.from: The bitrate of the stream at the time the switch was attempted.

rateSwitchingEvent.to: The target bitrate of the switch.

rateSwitchingEvent.result: Indicates whether the switch was successful or not. For example, a switch may not be succeed if the client cannot sustain the "rateSwitchingEvent.result" bitrate.

resourceSwitchingEvent.time: The time at which a resource switch was attempted, measured from the start of the session.

resourceSwitchingEvent.from: The resource from which the content was streamed at the time the switch was attempted.

resourceSwitchingEvent.to: The target resource for the switch.

resourceSwitchingEvent.results: Indicates whether the switch was successful or not.

errorList: A list of errors encountered from the start of the session.

FIG. **3** illustrates an example of a process for monitoring the performance of a content player. In some embodiments the process shown in FIG. **3** is performed by a client such as client **172**. The process begins at **302** when information relating to the state of a content player is obtained. For example, at **302**, client **172** collects various information pertaining to the video player that Alice uses to watch the streaming soccer game. Information such as the amount of time an advertisement took to load and which ad server supplied the advertisement is also obtained. In some embodiments, additional processing is performed in the collected information. Examples include computing averages. minimums, maximums. and the $x^{th}$ percentile over the last few samples.

At **304**, at least a portion of the obtained information is reported to a content distribution monitoring server. For example. at **304**, client **172** sends a heartbeat to content distribution monitor **102**. including some or all of the information collected at **302**. Information sent at **304** can also include processed results such averages and minimums, instead of or in addition to raw data.

In various embodiments, the process shown in FIG. **3** repeats throughout a content playing session. The process may repeat regularly (e.g.. once every second). and may also repeat with varying frequency.

The amount of data reported by the client in a heartbeat to content distribution monitor **102** can affect the scalability of the content distribution monitor. The amount of data collected and reported by the client can also potentially impact the performance of the client. If the client is otherwise resource constrained (e.g.. due to other processes. or due to hardware limitations), adjustments can be made to how much information is collected and with what frequency. For example, suppose the buffer size is expected to be of size three seconds. The time period employed by the monitoring module can be set to one second or a few hundred milliseconds. The adjustment can be dynamically adjusted as needed. The period can be decreased if the buffer grows and increased if the buffer shrinks, thus minimizing overhead while still being able to detect a low buffer size which can impact the video quality.

The amount of data sent can be adjusted according to two parameters: the heartbeat "period," and the size of heartbeats, both of which can be adaptively changed by either the client or the content distribution monitor as needed. For example, if the quality experienced by the client is acceptable. the client may reduce the heartbeat frequency. This can entail collecting state information less frequently and can also entail sending collected information less frequently. If the quality degrades, the client can increase the heartbeat frequency accordingly. As one example. client **172** can employ a rule that, as long as the buffering ratio is less than or equal to 0.5%, the heartbeat period is 30 seconds. If the buffering ratio is between 0.5% and 1% the heartbeat period is adjusted to 20 seconds. If the buffering ratio is equal to or greater than 1%, the heartbeat period is adjusted to 10 seconds.

Content distribution monitor **102** can also direct client **172** to increase or decrease the heartbeat frequency based on factors such as the current load on the content distribution monitor (or applicable component thereof). For example, if the load on content distribution monitor **102** exceeds a predefined threshold, clients are instructed to increase the heartbeat interval and/or reduce the detail of the information sent in a heartbeat.

As one example. content distribution monitor **102** can employ a rule that, if its resource utilization exceeds 80%, clients are instructed to double their respective heartbeat intervals. Monitor **102** can also selectively send such instructions to clients. For example, clients with periods under 30 seconds can be instructed to double their periods, while clients with periods above 30 seconds are not sent such instructions.

As another example, when monitor **102**'s utilization exceeds 80%. it can instruct clients to reduce heartbeat size by sending less information. This can be less detailed information (e.g., sending only the number of resource switches, instead of the detailed information about these switches), or can include aggregate measurements about certain events instead of sending individual details. If the client is already aggregating certain information, the measurements can be aggregated over a longer time interval (e.g.. 10 seconds instead of 2 seconds). Additional examples follow.

Instead of sending every buffering event, send the aggregate time the session spent in the buffering state since the last heartbeat.

Instead of sending every measurement of the rendering rate, send the average over the entire heartbeat interval.

Instead of sending information about a switch event at the time when the switch event happened, store the event on the client and send it with the next heartbeat.

FIG. **4** illustrates an example of a process for dynamically adjusting a heartbeat. In some embodiments the process shown in FIG. **4** is performed by a client such as client **172**. The process begins at **402** when an indication is received that a change should be made to either the collection, or reporting of status information (or both). As explained above, the indication can be received as an instruction from content distribution monitoring server **102**, and can also be received as the result of a determination made by the client itself. At **404**, an adjustment is made to the collection/reporting of status information implicated by the indication received at **402**.

Inferring State Information

In some cases, the monitoring and reporting functionality described herein as being provided by a client is wholly integrated with the player itself. For example, Studio might make available a custom player application that can be installed on a client by users that want to watch Studio films. The customer player includes all of the necessary logic to provide complete heartbeat information, and also includes logic for communicating with the content distribution monitor, adjusting the frequency with which status information is communicated to the content distribution monitor, and other functionality described herein. However, depending on the player deployed on a client, not all of the example heartbeat information described above may be directly accessible for collection and transmission. For example, the Flash player plugin used by Alice to view the soccer game does not provide direct access to its state. Using the techniques described herein, content distribution monitor **102** can nonetheless be provided with the state of the player.

In some embodiments, a monitoring module is embedded in the content player deployed on the client. For example, when Alice directs her browser to XYZ's website, a custom Flash-based player is loaded. The custom player has direct access to the API of the video plugin. Possible states of the player are as follows:

Playing: The player's buffer is not empty, and the player renders frames on the screen.

Buffering: The player's buffer is either empty or is less than a predefined threshold, making it impossible for the player to render new frames.

Joining: The player has just connected and has not yet accumulated enough data in its buffer to start rendering frames.

Pause: The player is paused, usually, as a result of a user action. During the pause state, no new frames are rendered even if the buffer is full.

Sleep: The client is suspended.

In the case of the aforementioned Flash player, NetStatus events provided by the player can be used to infer the state, as described below:

Buffering: When one of a NetStatus.Buffer.Empty, NetStream.Play.Start, or NetStream.Play.Reset events is received, an inference is made that the player is buffering data and the video is not playing. For live RTMP streams, the mentioned events might not be available. In that case, if the playhead time has stopped moving and there is no data in the content player buffer, an inference is made that the player is buffering data and the video is not playing.

Playing: When a NetStatus.Buffer.Full event is received, an inference is made that the player is playing video. If the player is currently in a Paused state and the playheadTime starts to move, an inference is made that the player is playing video.

Paused: If the playheadTime does not progress for 1.6 seconds, an inference is made that the player is paused.

Sleep: If the amount of time that has elapsed between two firings of a periodic one second timer is greater than 30 seconds, an inference is made that the client has been in a sleep mode since the last firing of the timer.

Stopped: For HTTP streams, when a NetStatus.Play.Stop event is received, an inference is made that the stream has ended. For RTMP streams, when a NetStatus.Play.Complete event is received, an inference is made that the stream has ended. When a NetStatus.Play.Stop is received, an inference is made that the download has finished, but the stream is still playing.

Error: If the stream stays in stopped state for fifteen seconds at the start of a session, an inference is made that it has failed to connect.

In addition to the player states described above, an additional state can also be inferred:

Zombie state: An inference is made that the player is in a zombie state (a non-operating state with no user actively using it) if it persists in a non-playing state for more than a specified time. Once the player is in a zombie state, the transmission of heartbeats to content distribution monitor **102** is halted. Reporting will resume if the player returns to a playing state. Non-playing states include: "Buffering." "Paused." "Stopped." and "Error."

In addition to the player states, the monitoring module monitors various metrics, including the number of bytes downloaded/streamed since the session started, the bitrate of the stream, and the rendering rate.

Some players, such as the Flash **9** plugin, do not provide a direct API from which the number of bytes streamed/downloaded or the stream bitrate. Nonetheless, a variety of techniques can be used to obtain or estimate the bitrate. As one example, the developer of the player might expose the bitrate through an API call.

Another technique for obtaining a bitrate is to use metadata associated with the content. For example, suppose metadata is made available through a configuration file associated to the content. The configuration resides on an origin server such as origin server **198** and also includes other information such as the title of the content, its genre, and a list of CDNs where the content is available.

Yet another technique for obtaining a bitrate is to examine the content URL. For example, a movie might be accessible at the URL: http://www.CDN-C.com/Studio/JaneEyre300 Kbps. In this case, the bitrate of the content is likely to be 300 Kbps.

An estimation of the stream bitrate can also be made. The estimation, along with the amount of time the player was in the playing state, and the size of the buffer, are used to estimate the number of bytes downloaded/streamed. As one example, suppose a player streamed data at 400 Kbps, was in a playing state for 315 seconds, and at the time of taking the measurement the buffer size contained data for playing another 15 seconds. The total number of bytes downloaded is: 400 Kbps*(315 seconds+15 seconds)=16 MB.

In the event that the bitrate is available through multiple of the above techniques, in various embodiments each

applicable approach is used, in a specified order, until a valid bitrate (e.g., greater than 0 and less than 10,000 kbps) is obtained.

The number of bytes downloaded can be estimated as (totalPlayingTime+bufferLength)*bitrate. If multiple bitrates are used, in some embodiments the bytes downloaded is estimated as the sum of totalPlayingTime[bitrate] *bit-rate+bufferLength*current Bitrate.

Rendering quality is another example metric that can be obtained and provided to content distribution coordinator **102**. Rendering quality is the ratio between the frames per second (FPS) rendered by the player and the FPS at which the stream was encoded.

Some players do not directly provide the information needed to compute the rendering quality. For example, Flash **9** has an API that exposes rendered frames per second. but not encoded frames per second. One technique to compute the encoded FPS, in such a scenario, is as follows. The rendering FPS is measured and an assumption is made that the encoded FPS is the maximum FPS observed over the course of the session. Another technique is to estimate the encoded FPS as the maximum of a three second sliding average of rendered FPS sampled five times a second.

In order to ensure accurate accounting of the player status over the course of the session. it is important to determine the time intervals when the system is in sleep or suspend mode. These intervals can be detected using notification APIs provided by the system for this purpose. For systems that do not provide such API (e.g.. Flash Player, or Silverlight). the intervals can be estimated by setting a timer to fire at periodic intervals (e.g. every 1 second). At each firing of the timer the current system time is recorded, and the time elapsed since the last firing is computed. If the time elapsed is greater that a given threshold (e.g., 10 seconds). an inference is made that the given elapsed time interval was spent in sleep or suspend mode.

Inferring Player Capability Information

The following are example player capabilities that impact the quality of the video experience: processing speed, video rendering capabilities, amount of memory available. and amount of bandwidth available.

- Processing speed: The monitoring module can use the API of the content player, or the underlying platform to read the processing speed of the player. For example. if the content player is implemented on the Silverlight platform the monitor can use the Environment.ProcessorCount to obtain the count of the processors available.

For platforms whose API does not provide direct reading of processing speed capabilities. the monitoring module can derive it by using a timer to measure the time required to perform a fixed CPU-intensive computation.

- Video rendering capabilities: Some content players have the ability to use hardware acceleration to render video. The monitoring module can determine the speed at which the content player can render video, either by using the API of the content player or the underlying platform. For example, a monitoring module using the Flash platform can use the flash.system.Capabilities.ScreenResolutionX to determine the screen resolution.
- Available memory: The amount of available memory has a direct influence on how well the content player can support rendering video while performing the computation required for monitoring and other user-interaction tasks. The monitoring module can obtain the amount of available memory by using the API of the underlying platform. For example, a monitoring module using the Flash platform can use the API flash.system.System.totalMemory to termine how much memory is available.
- Available download bandwidth: The available download bandwidth has a direct impact on the quality of the video stream that can be downloaded and played. The monitoring module can infer the available bandwidth by measuring the time interval it takes to download a fixed size file from the server. Alternatively. the underlying platform can provide API that can be used to determine the download bandwidth. For example, a player using the Flash platform can use the API flash.net.NetStreamInfo.currentBytesPerSecond.

FIG. **5** is an illustration of an environment in which status information is received and processed. In various embodiments, the services provided by content distribution monitor **102** are implemented across a scalable infrastructure, particularly in embodiments where telemetry data is received from all clients. In the example shown, the elements contained within dashed region **502** collectively provide the functionality of content distribution monitor **102**. Each of the layers (e.g., dispatcher layer **520**) is horizontally scalable and their respective components can be implemented on standard commercially available server hardware (e.g., having a multi-core processor, 4G+ of RAM. and Gigabit network interface adaptors) running a typical server-class operating system (e.g., Linux).

Clients **504**-**512** each include a monitoring module that collects status information. When the monitoring module on a client is activated, the client is mapped to a dispatcher server. As one example. when the monitoring module starts, it reads a configuration file that includes a list of dispatcher servers. The monitoring module selects a dispatcher server at random from the list.

A dispatcher server (**514**) includes two conceptual modules. The first module implements a communication interface for receiving status information from clients. In some embodiments the module is implemented using an off-the-shelf web server. and allows clients to connect over the HTTP protocol (and also allows clients to securely communicate via SSL). Data received by the first module is passed to the second module. The second module normalizes the data (to a format suitable for further processing) and passes the normalized data to a real-time stream processing component (**516**).

The real-time stream processing (RSP) layer includes an optimized software component that processes the telemetry data that it receives from the dispatcher in real-time. A dispatcher sends all heartbeats belonging to the same session to the same RSP component.

In some embodiments the RSP component is implemented as a continuously running service that reads and processes the telemetry data received from dispatchers via the network over TCP. The telemetry data stream comprises individual records. each of which represents the telemetry data sent by the monitoring module. The RSP component reads network data one record at a time and parses the data into a local data representation. The data received by the RSP component can be stored as in-memory hash tables of records allowing fast execution. and very high throughputs. Since the RSP component does not require old information, it can periodically purge the in-memory hash tables and increase scalability accordingly. In other embodiments, optimized in-memory databases are used.

A mapping function to map heartbeats having a session identifier "ID" to a particular RSP component "i" as follows:

i=hash(ID) mod m.

where hash( ) is a hash function and "m" is the total number of RSP components.

Once an RSP component parses the data records, it performs two main tasks. First, it performs data filtering. A filter is a logical expression and is installed at each RSP component instance. As one example, the following filter would identify viewers located in San Francisco, connected to ISP SP1, streaming from CDN A, one of two particular shows:

(city="San Francisco" AND ISP="SP1" AND CDN "CDN A" AND ((show "NewsAt10") OR (show "SundayMagazine"))

For each message of incoming telemetry data, the (key, value) pairs in the record are matched against the filter. If the filter is matched, the data is associated with the filter.

The second task performed is to compute snapshots and on-line statistics over the telemetry data matching each filter. One example of a snapshot is the number of players that are in a particular state (e.g., "playing"). The RSP component generates sequences of these snapshots (e.g., one every second). Examples of statistics computed by the RSP component include: the average number of bytes played over all video streams matching a filter over a given time interval (e.g., 10 seconds) and the minimum frames per second experienced by a stream matching a filter over a time interval. Snapshots and statistics are updated continuously, from new telemetry data received from clients.

The RSP component provides its computed snapshots and statistics to a real-time global aggregation component (**518**). The real-time global aggregation (RTGA) component aggregates the information provided by the RSP component for each filter specified by a user (described in more detail below).

As explained above, each RSP component (**516**) receives (via a dispatcher) telemetry data from a subset of the monitoring modules and calculates snapshots and statistics for all filters. Each RGA component instance is in turn responsible for a subset of the filters. Based on the identifier of the filter, all RSP components send data for that filter to a single RGA component. The RGA component combines the data from all RSP components for the filters that it is responsible for, resulting in a global snapshot and statistics based on information from all monitoring modules. Examples of aggregation operations performed by an RGA component include: counting the total number of viewers that match a particular filter, determining the percentage of viewers matching a given filter that are in buffering state, the join time distribution experienced by viewers joining a given stream, the current number of viewers in a given city, the rank of the most popular live events, and so on.

In some embodiments an RGA component's functionality is implemented as a continuously running service. It reads records sent by the RSPs asynchronously, thus achieving a high throughput. The RGA component stores the records it receives in in-memory hash tables allowing optimized access for real-time processing. Old information is periodically purged from the hash tables to improve efficiency.

As shown in FIG. **5**, gateway **522** provides a web service API **524** for accessing data. Through the API, RGAs data is available on a per-filter basis. In addition, it also exposes APIs to edit and install new filters and aggregation functions. In some embodiments gateway **522** is implemented using an off-the shelf web server (such as Apache) with customized code to handle the various web service API calls. The handlers return data for a given web API call in various data formats including XML, JSON, SOAP, and HTML, as applicable. The handlers serve as middleware for querying and interactively controlling the RSPs and RGAs.

Gateway **522** also provides access controls and persistently stores the information regarding the API requests, data access and presentation policies, and filter descriptions. The information is maintained in a persistent database, such as mySQL or Oracle database.

Automatically Detecting and Resolving Content Distribution Problems

As explained above, content distribution monitor **102** aggregates telemetry information from all clients, processes the information, and makes available multi-dimensional results in realtime. Examples of dimensions include:

- Client properties: Including browser type and version, player type and version, operating system, CPU speed, connection type, IP address, geographic location, language, autonomous system, and ISP.
- Content properties: Including category, title, show, episode number, duration, encoding format, encoding quality, and language.
- Content source properties: Including CDN, data center, and clients served.
- User properties: Including whether the user (of the client) is a premium or free user, and returning or first-time visitor.

Using the techniques herein, granular information such as the quality experienced by viewers located in Denver, connected to SP2, and streaming videos from CDN B, using Flash Player 10 can be tracked. Further, by aggregating and correlating the data it receives from all clients, content distribution monitor **102** exposes, in realtime, the performance of content sources and network providers, (ISPs). Problems in content delivery can be automatically detected by examining the results.

FIG. **6** illustrates an example of a process for detecting a problem in a content distribution. In some embodiments the process shown in FIG. **6** is performed by content distribution monitor **102**. The process begins when information associated with a first content player and a second content player is received from a first client (**602**) and second client (**604**), respectively. For example, at **602**, telemetry information is received from client **504** by dispatcher **514**. At **604**, telemetry information is received from client **506** at dispatcher **526**. At **606**, the received information as aggregated. For example, at **606**, distributed processing of the received data is performed by the RSP and RGA layers. Finally, at **608**, a determination is made from the aggregate information that a content distribution problem is indicated. For example, at **608**, gateway **522** determines that the CDN from which client **504** is obtaining content is experiencing a problem. Specific examples of analysis and diagnosis will now be given.

Example: Diagnosing Client Problems (Single Client, Single CDN)

Suppose a content owner, such as Studio-A, distributes its content via a single CDN, such as CDN C. Suppose that the content is encoded at multiple bitrates B1 and B2 where B1<B2. If a client A is able to sustain the download rate, but detects problems rendering the frames, an inference can be made that the client A likely has problems with its CPU utilization. An example of performing such an inference follows.

Observe the client buffer size over a period of time T (e.g., T=10 seconds) when the client is streaming at bitrate B2. Observe the rendering quality measured using dropped frames reported by the player. If the client buffer size is greater than B_threshold (e.g., B_threshold 0.5*B_max) at all times during the period of observation, and the rendering quality is less than R_threshold_1 (say, R_threshold_1=0.6), then a conclusion can be made that the client cannot sustain displaying bit-rate B2 due to CPU issues on the client.

Perform the same observation with the client playing at bit-rate B1. If the client buffer size is greater than B_threshold (e.g., B_threshold=0.5*B_max) at all times during the period of observation, and the rendering quality is greater than R_threshold_2 (e.g., R_threshold_2 0.75), then a conclusion can be made that the client can sustain displaying bit-rate B1.

A numerical example is now provided. Consider a setting where a client streams a 1 Mbps stream for 10 seconds with buffer size varies between 9 and 10 seconds. Suppose that B_max is 10 seconds and B_threshold is 5 seconds. Assume that the rendering quality at 1 Mbps is 0.55 over this interval. When the client plays a 500 kbps stream instead, the buffer size is in the same range (9-10 seconds), but the rendering quality is 0.9 instead. Then, assuming that R_threshold_1 0.6 and R_threshold_1 0.75, a conclusion can be made that the client can sustain 500 kbps but not 1 Mbps.

### Example: Diagnosing CDN Problems (Single Client, Two CDNs)

Suppose a content owner, such as Studio-A, distributes its content via two CDNs, such as CDN D and CDN E. If a client A is able to sustain the download rate from CDN D, but not CDN E, then the client can conclude that CDN E has problems streaming to client A. An example of performing such an inference follows.

Compute an aggregate quality metric Q using various measurements from the client over a time T (e.g., T 60 seconds). Examples of measurements which can be directly used as the metric Q across a group of clients over a time T include:

- buffering ratio: The total number of seconds the client experienced buffering divided by the total playing time of the clients during interval of time T.
- join time: The average joining time over all clients during time T over a group of clients.
- join failures: The fraction of cases (across all attempts) where the join failed during time T.

If the quality Q is greater than a threshold T1 (e.g., T1=0.95) for CDN D, but lower than another threshold T2 (e.g., T2=0.50) for CDN E, then a conclusion can be made that CDN E has problems streaming to client A.

One example remedial action that can be taken at the client is to select CDN D for streaming the current stream, or use it first for future streams.

A numerical example is provided. Consider a setting where a client M streams from CDN D and experiences a buffering ratio of 0.5 over K1 attempts and time T1. Also, client M streams from CDN E, and over K2 attempts and time T2 experiences a buffering ratio of 0.02. In this case, given that K1 and K2 are above a threshold (e.g., both greater than 2), and T1 and T2 are above a threshold (e.g., both are greater than 10 minutes), a conclusion can be made that CDN D has problems streaming to the client. A further refinement can be performed by computing the buffering ratio for each of the K1 (or K2) streaming attempts from the CDNs to the client. In this case, it can be stated that CDN D has problems streaming to client M if more than 75% of the attempts from CDN D (out of the K1 attempts) have buffering ratio greater than 0.5, and more than 75% of the attempts from CDN D have buffering ratio less than 0.02.

### Example: Two Clients in the Same Location Using Different ISPs Both Streaming from a Single CDN

Suppose a content owner, such as Studio XYZ, distributes its content via a single CDN, such as CDN C. The CDN comprises several geographically distributed servers. As clients request content, they are assigned to a particular server based on their geographical location. If many clients at the same geographical location, but using different ISPs, experience quality problems, an inference can be made that the CDN servicing that geographical location is having a problem. Note that since XYZ also uses the services of CDN C, the experiences of clients obtaining XYZ's content from the same location can be included in the analysis. One way this analysis can be implemented on content distribution monitor 102 is as follows.

First, obtain location and network information for all clients. Examples of location information include city, state, country, DMZ code, and geographic coordinates. The location information can be obtained in a variety of ways, such as by providing the client's IP address to a geo-location service such as Quova. Also obtain network information such as Autonomous System Number (ASN) information and ISP information for the client.

Second, group clients based on their ASN and one geographic attribute, such as DMZ code, city, or state. Let G(ASN1, Geo1) denote the set of clients connected to a specific ASN, ASN1, which have the same geographic attribute value, Geo1.

Third, compute an aggregate quality Q for each group G(ASN, Geo) over last T seconds. Denote this aggregate quality by Q(G(ASN, Geo)). Examples of quality metrics are provided above.

Finally, for each Geo attribute value Geo1, check whether there are at least two group of clients connected via different ASNs. If the aggregate quality of a certain fraction F of the groups is greater than a threshold BufferingThreshold1, then conclude that the CDN is experiencing problems serving the clients at location Geo1. In some embodiments a minimum group size is defined so that only groups having a number of clients larger than a given threshold are considered. As one example, the fraction F is selected between 0.75 and 1.

A numerical example is now provided. Consider a system setting where BufferingThreshold1=0.1, MinGroupSize=200, and F=1.0. Based on an analysis of their IP addresses, 20,000 viewers are determined to be located in San Francisco, connected to CDN C, and watching a new release by Studio. Assume 5,000 clients are connected to ASN1 belonging to SP1; 5,000 clients are connected to ASN2 belonging to SP2; 9,900 clients are connected to ASN3 belonging to SP3; and 100 clients are connected to ASN4 which belongs to a wireless company. Four groups exist: G(SF, ASN1), G(SF, ASN2), G(SF, ASN4), and G(SF, ASN4), respectively.

Each client reports for each session (a) the total time it spent in buffering during every 10 second interval; and (b) the total playing time during the same 10 second interval. For example, suppose a user Charlie has the IP address 1.2.3.4 which belong to ASN1, and watches the stream with the URL rtmp://www.CDNC.example.com/12346/video.flv. Charlie reports in a particular heartbeat that he has experienced 1 second of buffering and 9 seconds of playing time.

Another user, Bob, has the IP address 2.3.4.5 which belongs to ASN3, and watches the stream with the same URL. Bob reports in a particular heartbeat that he has experienced 0.5 seconds of buffering and 5 seconds of playing time. Bob's player spent the remaining 5 seconds in pause state. The pause time is not used in the computation of the buffering ratio.

Content distribution monitor **102** computes over each interval of time T=5 minutes. the aggregate average buffering ratio. defined as the (1) the total buffering time experienced by all sessions in a group during T divided by (2) the total playing time over all sessions in the group during the same time interval T. The aggregate quality ratio for each of the four groups is as follows:

Q(G(SF. ASN1))=0.14. Q(G(SF, ASN2))=0.21, Q(G(SF. ASN3))=0.18. Q(G(SF. ASN4))=0.04.

Since the number of clients in G(SF, ASN4) is less than MinGroupSize, the diagnosis analysis ignores this group. Also, since the buffering ratio of all remaining groups is greater than BufferingThreshold1. a conclusion is made that CDN C experiences quality issues serving clients in San Francisco.

Suppose the above example is modified such that ASN4 has 1000 clients, and Q(G(SF, ASN1)) 0.02, Q(G(SF, ASN2))=0.21. Q(G(SF. ASN3))=0.18. Q(G(SF. ASN4))=0.04. In this case, an inference can be made that CDN C does not experience quality issues serving clients in San Francisco in general. The problem can then be narrowed down further and smaller or different sets of clients investigated, such as clients that are connected to ASN1 served by CDN C.

### Example: Inferring Problems in a Particular ASN (Single CDN)

Suppose a content owner, such as Studio XYZ. distributes its content via a single CDN, such as CDN C. If the quality of clients at the same location but using different ISPs is very different. an inference can be made that the quality problems experienced by the clients are due to the ISP (and not due to the CDN). One way this analysis can be implemented on content distribution monitor **102** is as follows.

Perform the first three steps of the previous diagnosis (describing the diagnosis of CDN quality issues). As a final step, let G(ASN1, Geo1) be a group experiencing low quality. If there is at least another group, G(ASN2, Geo1) at the same location experiencing high quality, then conclude that ASN1 is the cause of quality issues. In particular, a conclusion that ASN2 has quality issues is reached if, and only if. Q(G(ASN2, Geo1))−Q(G(ASN1, Geo1))>BufferingRatioDiscrepancy.

A numerical example is now provided. Consider a system setting where BufferingRatioDiscrepancy=0.1. Suppose 5.000 viewers are determined to be located in New York. connected to ASN1, and streaming a baseball game from CDN C. Another 4,000 viewers in New York are connected to ASN2 and streaming the same game from CDN C.

Each client reports for each session (a) the total time it spent in buffering during every 10 second interval: and (b) the total playing time during the same 10 second interval. The aggregate quality ratio for each group as computed by content distribution monitor **102** is: Q(G(NY. ASN1))=0.14. and Q(G(NY. ASN2))=0.03, respectively.

Since Q(G(SF, ASN1))−Q(G(SF, ASN2))>BufferRatioDiscrepancy, the quality issues are pinpointed as being ASN1.

### Example: Inferring Problems with a Particular CDN (Multiple CDNs)

Suppose a content owner, such as XYZ, distributes its content via multiple CDNs (e.g., CDN A, CDN B, and CDN C). If clients at the same location and using the same ISP experience significantly different quality when connected to different CDNs, the difference in quality can be attributed to CDNs. One way this analysis can be implemented on content distribution monitor **102** is as follows.

First, obtain location and network information for all clients. Second. classify each client based on its ASN. the CDN it gets data from, and one of its geographic attributes (e.g., DMZ code, city, or state). Let $G(ASN_i, CDN1, Geo_i)$ denote the set of clients connected to a specific ASN and a geographic region (e.g., for i 1, 2, 3, 4, 5), getting data from same CDN. CDN1. One way of determining the particular CDN from which a client is receiving content is to extract it from the URL used by the client to obtain the content.

Third, compute an aggregate quality Q for each group G(ASN. CDN. Geo) over the last T seconds. Denote this aggregate quality by Q(G(ASN. CDN, Geo)).

Finally, check whether there are at least K pairs of groups that share the same ASN and geo-location, but get their data from different CDNs, and which experience different quality. In particular, let G(ASN1. CDNi, Geo1) and G(ASN1, CDNj, Geo1) be one such pair of groups. Then if Q(G(ASNi, CDN1, Geoi))−Q(G(ASNi, CDN2, Geoi))>QualityThreshold2 for greater than a fraction F of the K pairs, a conclusion is made that CDN2 has problems serving clients in general across a large set of regions. Alternatively, if this happens for smaller than a fraction F but a non-zero set, then a conclusion can be made that CDN2 has problems serving those (ASN, Geo) combinations for which the difference in quality exceeds the quality threshold.

In some embodiments, a minimum group size is defined so that only groups having a number of clients larger than a given threshold are considered.

A numerical example is provided. Content owner Charlie's Studio uses CDNs A and B for delivering content. At some point. there are 1100 streams from ISP-1 in San Francisco, 1200 streams from ISP-2 in Los Angeles, 1500 streams from ISP-3 in New York, 1050 streams from ISP-4 in Boston, 600 streams from ISP-1 in Denver, with exactly half the users in each group streaming from CDN A and the other half from CDN B. A minimum group size of 500 users streaming from a particular (ASN, CDN, Geo) group is used, and hence there are four groups to consider. A requirement of at least four pairs (K 4) exists. For CDN A, the buffering ratios for users in San Francisco. Los Angeles, and Boston is 0.3. and for users in Denver it is 0.1. For CDN B. the buffering ratio of all user groups is 0.05. The quality threshold to separate good and bad quality is 0.1, and the fraction F required is 75%. This condition is satisfied. since three out of the four groups have a quality difference between CDN A and CDN B as 0.25. Hence, a conclusion is made that CDN A is having problems streaming to users.

### Example: Multiple CDNs with CDN Optimization

Suppose a content owner, such as XYZ, distributes its content via multiple CDNs (e.g., CDN A, CDN B, and CDN C). Assume that if a client connected to CDN A experiences buffering beyond a threshold. it switches to another CDN (and stays there for at least a threshold interval of time). Based on the amount of switching observed from a CDN, the relative extent of quality problems the CDNs have can be

quantified. One way this analysis can be implemented on content distribution monitor **102** is as follows.

For each CDNi for a period T, determine the number of clients Ni that start with CDNi, and the number of clients Mi that start with CDNi and switch away from CDNi (i.e. count only the first switch of a client). Compute the switch fraction SWi=Mi/Ni for all CDNs. If (SWi−avg(SW)>SW_Threshold). A conclusion can be made that CDNi has quality problems in general. Similar analysis can also be performed with respect to a geographic region or an ASN restriction.

A numerical example is provided. A content owner Charlie's Studios uses CDN A and CDN B to deliver content. Over a 10-minute period, 10,000 users watch content starting streaming from CDN A, and in the course of viewing, 1000 users switch to CDN B based on a client-based quality detection algorithm. In the same period, 10,000 users watch the same content starting streaming from CDN B, and out of these, 2,500 switch to CDN B. The switch fraction for CDN A is 0.1 and that for CDN B is 0.2. Switching threshold SW_Threshold is 0.1, and a conclusion is made that CDN A has quality problems in general.

Other content distribution problems can also be detected using the techniques described herein. For example, a problem with a content item itself (e.g., a particular movie) can be inferred if several clients, using different CDNs and different ISPs, experience quality issues with the same content. As another example, a problem with ad server **150** can be inferred if several clients report excessively long advertisement load times or timeouts.

A numerical example is provided. A content owner publishes 100 clips daily using two CDNs: CDN A and CDN B. One of the clips, clip X, has 100% join failures on both CDNs. Using this information, an inference can be made that there is a problem in publishing clip X. Now assume that there is 100% join failure for clients that join CDN A, but <2% of the users have problems with CDN B. Then an inference can be made that CDN A's publishing path has problems for that clip (but CDN B's does not).

Other content distribution problems can also be detected using the techniques described herein. For example, a problem with a content item itself (e.g., a particular movie) can be inferred if several clients, using different CDNs and different ISPs, experience quality issues with the same content. As another example, a problem with ad server **150** can be inferred if several clients report excessively long advertisement load times or timeouts.

Protecting Content Delivery

In various embodiments, when problems are detected by content distribution monitor **102**, corrective actions are automatically taken, such as by a control module included in content distribution monitor **102**, or by a separate control system, configured to work in conjunction with content distribution monitor **102**.

FIG. **7** illustrates an example of a process for correcting a problem in a content distribution. In some embodiments the process shown in FIG. **7** is performed by content distribution monitor **102**. The process begins at **702** when a determination is made that a problem in a content distribution exists. In some embodiments the processing of portion **702** of FIG. **7** occurs in accordance with the process shown in FIG. **6**. At **704**, a determination is made that at least one setting of a client should be updated, and at **706**, the update is sent to the client.

As one example, suppose content distribution monitor **102** continuously monitors the quality of clients' receipt of content. If this quality drops under a predefined threshold, the control module (or control system) will try to localize the problem to one or more (CDN, city) pairs, as described above. If successful, the control module instructs clients in each city to use a different CDN. As another example, the content distribution monitor can be configured to monitor audience quality for triplets (CDN, city, ISP), and based on the data infer whether a quality issue is due to a client's CDN, ISP, or both.

The communication between the control module and the clients can be implemented in a variety of ways. In one example, the control module updates a centrally maintained configuration file that is periodically read by each client. In another example, the control module opens a connection to any clients implicated in the problem and sends commands directly to those clients. In various embodiments, instead of specifically instructing the client to make a change, a list of alternatives or suggestions is instead provided to the client, and the client is configured to perform some local decision making. As one example, suppose a client is experiencing difficulties obtaining advertisements from advertisement server **150**. If multiple other advertising servers exist, rather than content distribution monitor **102** determining which advertisement server the client should switch to using, content distribution monitor **102** sends a list of three such serves, and the client is tasked with selecting from the list.

If the problem is determined to be CDN A, and CDN A is the only CDN able to provide the content, in some embodiments the control module instructs those clients affected by CDN A's problem to reduce the rate at which they stream. In addition, new clients joining CDN A can be instructed to use a lower rate than they might otherwise use.

If the problem is determined to be CDN A, and multiple CDNs are able to provide the content, in some embodiments the control module instructs those clients affected by CDN A's problem to switch to another CDN. In addition, new clients joining CDN A can be instructed to avoid CDN A.

If the problem is determined to be an ASN, in some embodiments the control module instructs the clients connected to that ASN to reduce the rate at which they are streaming. In addition, newly clients joining from the ASN can be instructed to use the lower rate.

If the problem is determined to be the client (e.g., the client's connection is congested or the CPU is overloaded), in some embodiments the control module instructs the client to reduce the rate at which it streams.

If the problem is determined to be an ad server, in some embodiments the control module instructs all clients to cease fetching or attempting to fetch advertisements from the ad server, and instead to fetch them from a different ad server.

In various embodiments, clients include logic to interpret the control module instructions and information. A client makes the ultimate decision based on the instructions and information received from control module information and/or its own internal state. The state can include both real-time state as well as historical state. Examples of states include, but are not limited to CPU utilizations, rendering rate, and whether the viewer watches full screen or not.

Example: Synchronized Data Requests

The attempt by many clients to access the same content at the same time is referred to herein as a synchronized data request. One type of synchronization is "viewer-induced synchronization" (also referred to as a "flash crowd"). This can occur, for example, when a large number of viewers attempt to connect to a live event in a short period of time, such as at the start of a soccer game. Viewer-induced synchronization can also occur with video on demand con-

tent, such as the first day a movie is released, or when a video goes viral. A second type of synchronization is "failure-induced synchronization." This can occur when a failure impacts a large number of players and the players attempt to recover. For example, a failure of one CDN could trigger impacted clients to failover to another CDN. This would cause synchronized requests on the second network, potentially causing adverse effects on the second network. If the first CDN recovers, an oscillation can occur where all of the clients of both CDNs swarm back and forth between the two CDNs because neither CDN is able to handle the entire capacity. A third type of synchronization is "player-induced synchronization." This can occur when a large number of players synchronously take an action that triggers requests on a backend server or servers. An example of this is the fetching of advertisements from an ad server during a timeout in a live sporting event. In traditional content delivery environments, overloaded servers or other resources can rapidly become unusable—not just to new clients but to existing clients as well.

Using the techniques described herein, the detrimental impact of synchronized data requests is reduced, and in some cases is prevented from otherwise occurring (e.g., through predictive analysis). One process for managing synchronized data requests is illustrated in FIG. **8**. In some embodiments the process shown in FIG. **8** is performed by content distribution monitor **102**. In other embodiments, portions of the process shown in FIG. **8** are performed by other components within the content delivery environment, such as CDNs, or nodes within those CDNs.

The process begins at **802** when load thresholds for components in a content delivery environment (e.g., as illustrated in FIG. **1**) are determined. Examples include the maximum number of concurrent connections (e.g., **1000** for a particular server in CDN **142**), the amount of bandwidth available (e.g., 5 Gbps collectively for nodes located at a particular data center), and the rate at which new connections can be accepted (e.g., 100 per second for requests that require the involvement of an authentication server or 500 per second for requests that do not). The capacity information can be collected and stored in a variety of ways. As one example, content distribution monitor **102** can be configured to receive information from applicable components (e.g., as part of an initialization routine) and store it in a database. As another example, instead of providing explicit numerical limits, components (e.g., CDN **146**) can provide to content distribution monitor **102** periodic status information indicative of load, such as "can accept up to 1000 more clients," "do not send more clients," and "send clients at the rate of 10 per second."

At **804**, data is received from multiple nodes included in the content delivery environment. As one example of the processing at **804**, heartbeat data is periodically collected from clients, as explained above. Data is also collected from sources such as content distribution network servers and from servers supporting content distribution, such as authentication token generation servers. Data collected from servers can be made available to content distribution monitor **102** in a variety of ways. As one example, servers can explicitly provide load information, such as "have 948 viewers," "using 800 Mbps," or "load average is 0.83." As another example, servers can monitor themselves (or be monitored by another appropriate component) and provide to content distribution monitor **102** periodic status information indicative of load, such as "can accept up to 1000 more clients," "do not send more clients," and "send clients at the rate of 10 per second."

At **806**, the data received at **804** (or portions thereof) is evaluated to (1) determine whether any of the components of the content delivery environment are in a loaded state and/or (2) predict whether any of the components are likely to be in a loaded state in the future. As one example, at **806**, content distribution monitor **102** evaluates aggregate client heartbeat information and determines that, in the last two minutes, the percentage of clients experiencing connection failures with respect to a particular CDN has increased from 2% to 10%. Suppose that when a client attempts to establish a new connection (with a first CDN) and that connection fails, the client is by default configured to wait five seconds, retry the first CDN once, and then attempt to connect to a second CDN. So long as the connection failure rate is low, the second CDN is able to absorb users that would otherwise obtain content from the first CDN. If the connection failure rate exceeds 15%, however, the second CDN will not be able to accommodate the new users and service will degrade. Additional examples of the analysis performed at **806** are provided below.

Based on the processing performed at **806**, remediation actions are taken (**808**) to prevent specific components' thresholds from being violated. Using the previous example, content distribution monitor **102** can temporarily modify the retry behavior of those clients experiencing connection failures by sending them the appropriate set of commands. As one example, content distribution monitor **102** could instruct clients to retry the first CDN multiple times before attempting to connect to the second CDN. As another example, content distribution monitor **102** could instruct clients to wait 20 seconds before the first retry, instead of the default 5 seconds. As yet another example, clients with existing connections to the second CDN could each have their bitrates reduced to accommodate the anticipated increase in new connections. Additional examples of the actions that can be taken at **808** are provided below.

Remediation Scenario 1: Balance the Load Between CDNs During an Initial Period and Reduce the Starting Bitrate.

One load parameter for a CDN is total traffic (e.g., Gbps) delivered. Content players are sometimes configured to adaptively change their bitrates midstream based on changes in the environment. Typically such players initially receive at a lower bitrate and then are increased to a higher bitrate if the client can support it. During a viewer-induced synchronization, the CDN is subject to two different increases in total traffic—the increased number of users demanding the content, and the automatic increase in bitrate their clients will attempt. An approach to protecting the CDN as a resource is as follows:

(a) Determine the load threshold beyond which the system will degrade (**802**). One way of performing the determination is to use historical data to correlate the performance of the system with the load of the system. From this information a threshold is determined beyond which the load begins to degrade.

(b) Collect bitrates from all or a sample of viewers viewing the event (**804**).

(c) Detect that a viewer-induced synchronized request is occurring and that the load will exceed the threshold (**806**). One way of determining that the load will exceed the threshold is as follows. Realtime performance and load data is analyzed using anomaly detection logic to determine if any of the performance metrics show signs of degradation. Examples of performance metrics include buffering quality,

connection failures, connection interruptions/other failures to stream, time taken to setup a connection, and time taken to start streaming.

(d) Maintain a low bitrate for new viewers and reduce bitrates for existing viewers until the flash crowd is over and then gradually increase the bitrate to provide the best experience for the viewers (**808**).

An illustration of this approach is provided in FIGS. 9A-9C.

FIG. **9A** is a graph that illustrates the number of concurrent viewers over time across three CDNs. In the example shown, "CDN C" hits a limit when the load approaches 200,000 viewers.

FIG. **9B** is a graph that illustrates the average bitrate for viewers across each of the three CDNs. Since CDN C has reached a limit, the bitrate increase is stopped to prevent the CDN from overloading.

FIG. **9C** is a graph that illustrates the total traffic pushed by each of the three CDNs. By balancing the load across multiple CDNs and by keeping the average bitrate low, the increase in total traffic is maintained at a steady pace and the compounding effect of the bitrate increase and the flash crowd is avoided.

Remediation Scenario 2: Fan Out Clients Over Multiple CDNs During Flash Crowd Joins and Consolidate Once the Peak has been Reached.

A second load parameter for a CDN is the number of new connections it is able to accommodate per interval of time (e.g., **100** joins per second). If the CDN is approaching the limit for the parameter, one way to protect the CDN while still allowing new viewers to connect is to offload some of the new connections to other (backup) CDNs during the flash crowd and then to migrate them back gradually to the primary CDN after the flash crowd is over.

Different reasons exist for, in the absence of a flash crowd, constraining viewers to particular CDNs. One reason is cost—one CDN may be considerably less expensive than another, thus, wherever it is possible to constrain viewers to the cheaper CDN (without losing them), it may be desirable to the content owner to do so. Another reason is performance—one CDN may serve content more efficiently than another (e.g., due to where data is cached), thus it may be desirable to the content owner to constrain viewers to the more efficient CDN where possible. It may likewise be desirable for the content owner to relax rules that would otherwise be applied to the routing of traffic, such as at key times (e.g., the start of a game or the last few minutes of a detective show), and particularly if the alternative is to allow the network component to potentially get into a state of overload.

Remediation Scenario 3: Use Policies.

Actions to protect CDNs can be based on policies set by the content provider. As one example, the content provider can specify a bitrate policy that allows certain (premium) viewers to stream at a high bitrate, but forces other (non-premium) viewers to stream at the lowest bitrate. Such policies can be provided to content distribution monitor **102** through an administrative console and used both during normal operation, and/or used when remediating significant synchronized requests. As another example, a CDN allocation policy can specify that certain viewers can connect to a more expensive and higher quality CDN while all other viewers are pushed to less expensive overflow/backup CDNs during periods of synchronized requests. Overflow capacity can also make use of cloud platforms which provide increased capacity, but will not provide service levels required for streaming in the normal case.

Actions to protect CDNs (or other resources) can be based on policies set by the CDN. As one example, the operator of CDN **146** can specify that, in the event of a failure of another CDN, CDN **146** is willing to take on additional capacity up to 85% of its full capacity (thus creating an artificial ceiling that reserves 15% of its capacity). As another example, the CDN can provide simple status information to content distribution monitor **102** indicating its willingness to take on additional clients. If several servers within the CDN are loaded or approaching being loaded, the CDN (and/or individual servers) can indicate that they would like to refuse new connection attempts.

Remediation Scenario 4: Predictions.

In addition to realtime or other current metrics, historical information about the patterns of clients (and/or loads on servers) can also be used to determine whether preventative/remediation actions should be taken. As one example, suppose a popular television show airs every week at 7 pm Pacific Time and draws a sizable number of viewers. It can be anticipated that a flash crowd might occur during the next airing that occurs.

Another use of historical information is as follows. CDNs typically show signs of high load before they collapse due to that overload. For example, the time to setup a connection on a CDN server or the number of errors received from CDNs may increase as load increases. These can be used as signals to prediction logic that will determine that a CDN is loaded before reaching this state and without hardcoded thresholds. Three examples of detecting CDN load thresholds are as follows:

(a) The median connection setup time over the last 1 minute exceeds 5 seconds.

```
loaded false
  connectionSetupTimesMs=
  <connection setup times over the last 1 minute in
  milliseconds>
medianConnectionSetupTimeMs
  percentile(connectionSetupTime, 0.5)
if medianConnectionSetupTimeMs>5000 and
  concurrentViewers>10000
    loaded true
```

(b) Over a period of 1 minute the median connection setup time increases by 50%.

```
loaded=false
connectionSetupTimeSlidingWindowMs=
  <median connection setup time within each second
  for the last 60 seconds>
for each sample i from 1 to 60
  for each sample j greater than i
  if (j-i)/i>0.5 and concurrentViewers>10000
    loaded true
```

(c) The variation in the median connection setup time over the last minute is greater than 25%.

Remediation Scenario 5: Change Protocols.

Various protocols can be used to distribute live event streams. Different distribution protocols have different properties and a given CDN may have different limits based on which type of protocol is being used. For example, some CDNs have larger HTTP networks than RTMP streaming ones. Both are protocols that can be used to distribute video to Flash players, however RTMP has lower latency to the source of the video than HTTP. Given the lower latency, under normal load, the CDN streams video using RTMP. During a high load state, the system can balance between RTMP and HTTP to prevent the RMTP network from being overloaded.

Additional Embodiments

Suppose two CDNs are both streaming a live video feed of a soccer game. Each CDN has 100,000 concurrent visitors and each CDN can support a maximum of 150,000 concurrent visitors (also referred to as having a "peak concurrent viewership" of 150,000). If one of the CDNs fails (e.g., "CDN A"), at most, the other CDN ("CDN B") will only be able to accommodate half of failed CDNs viewers (assuming no changes to bitrate or protocol are made). Various policies can be used to determine which clients will ultimately be allowed to continue watching the soccer game until the first CDN recovers (or additional resources are made available). Some examples of policies that can be used to address the failover-induced synchronization are as follows. The policies can be used in conjunction with CDN failures, server failures, and other components of the environment shown in FIG. 1, such as ISPs.

1. Existing CDN B viewers can continue to watch the game at their existing bitrates. All 100,000 orphaned CDN A viewers are also allowed to watch the game via CDN B, but must do so at half the bitrate of the existing CDN B viewers.

2. Existing CDN B viewers can continue to watch the game, but at a bitrate that has been reduced by 25%. All 100,000 orphaned CDN A viewers are also allowed to watch the game via CDN B, and will likewise be served at the 25% reduced bitrate.

3. Existing CDN B viewers can continue to watch the game at their existing bitrates. CDN A viewers (up to the available capacity of CDN B) can switch to CDN B but the rate at which they make new connections cannot exceed 100 connections per second. One reason for such a rule is if viewing the soccer game requires an authentication token. While the servers of CDN B that serve content may be able to accommodate many more new connections than 100 per second, generating authentication tokens is relatively more computationally intensive and thus the token server may be a bottleneck whose limitations should be considered when addressing the failover-induced synchronization.

4. Existing CDN B viewers can continue to watch the game at their existing bitrates. CDN A viewers are offered the ability to switch to CDN B (with the full bitrate) by paying either $1 or agreeing to view additional advertisements. CDN A viewers that decline the offer are also switched to CDN B but at a lower bandwidth.

5. Premium users of CDN A are allowed to connect to CDN B and watch the game at the full bitrate. Non-premium users of CDN A are also moved to CDN B (also at full bitrate), based on available resources. Since not all users can be moved, the non-premium users are moved based on a factor, such as how long they were previously connected to CDN A (i.e., with those more invested in the game being moved sooner than those who just started watching, and/or those who have watched more games over the period of time such as the last week or month, referred to as an engagement measure), and how long they have been waiting. If more than 50,000 of the 100,000 orphaned CDN A viewers are premium users, and at least some of the existing CDN B viewers are non-premium viewers, in some embodiments the connections of the non-premium viewers of CDN B are preempted (e.g., terminated) to make room for all of the premium users. The bitrate of the non-premium viewers can also be significantly restricted to ensure that all premium users have access to the game at an appropriate level of quality.

If multiple CDNs are available to accommodate the failure of CDN A (e.g., CDN B and CDN C), a variety of approaches can be used to determine how to allocate the orphans from CDN A. As one example, as many of the orphans as possible can be redirected to CDN B until it nears capacity, and the remainder can be redirected to CDN C. As another example, a round robin approach can be used to evenly distribute the CDN A clients. As yet another example, factors such as the relative performance of CDN B and CDN C and the relative cost of CDN B and CDN C can be examined when determining how to address CDN A's orphans.

FIG. **10** illustrates an example of a process for allocating resources in a content delivery environment. In some embodiments the process shown in FIG. **10** is performed by content distribution monitor **102**. The process begins at **1002** when an initial demand from initial clients for content is serviced by assigning those initial clients a first content source. Using Example 4 above, the clients that initially access the soccer game via CDN B are serviced at **1002**. At **1004**, a determination is made that additional demand from additional clients is either present or anticipated for the first content source. In Example 4 above, such a determination would be made at **1004**, when CDN A fails (and its orphaned users will need a new way to view the game). Finally, at **1006**, an interaction with at least one of the initial clients is altered to allow the additional demand to be at least partially serviced without overloading the first content source. In Example 4 above, such an interaction is altered when an existing client of CDN B has its bitrate reduced.

Additional Information Regarding Client Configuration and Assignments

Selecting a CDN Based on External Input

Suppose a content owner distributes its content via multiple CDNs (CDN A and CDN B). Suppose also that the content owner's desire is to use CDN A for clients in the United States and CDN B for clients in Europe. When a client connects to the content owner's website to view a stream, an entity on the server side determines the location of the client and returns the CDN that the client should used based on its location. This technique can also be used with respect to more sophisticated policies, such as ones based on content being watched, or user class (e.g., premium versus regular).

Selecting a CDN Based on External Input of Multiple CDNs

FIG. **11** illustrates an example of an environment in which content is distributed. In the example shown, a content owner distributes its content via multiple CDNs—CDN A (**1108**) and CDN B (**1110**). Suppose the content owner's policy is to use the CDN that provides the best quality for a client at the current time. When client A (**1112**) connects to the content owner's website to view a stream, an entity at the backend determines the CDN that is expected to provide the best quality to client A. The backend entity (**1114**) determines this using performance information from other clients such as clients B (**1116**) and C (**1118**). The backend entity sends the list of CDNs to client A in the preferred order based on the expected performance for client A. Client A then chooses the CDN based on this ordered list and local state. In some embodiments the local state keeps a black list of CDNs specific to client A based on attempts made by client A to each CDN. One reason for doing this is for when a CDN is expected to perform well for a client based on information known to the backend entity, but does not perform well in the particular case for a client. Once client A chooses a CDN, it connects and streams from that CDN.

Selecting Bit Rate Based on External Input and Local State

Suppose a content owner distributes its content using multiple bitrates (e.g., 300 kbps, 700 kbps, 1500 kbps, and 2700 kbps). Suppose also that the content owner wants to make all bitrates available to its premium customers in country A, but only make the 300 kbps and 700 kbps bitrates available to its regular customers in country A. In country B, the content owner wants to implement similar rules, but not provide the 2700 kbps bitrate to any customer. For security reasons, the knowledge of whether a customer is premium or not is maintained at the content owner's servers rather than at the client. One example of a premium customer is one who pays more for a higher class of service. When a client connects to the content owner's servers to stream the video, an entity on the server side determines the set of bitrates available to the client based on its location and service class and returns this set. The client then selects one of the available bitrates based on local state on the highest bitrate it can play without experiencing quality problems. This state can be maintained at the client based on previous viewings of content on this site or different sites.

Placing an Advertisement on a Page Based on External Input

Suppose a content owner wants to experiment with whether placing an advertisement on the left side of the page or the right side of the page has a better chance of the user clicking on the advertisement. When a client connects to the content owner's website, an entity on the server side determines which side the client should place its ad. The client places its ad in the specified location and reports back if the ad was ever clicked.

Treating Inputs from an External Entity as Commands

Suppose a content owner distributes content via multiple content distribution networks (CDN A and CDN B). Suppose also that the content owner wants to tightly control the traffic usage of the two CDNs and wants to change the usage between the CDNs based on quality and pricing in a continuous manner. In an extreme scenario the content owner may wish to turn off a CDN completely and migrate all users to the other CDN within a short period of time (e.g. a few minutes). To achieve this, the client periodically (e.g., every minute) sends a request to the content owner's website. An entity on the sever side determines for each client the CDN it should be connected to based on current policy settings. On the response to the first request from a client, the client will connect to the CDN provided. On subsequent responses, if the client is connected to a different CDN than the one returned, then it will immediately switch to the one returned. Here the client treats the response as a command to make sure the policies in the back end are enforced.

Treating Inputs from an External Entity as Hints

Suppose a content owner distributes content using multiple content distribution networks (CDN A and CDN B) and multiple bitrates (300 kbps, 700 kbps, and 1500 kbps). Suppose also that the player used by the content owner automatically adjusts the bitrate to the highest bit rate that the client can stream from the current CDN but does not switch between CDNs if streaming is working well. With this setup, it is possible that the client will get into the scenario where it plays the 700 kbps bitrate well on CDN A but cannot play the 1500 kbps on CDN A. Now suppose that the client could play at 1500 kbps at CDN B, but the client does not know this. In various embodiments, a "hint" from an external entity that has knowledge of this possibility (e.g., through inference based on other clients) can be sent to this client letting the client know that it may be able to play a higher bit rate on CDN B. The client may choose to ignore or take this hint based on local state such as user settings.

Decision on CDN is Updated Periodically

Suppose a content owner distributes its content via multiple CDNs (CDN A and CDN B). Suppose also that the content owner's policy is to use the CDN that provides the best quality for a client at the current time. When client A connects to the content owner's web site to view a stream, an entity at the backend determines the CDN that is expected to provide the best quality to client A. The backend entity determines this using performance information from other clients (in this example clients B and C). The backend entity sends the list of CDNs to client A in the preferred order based on the expected performance for client A. Client A then chooses a CDN based on this ordered list and local state. Once client A chooses a CDN, it connects to and streams from that CDN. Network and CDN performance change over time and the CDN selection decision is updated periodically accordingly. Client A periodically (e.g., once every one minute) requests a new list of CDNs from the backend entity. If the backend entity determines that the CDN currently being used by client A is no longer best suited for client A, it will return a list with a different CDN as the most preferred CDN. When client A receives this new list, it makes a decision using the new list and local state on whether to stay on the current CDN or switch to the new one. If it decides to switch, it will immediately switch to the new most preferred CDN.

Client Decision when Communication with Backend Entity is Lost

Suppose a content owner distributes its content via multiple CDNs (CDN A and CDN B). Suppose also that the content owner's policy is to use the CDN that provides the best quality for a client at the current time. When client A connects to the content owner's website to view a stream, an entity at the backend determines the CDN that is expected to provide the best quality to client A. The backend entity determines this using performance information from other clients (in this example clients B and C). The backend entity sends the list of CDNs to client A in the preferred order based on the expected performance for client A. Client A then chooses the CDN based on this ordered list and local state. Once client A chooses a CDN, it connects to and streams from that CDN. During normal operation, if client A detects a quality problem with the CDN it is streaming from, it notifies the backend entity to get a different CDN. However, in the event it loses connectivity with the backend entity, the client uses the next CDN in the most recent list provided by the backend entity to select the next CDN to try.

The control module can use the information collected from the clients to determine the correlation between the viewer engagement and quality. Alternatively, an operator can use the historical or real-time information to determine this correlation.

Enforcing Policies

In various embodiments, content owners and other appropriate parties are granted access to a policy engine that allows them to configure and revise a list of policies. Policies provided by the customer are based on the following pattern:

1. matching rule A: action A
2. matching rule B: action B
3. matching rule C: action C

The matching rule determines the subset of the viewers on which the policy is to be applied. In addition, multiple matching rules can be provided for certain policies according to a specified priority order. When the policy engine

needs to decide an action for a viewer, it iterates through the ordered list and based on the first match selects the appropriate action.

The matching rules are composed of predicates on several dimensions that identify the viewers. Example dimensions include:

1. Location of the viewer (e.g., Country. State, City. Zip code).

2. The ISP and AS through which the viewer is connected to the Internet.

3. Technographics of the viewer (e.g., browser, OS, network speed)

4. Content being watched by the viewer (e.g.. name of video, play list name, etc.)

Example actions include:

1. Maintain uninterrupted viewing by switching between an ordered list of resources (e.g., Data centers, CDNS) via which the viewer downloads the video. This provides high availability of the connection of the viewer to the content.

2. Maintain high quality viewing by switching in accordance with an ordered list of resources via which the viewer downloads the video. This provides a glitch free viewing experience (e.g., minimizes the impact of overloaded delivery servers, or network path congestion).

3. Optimize video resolution by switching the bitrate of the video. This allows the download bandwidth of the viewer to provide the highest resolution possible.

4. Optimize video viewing quality given limited bandwidth available to the user by using a lower bitrate video. This provides consistent delivery of content even if the bandwidth available to the user is low.

5. Minimize cost of delivery by switching the viewer to lower delivery cost resource if the quality of download offered by the lower cost resource is consistent.

6. Maintain load balance between a list of available resources. The load balancing is done based on priority weights associated with each resource in the list.

7. Enforce a usage limit on a resource by forcing viewers to switch away after a bandwidth or downloaded byte threshold is met.

As mentioned above, in various embodiments, content distribution monitor **102** includes a control module or works in conjunction with a separate control system. The control system is configured to enforce policies specified via the policy engine. As one example, in some embodiments whenever an initial client request is made for content. the control system determines an appropriate content source for the client based on the policies and also based on telemetry information made available via content distribution monitor **102**.

Although the foregoing embodiments have been described in some detail for purposes of clarity of understanding, the invention is not limited to the details provided. There are many alternative ways of implementing the invention. The disclosed embodiments are illustrative and not restrictive.

What is claimed is:

**1**. A system, comprising:

one or more hardware processors configured to:

determine a load threshold beyond which a component in a content delivery environment will degrade;

collect, from a set of remote clients, quality information associated with receipt of audiovisual content;

detect, based at least in part on an evaluation of at least a portion of the quality information collected from the set of remote clients, a potential occurrence of an amount of synchronized requests for the audiovisual content that has a potential to cause the load threshold for the component to be exceeded; and

in response to detecting. based at least in part on the evaluation of the at least portion of the quality information collected from the set of remote clients, the potential occurrence of the amount of synchronized requests for the audiovisual content that has the potential to cause the load threshold for the component to be exceeded, prevent the load threshold from being violated at least in part by performing one or more remediation actions, wherein performing the one or more remediation actions comprises modifying, for a remote client, at least one of a protocol with which the remote client obtains audiovisual content. a source from which the remote client obtains audiovisual content. a retry rate, and a bitrate at which the remote client obtains audiovisual content; and

a physical memory coupled to the one or more hardware processors and configured to provide the one or more hardware processors with instructions.

**2**. The system of claim **1** wherein the detecting is based at least in part on load information associated with the component.

**3**. The system of claim 1 wherein the load threshold is determined based at least in part on capacity information associated with the component.

**4**. The system of claim 1 wherein the determined load threshold includes one of a maximum number of concurrent connections, an available amount of bandwidth. and a rate at which new connections can be accepted.

**5**. The system of claim **1** wherein the detecting includes analyzing at least one of real-time performance and load data to determine whether one or more performance metrics are indicative of degradation.

**6**. The system of claim **5** wherein the one or more performance metrics includes at least one of buffering quality, connection failures, connection interruptions, time taken to setup a connection, and time taken to start streaming.

**7**. The system of claim **1** wherein the one or more hardware processors are further configured to collect at least one of peak concurrent viewer information, join information. join failure information. connection interruption information. and bitrate switching information.

**8**. The system of claim **1** wherein the detecting is based at least in part on historical information and wherein the one or more remediation actions are performed based at least in part on a prediction using the historical information.

**9**. The system of claim **8** wherein the historical information is associated with at least one of client patterns and load patterns associated with the component.

**10**. The system of claim 1 wherein the detecting is based at least in part on information received from a server within a content delivery network.

**11**. The system of claim **10** wherein the server comprises an authentication server.

**12**. The system of claim **1** wherein the synchronized requests comprise at least one of viewer-induced requests, failure-induced requests, and player-induced requests.

**13**. The system of claim **1** wherein at least one of the remediation actions treat premium clients preferentially from non-premium clients.

**14**. The system of claim **1** wherein at least one of the remediation actions treat clients with longer engagements preferentially over clients with shorter engagements.

15. The system of claim 1 wherein at least one of the remediation actions treat clients having longer wait times preferentially over clients having shorter wait times.

16. The system of claim 1 wherein performing the one or more remediation actions comprises sending a command to a client.

17. The system of claim 1 wherein performing the one or more remediation actions comprises instructing the remote client to perform at least one of changing a protocol with which it obtains content, switching a source from which it obtains content, reducing a retry rate, and reducing a bitrate.

18. A method, comprising:

determining a load threshold beyond which a component in a content delivery environment will degrade;

collecting, from a set of remote clients, quality information associated with receipt of audiovisual content:

detecting, based at least in part on an evaluation of at least a portion of the quality information collected from the set of remote clients, and using one or more hardware processors, a potential occurrence of an amount of synchronized requests for the audiovisual content that has a potential to cause the load threshold for the component to be exceeded; and

in response to detecting, based at least in part on the evaluation of the at least portion of the quality information collected from the set of remote clients, the potential occurrence of the amount of synchronized requests for the audiovisual content that has the potential to cause the load threshold for the component to be exceeded, preventing the load threshold from being violated at least in part by performing one or more remediation actions, wherein performing the one or more remediation actions comprises modifying, for a remote client, at least one of a protocol with which the remote client obtains audiovisual content, a source from which the remote client obtains audiovisual content, a retry rate, and a bitrate at which the remote client obtains audiovisual content.

19. A computer program product embodied in a non-transitory computer readable storage medium and comprising computer instructions for:

determining a load threshold beyond which a component in a content delivery environment will degrade;

collecting, from a set of remote clients, quality information associated with receipt of audiovisual content:

detecting, based at least in part on an evaluation of at least a portion of the quality information collected from the set of remote clients, a potential occurrence of an amount of synchronized requests for audiovisual content that has a potential to cause the load threshold for the component to be exceeded: and

in response to detecting, based at least in part on the evaluation of the at least portion of the quality information collected from the set of remote clients, the potential occurrence of the amount of synchronized requests for the audiovisual content that has the potential to cause the load threshold for the component to be exceeded, preventing the load threshold from being violated at least in part by performing one or more remediation actions, wherein performing the one or more remediation actions comprises modifying, for a remote client, at least one of a protocol with which the remote client obtains audiovisual content, a source from which the remote client obtains audiovisual content, a retry rate, and a bitrate at which the remote client obtains audiovisual content.

20. The method of claim 18 wherein the detecting is based at least in part on load information associated with the component.

21. The method of claim 18 wherein the load threshold is determined based at least in part on capacity information associated with the component.

22. The method of claim 18 wherein the determined load threshold includes one of a maximum number of concurrent connections, an available amount of bandwidth, and a rate at which new connections can be accepted.

23. The method of claim 18 wherein the detecting includes analyzing at least one of real-time performance and load data to determine whether one or more performance metrics are indicative of degradation.

24. The method of claim 23 wherein the one or more performance metrics includes at least one of buffering quality, connection failures, connection interruptions, time taken to setup a connection, and time taken to start streaming.

25. The method of claim 18 further comprising collecting at least one of peak concurrent viewer information, join information, join failure information, connection interruption information, and bitrate switching information.

26. The method of claim 18 wherein the detecting is based at least in part on historical information and wherein the one or more remediation actions are performed based at least in part on a prediction using the historical information.

27. The method of claim 26 wherein the historical information is associated with at least one of client patterns and load patterns associated with the component.

28. The method of claim 18 wherein the detecting is based at least in part on information received from a server within a content delivery network.

29. The method of claim 28 wherein the server comprises an authentication server.

30. The method of claim 18 wherein the synchronized requests comprise at least one of viewer-induced requests, failure-induced requests, and player-induced requests.

31. The method of claim 18 wherein at least one of the remediation actions treat premium clients preferentially from non-premium clients.

32. The method of claim 18 wherein at least one of the remediation actions treat clients with longer engagements preferentially over clients with shorter engagements.

33. The method of claim 18 wherein at least one of the remediation actions treat clients having longer wait times preferentially over clients having shorter wait times.

34. The method of claim 18 wherein performing the one or more remediation actions comprises sending a command to a client.

35. The method of claim 18 wherein performing the one or more remediation actions comprises instructing the remote client to perform at least one of changing a protocol with which it obtains content, switching a source from which it obtains content, reducing a retry rate, and reducing a bitrate.

* * * * *