US008019885B2

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

(10) **Patent No.: US 8,019,885 B2**
(45) **Date of Patent: Sep. 13, 2011**

(54) **DISCONTINUOUS DOWNLOAD OF MEDIA FILES**

(75) Inventors: **Michael Yu**, Renton, WA (US); **Ryan Edward Cukierman**, Redmond, WA (US); **Stephen Michael Lacy**, Mountain View, CA (US)

(73) Assignee: **Google Inc.**, Mountain View, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 498 days.

(21) Appl. No.: **11/620,468**

(22) Filed: **Jan. 5, 2007**

(65) **Prior Publication Data**

US 2007/0162611 A1 Jul. 12, 2007

Related U.S. Application Data

(63) Continuation-in-part of application No. 11/428,319, filed on Jun. 30, 2006.

(60) Provisional application No. 60/756,787, filed on Jan. 6, 2006.

(51) **Int. Cl.**
G06F 15/16 (2006.01)
G06F 15/173 (2006.01)
G06F 15/177 (2006.01)

(52) **U.S. Cl.** **709/232**; 709/231; 725/34; 715/723

(58) **Field of Classification Search** 709/232, 709/203, 231, 217–229; 725/34; 715/723
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,122,662 A 9/2000 Emura
6,523,027 B1 2/2003 Underwood
6,792,468 B1 * 9/2004 Bloch et al. 709/231
6,931,658 B1 8/2005 Kawamura et al.
6,980,594 B2 * 12/2005 Wang et al. 375/240.12
7,042,844 B2 5/2006 Rasanen et al.
7,096,271 B1 8/2006 Omoigui et al.
7,136,875 B2 11/2006 Anderson et al.
7,370,144 B2 * 5/2008 Su et al. 711/113
7,386,573 B2 * 6/2008 Davis 369/30.06
7,519,717 B2 4/2009 Stanford-Clark et al.
7,558,869 B2 7/2009 Leon et al.
7,581,019 B1 8/2009 Amir et al.
7,669,121 B2 2/2010 Kiilerich
(Continued)

FOREIGN PATENT DOCUMENTS

EP 1492021 A2 12/2004
(Continued)

OTHER PUBLICATIONS

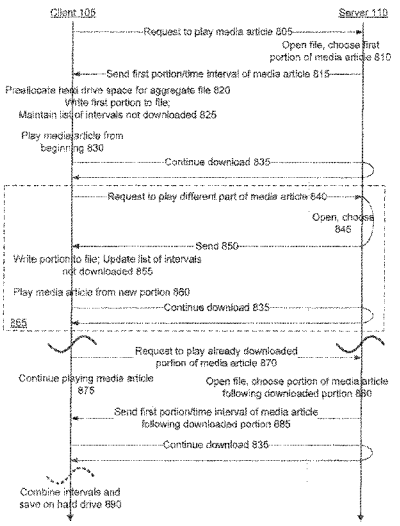
YouTube, Jun. 14, 2005, pp. 1-10, www.youtube.com.*
(Continued)

Primary Examiner — Haresh N Patel
(74) *Attorney, Agent, or Firm* — Fenwick & West LLP

(57) **ABSTRACT**

Systems and methods provide for discontinuous download of media files. The system and method work within the bounds of simple, existing, open protocols and the media files served are playable by standard media playback clients. The method is driven by a request to play a media file from any location within the media file, including sections of the media file that the initial download has not yet reached. The method comprises downloading the media file in segments corresponding to the location in the media file that the user desires to view and merging the segments. The method allows for tracking of which segments have been downloaded and which have not.

15 Claims, 9 Drawing Sheets



U.S. PATENT DOCUMENTS

7,683,940	B2 *	3/2010	Fleming	348/222.1
7,783,653	B1 *	8/2010	Manapragada et al.	707/758
2001/0004417	A1	6/2001	Narutoshi	
2002/0103920	A1 *	8/2002	Berkun et al.	709/231
2002/0131496	A1	9/2002	Vasudevan et al.	
2002/0136298	A1	9/2002	Anantharamu et al.	
2004/0098659	A1	5/2004	Bjerke et al.	
2004/0116183	A1	6/2004	Prindle	
2004/0187160	A1	9/2004	Cook et al.	
2004/0267908	A1 *	12/2004	Doi et al.	709/219
2004/0267952	A1	12/2004	He et al.	
2004/0267956	A1	12/2004	Leon et al.	
2005/0021830	A1	1/2005	Urzaiz et al.	
2005/0028213	A1	2/2005	Adler et al.	
2005/0066063	A1 *	3/2005	Grigorovitch et al.	710/1
2005/0141886	A1	6/2005	Srinivasan et al.	
2005/0144284	A1 *	6/2005	Ludwig et al.	709/226
2005/0183017	A1 *	8/2005	Cain	715/719
2005/0216839	A1 *	9/2005	Salvucci	715/723
2005/0254374	A1	11/2005	Lin	
2006/0002681	A1 *	1/2006	Spilo et al.	386/46
2006/0009983	A1	1/2006	Magliaro et al.	
2006/0037057	A1 *	2/2006	Xu	725/90
2006/0129909	A1 *	6/2006	Butt et al.	715/500.1
2006/0221788	A1	10/2006	Lindahl et al.	
2006/0235654	A1	10/2006	Parnes	
2006/0279628	A1 *	12/2006	Fleming	348/143
2007/0011012	A1	1/2007	Yurick et al.	
2007/0011343	A1	1/2007	Davis et al.	
2007/0033225	A1 *	2/2007	Davis	707/104.1
2007/0033494	A1	2/2007	Wenger et al.	
2007/0038669	A1 *	2/2007	Davis	707/104.1
2007/0091815	A1	4/2007	Tinnakornsrisuphap et al.	
2007/0100974	A1	5/2007	Standford-Clark et al.	
2007/0121728	A1	5/2007	Wang et al.	
2007/0124298	A1 *	5/2007	Agrawal	707/5
2007/0153916	A1	7/2007	Demircin et al.	
2007/0157228	A1 *	7/2007	Bayer et al.	725/34
2007/0168254	A1	7/2007	Steelberg et al.	
2007/0169146	A1	7/2007	Steelberg et al.	
2007/0192789	A1	8/2007	Medford	
2007/0271388	A1	11/2007	Bowra et al.	
2007/0299870	A1	12/2007	Finch	
2008/0092159	A1 *	4/2008	Dmitriev et al.	725/34
2008/0115161	A1 *	5/2008	Kurzion	725/32
2008/0167957	A1	7/2008	Steelberg et al.	
2008/0212575	A1	9/2008	Westberg et al.	
2009/0131141	A1	5/2009	Walker et al.	

FOREIGN PATENT DOCUMENTS

WO	WO 99/24987	A1	5/1999

OTHER PUBLICATIONS

PCT International Search Report and Written Opinion, PCT/US2007/000403, Jun. 29, 2007, 15 pages.

"Working Together with MXF," Pro-MPEG Forum, Sep. 3, 2002, pp. 1-4.

Almeroth, K. C., et al., "On the Performance of a Multicast Delivery Video-On-Demand Service with Discontinuous VCR Actions," CiteSeer, 2 pages, [online] [retrieved on Dec. 22, 2006] Retrieved from the Internet: <URL: http://citeseer.ist.psu.edu/almeroth95performance.html>.

"Eyespot Releases Version 2.0 of Online Video Mixing Community at Under the Radar Conference," Jun. 14, 2006, [online] [Retrieved on Jun. 29, 2006] Retrieved from the Internet: <URL: http://www.eyespot.com/press/.>, p. 1.

"Eyespot—Shoot, Mix, and Share Your Video," Mar. 13, 2006 [online] [Retrieved on Jun. 29, 2006]Retrieved from the Internet: <URL: http://www.solutionwatch.com/326/eyespot-shoot-mix-and-share-your-video/>, p. 1.

"Jumpcut Makes Movies Simple," MiraVlda Media, Inc., [online] [Retrieved on Jun. 29, 2006] Retrieved from the Internet: <URL: http://www.jumpcut.com/company/>, p. 1.

Kwon, J. B., et al., "Providing VCR Functionality in Staggered Video Broadcasting," CiteSeer, 2 pages, [online] [retrieved on Dec. 22, 2006] Retrieved from the Internet: <URL: http://citeseer.ist.psu.edu/590569.html>.

Marlowe, C., "Eyespot in the Mix with Legal Video Mash-Ups," Washingtonpost.com, Jun. 18, 2006, [online] [Retrieved on Jun. 29, 20006] Retrieved from the Internet: <URL: http://www.washingtonpost.com/wp-dyn/content/article/2006/06/18/AR2006061800966.html>, p. 1.

"Microsoft Advanced Streaming Format," Multimedia Wiki, last modified May 2, 2006, 2 pages, [online] [retrieved on Dec. 22, 2006] Retrieved from the Internet: <URL: http://wiki.multimedia.cx/index.php?title=Microsoft_Advanced_Streaming_Format>.

Robinson, D., "Jumpcut: Video Editing on the Web," FlashInsider, Apr. 6, 2006, [online] [Retrieved on Jun. 29, 2006] Retrieved from the Internet: <URL: http://www.flahsinsider.com/2006/04/06/jumpcut-video-editing-on-the-web/>, p. 1.

"VideoEgg: About Us," VideoEgg, Inc., 2006 [online] [Retrieved on Jun. 29, 2006] Retrieved from the Internet: <URL: http://www.videoegg.com/general/about>, p. 1.

"VideoEgg Offers New Video Upload Service," About, Inc., Sep. 19, 2005 [online] [Retrieved on Jun. 29, 2006] Retrieved from the Internet: <URL: http://desktopvideo.about.com/b/a/203840.htm>, p. 1.

Hahn, M., "Uniform Resource Locators," EDPACS, Dec. 1995, pp. 8-13.

Office Action of the European Patent Office for European Patent Application No. 07709597.4 dated Jan. 30, 2009, 6 pages.

Walpole, J., et al., "A Player for Adaptive MPEG Video Streaming Over the Internet," In Proceedings 26th Applied Imagery Pattern Recognition Workshop AIPR-97, SPIE, 1997, 12 pages, Dec. 31.

First Examiner's Report, Australian Patent Application No. 2007205046, Mar. 4, 2010, 3 pages.

Office Action for U.S. Appl. No. 11/620,346, Mar. 31, 2010, 41 pages.

Office Action for U.S. Appl. No. 11/620,346, Oct. 13, 2009, 28 pages.

Office Action for U.S. Appl. No. 11/620,354, Jan. 15, 2010, 7 pages.

Office Action for U.S. Appl. No. 11/620,354, Oct. 13, 2009, 7 pages.

Office Action for U.S. Appl. No. 11/620,386, Jan. 5, 2010, 38 pages.

Office Action for U.S. Appl. No. 11/620,354, May 18, 2010, 8 pages.

Office Action for U.S. Appl. No. 11/620,386, Jun. 9, 2010, 43 pages.

Office Action for U.S. Appl. No. 11/620,386, Nov. 24, 2010, 37 pages.

Richter, S., "'Streaming' flv video via PHP, take two," FlashConGuru.com, Nov. 2, 2005, 94 pages, [online] [retrieved on Dec. 7, 2010] Retrieved from the internet <URL:http://www.flashcomguru.com/index.cfm/2005/11/2/Streaming-flv-video-via-PHP-take-two>.

* cited by examiner

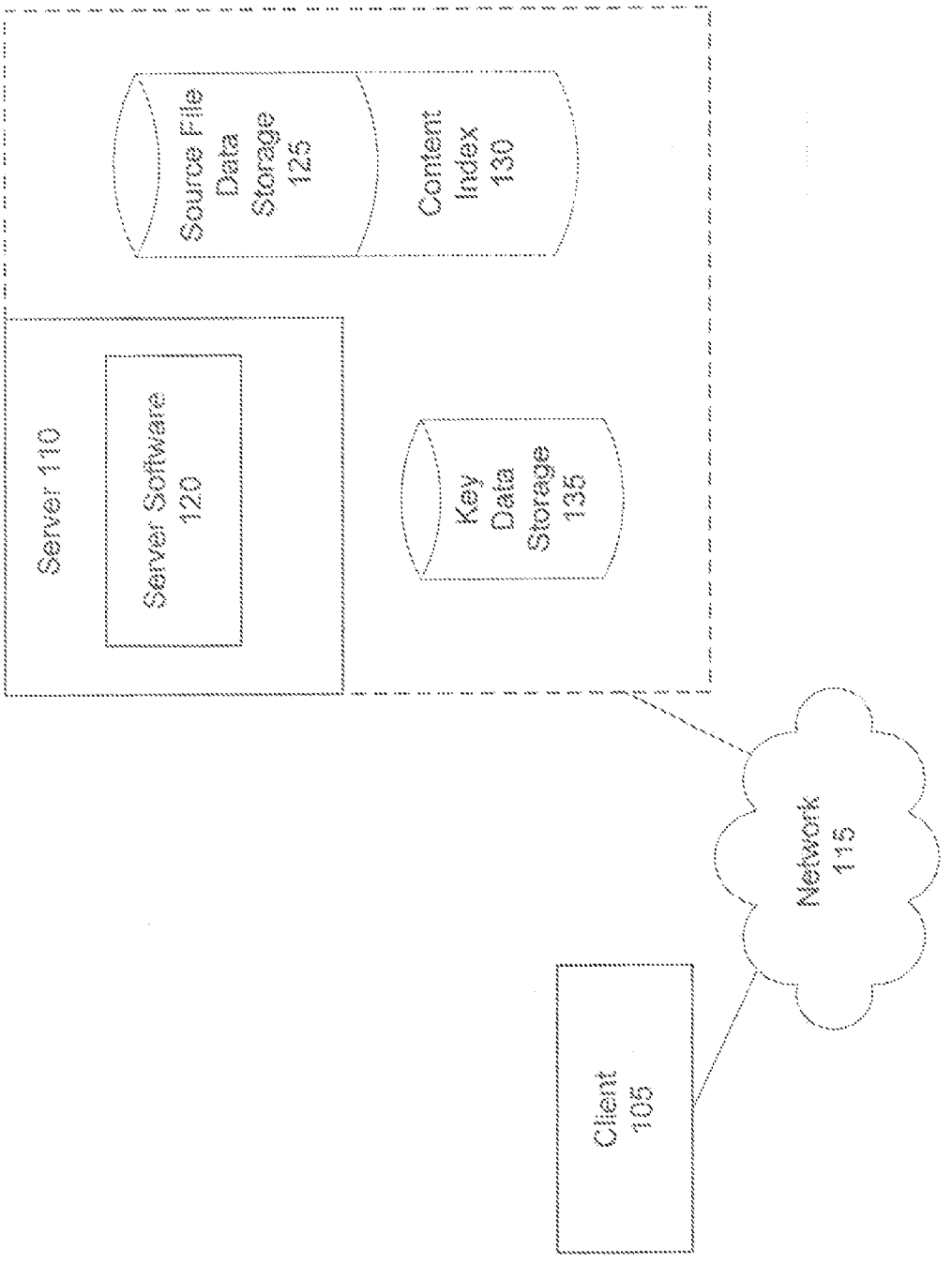

Source File Data Storage 125
Content Index 130
Server 110
Server Software 120
Key Data Storage 135
Network 115
Client 105

FIG. 1

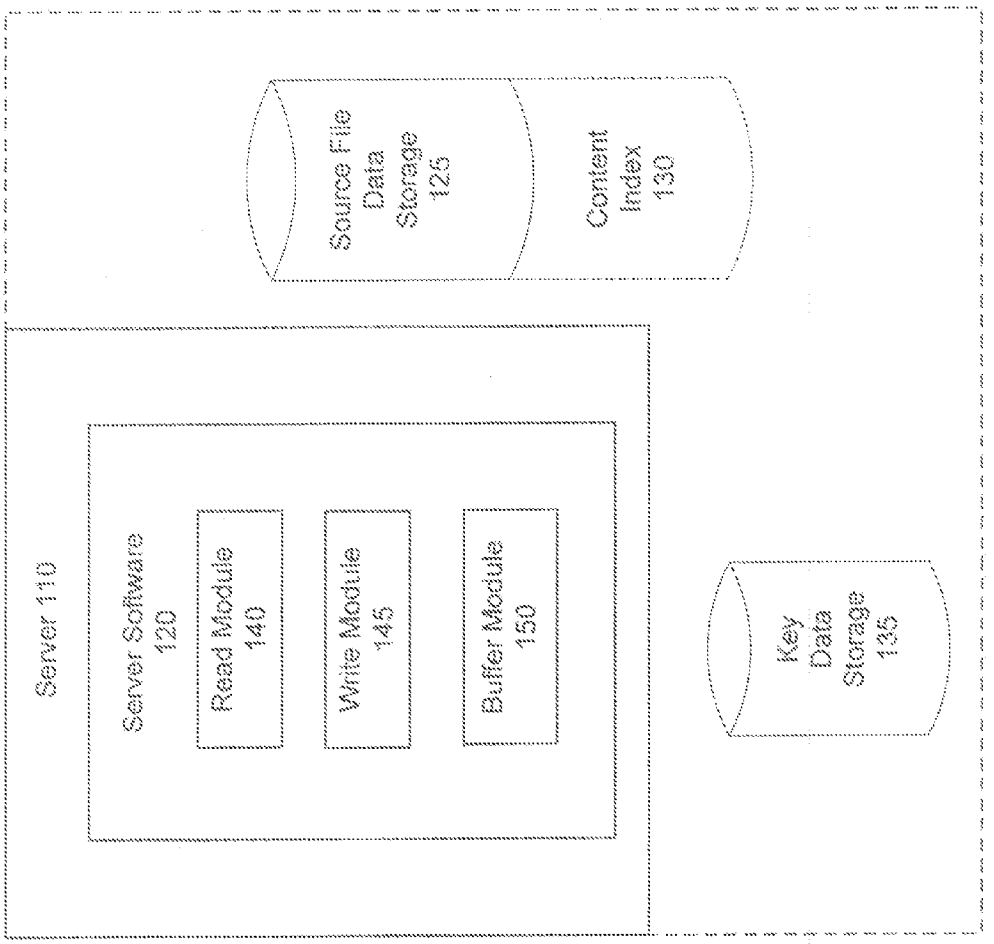

Source File Data Storage 125
Content Index 130
Server 110
Server Software 120
Read Module 140
Write Module 145
Buffer Module 150
Key Data Storage 135

FIG. 2

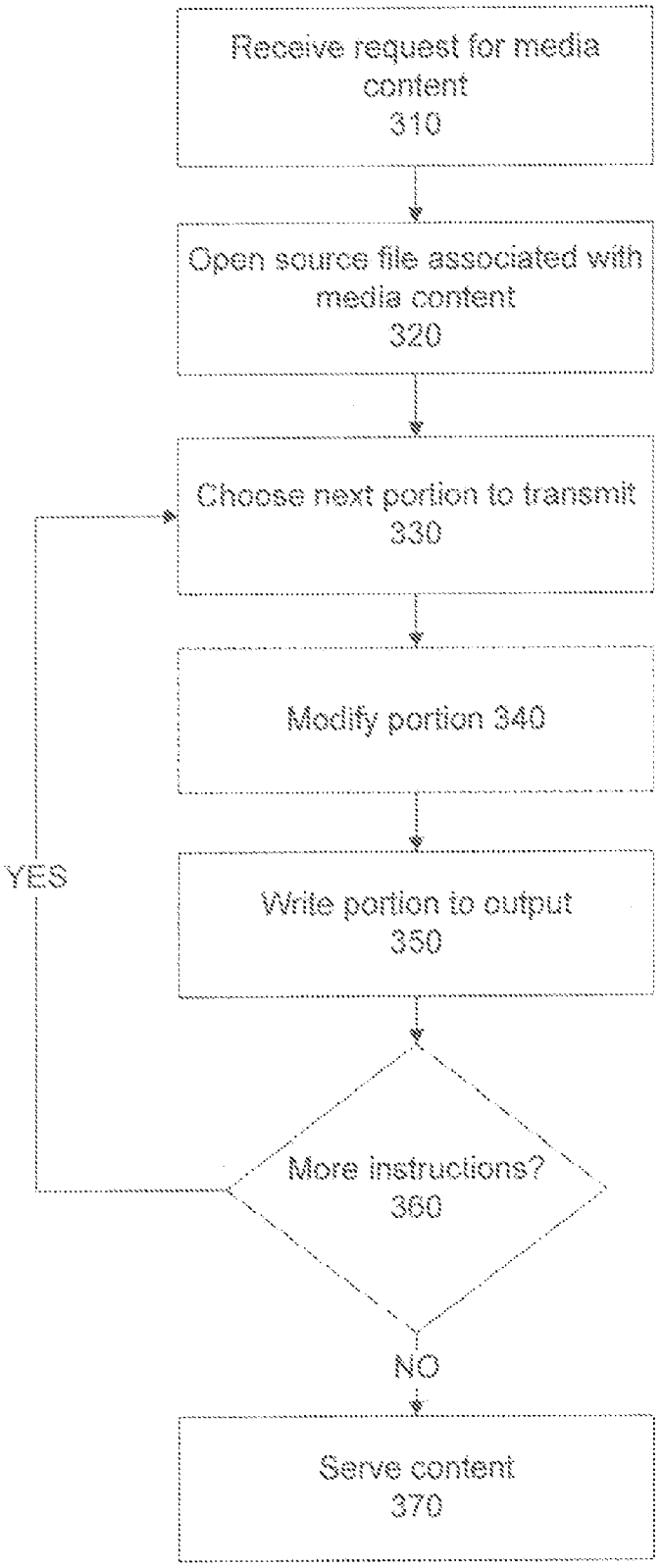

Receive request for media content
310
Open source file associated with media content
320
Choose next portion to transmit
330
Modify portion 340
Write portion to output
350
More instructions?
360
YES
NO
Serve content
370

FIG. 3

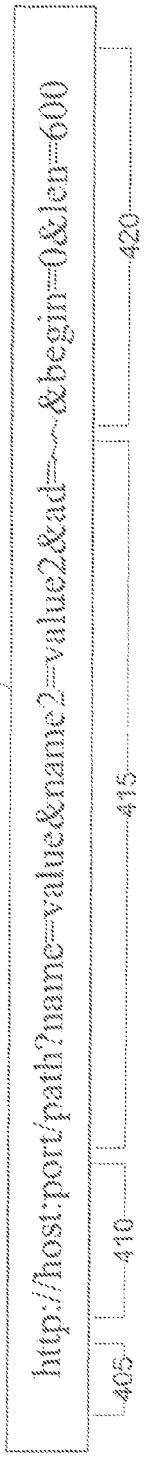


FIG. 4A

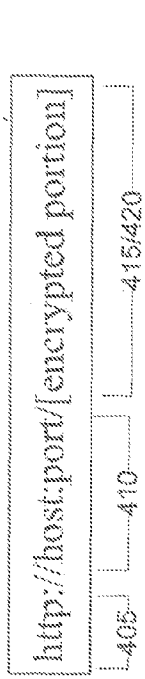


FIG. 4B

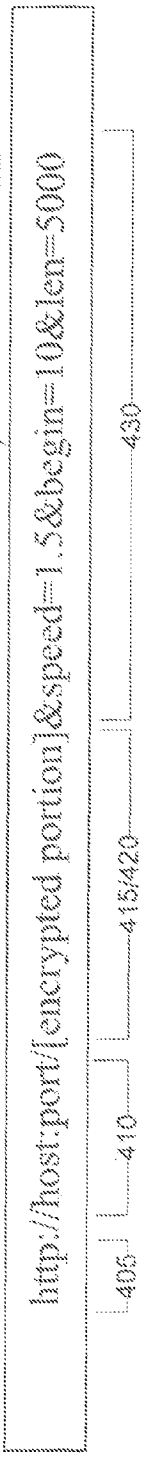


FIG. 4C

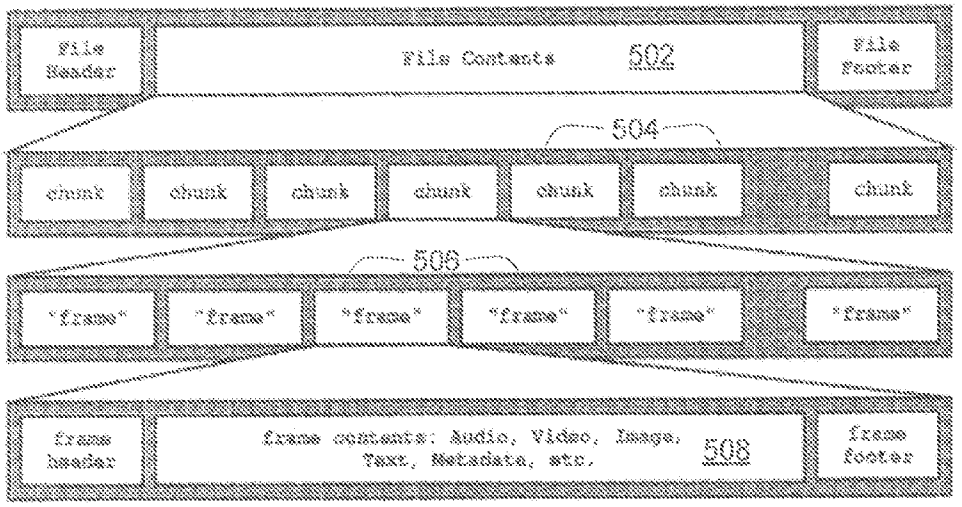

File Header
File Contents
502
File Footer
504
chunk
chunk
chunk
chunk
chunk
chunk
chunk
506
"frame"
"frame"
"frame"
"frame"
"frame"
"frame"
frame header
frame contents: Audio, Video, Image, Text, Metadata, etc.
508
frame footer

FIG. 5A

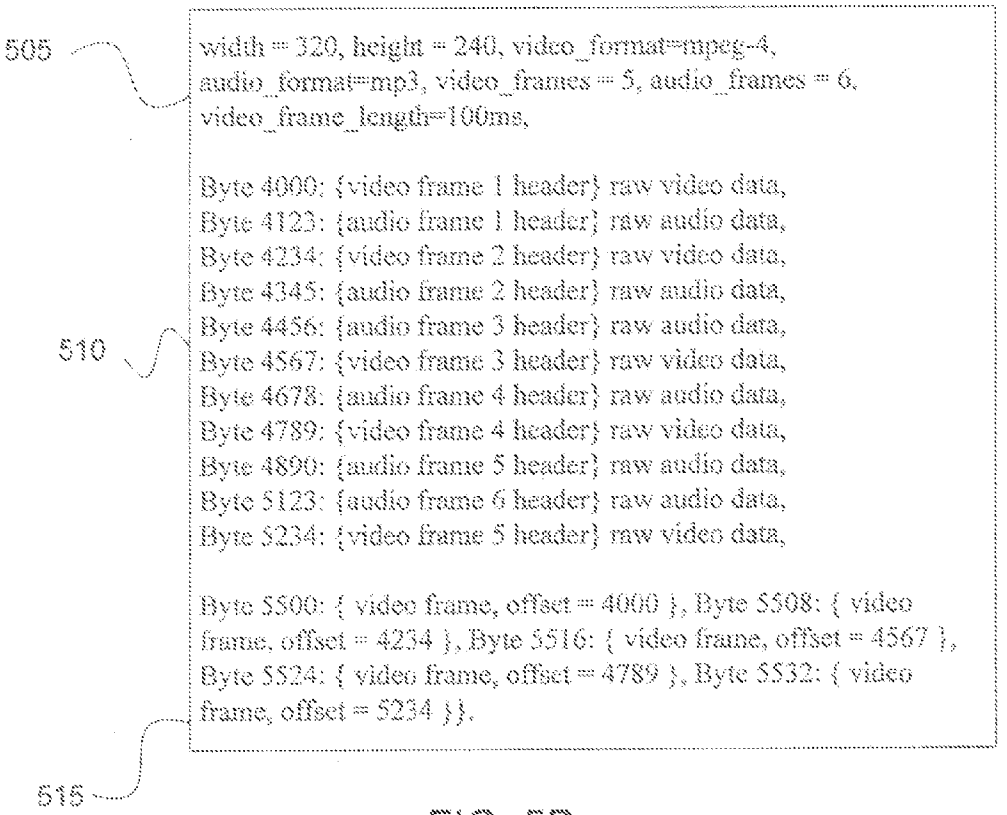

505
width = 320, height = 240, video_format=mpeg-4, audio_format=mp3, video_frames = 5, audio_frames = 6, video_frame_length=100ms,
Byte 4000: {video frame 1 header} raw video data,
Byte 4123: {audio frame 1 header} raw audio data,
Byte 4234: {video frame 2 header} raw video data,
Byte 4345: {audio frame 2 header} raw audio data,
Byte 4456: {audio frame 3 header} raw audio data,
Byte 4567: {video frame 3 header} raw video data,
Byte 4678: {audio frame 4 header} raw audio data,
Byte 4789: {video frame 4 header} raw video data,
Byte 4890: {audio frame 5 header} raw audio data,
Byte 5123: {audio frame 6 header} raw audio data,
Byte 5234: {video frame 5 header} raw video data,
510
Byte 5500: { video frame, offset = 4000 }, Byte 5508: { video frame, offset = 4234 }, Byte 5516: { video frame, offset = 4567 }, Byte 5524: { video frame, offset = 4789 }, Byte 5532: { video frame, offset = 5234 }}.
515

FIG. 5B

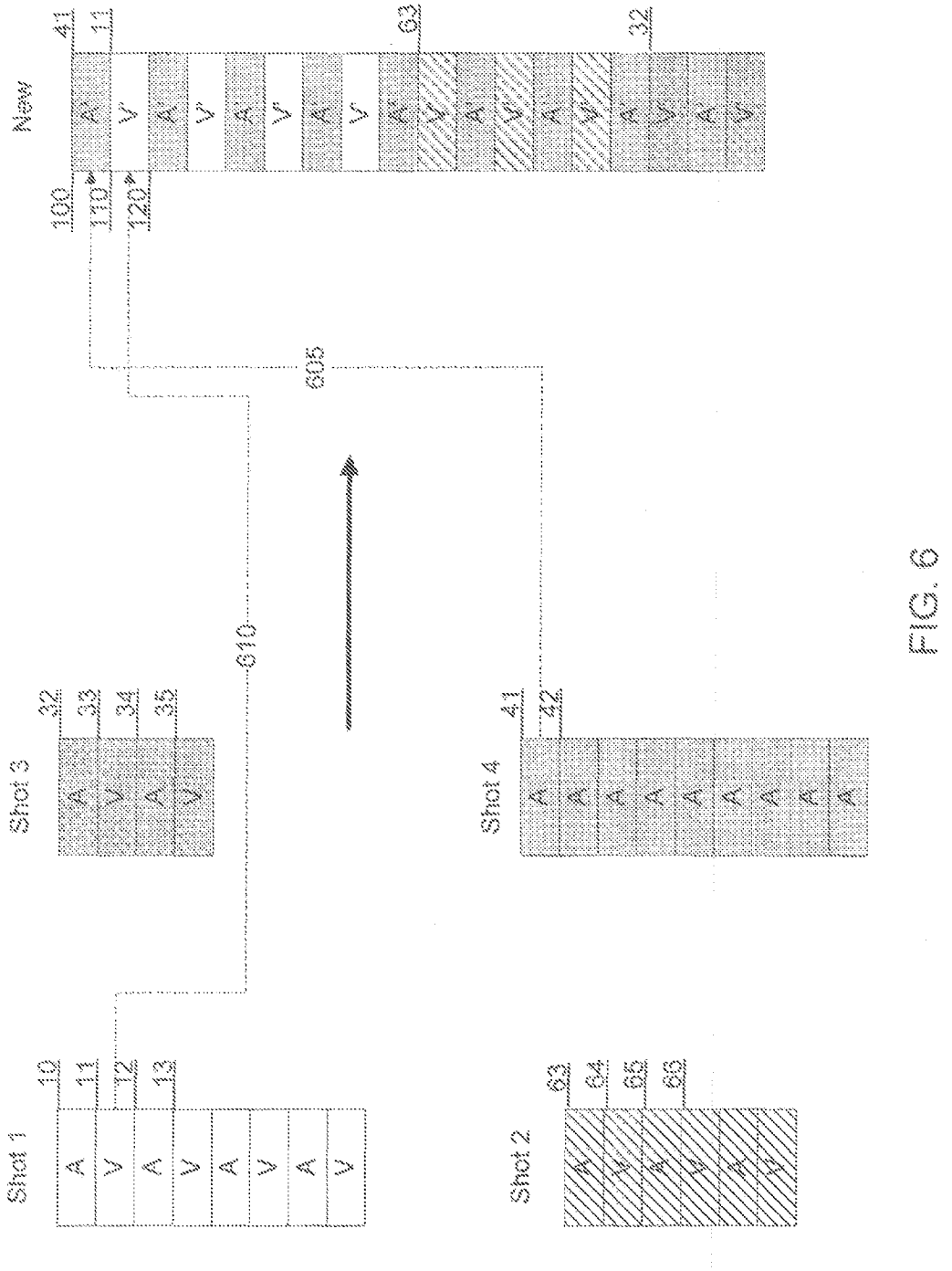

New
41
11
63
32
A
V
100
110
120
605
610
Shot 3
32
33
34
35
Shot 4
41
42
Shot 1
10
11
12
13
Shot 2
63
64
65
66

FIG. 6

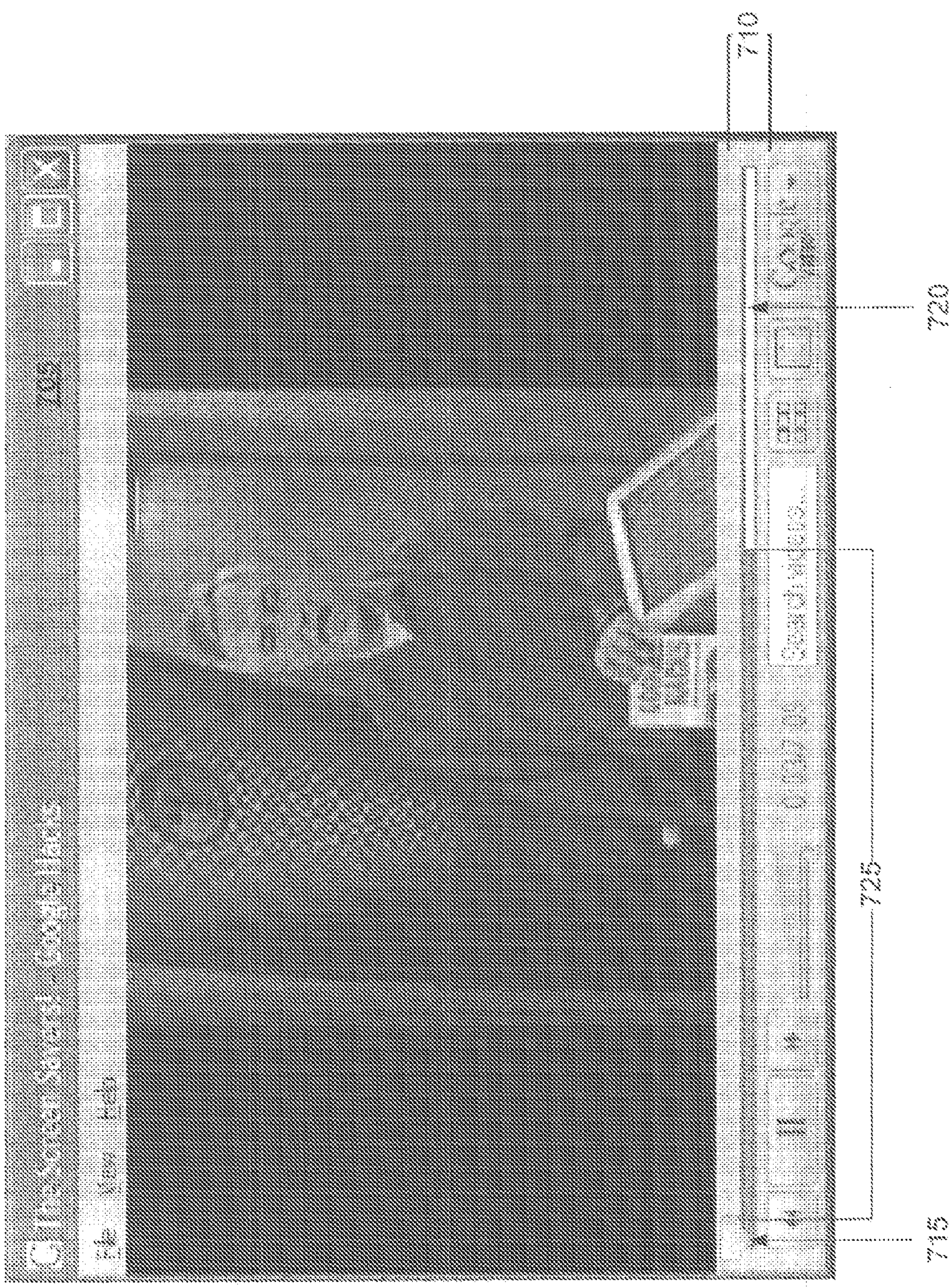


FIG. 7

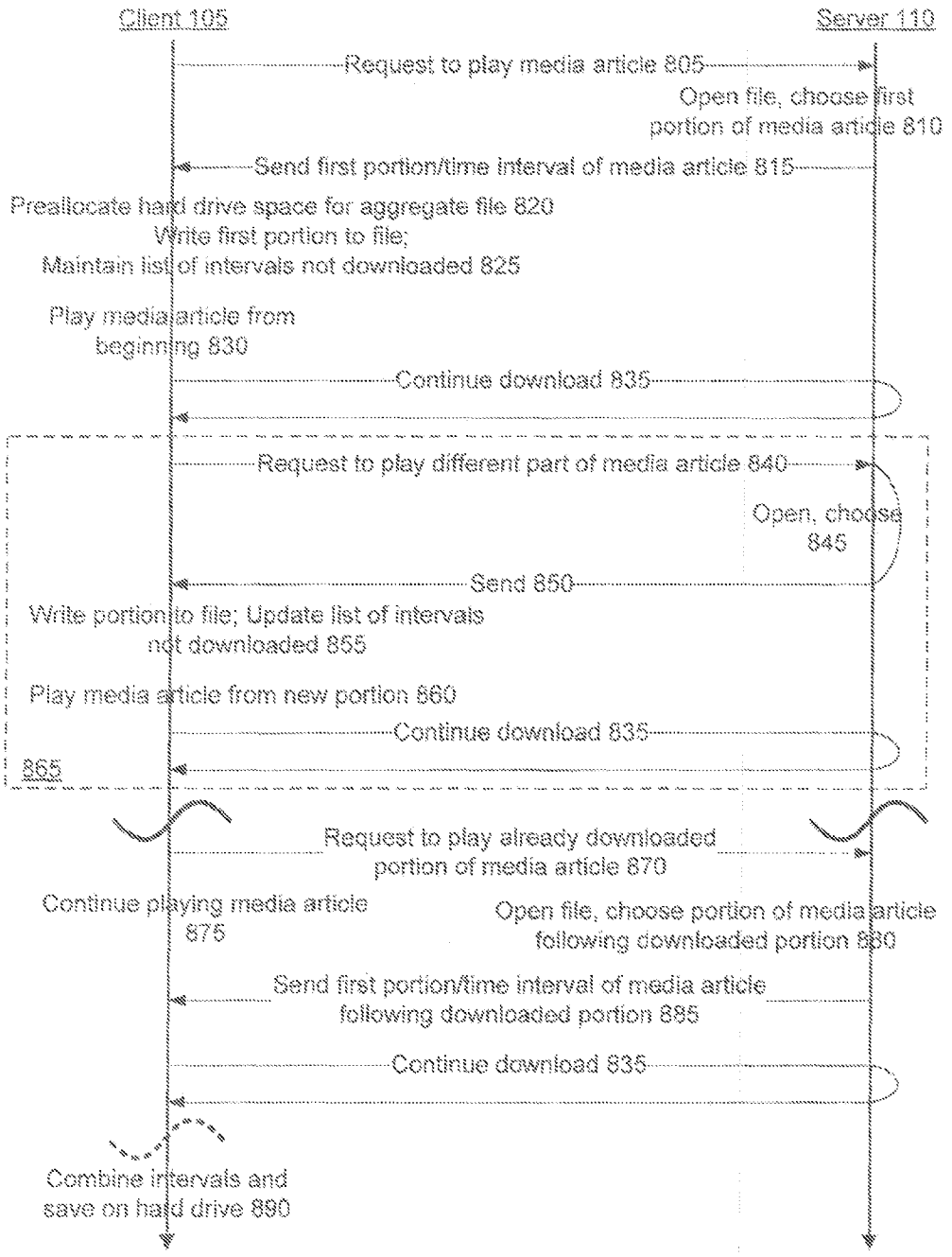

Client 105
Server 110
Request to play media article 805
Open file, choose first portion of media article 810
Send first portion/time interval of media article 815
Preallocate hard drive space for aggregate file 820
Write first portion to file;
Maintain list of intervals not downloaded 825
Play media article from beginning 830
Continue download 835
Request to play different part of media article 840
Open, choose 845
Send 850
Write portion to file; Update list of intervals not downloaded 855
Play media article from new portion 860
Continue download 835
865
Request to play already downloaded portion of media article 870
Continue playing media article 875
Open file, choose portion of media article following downloaded portion 880
Send first portion/time interval of media article following downloaded portion 885
Continue download 835
Combine intervals and save on hard drive 890

FIG. 8

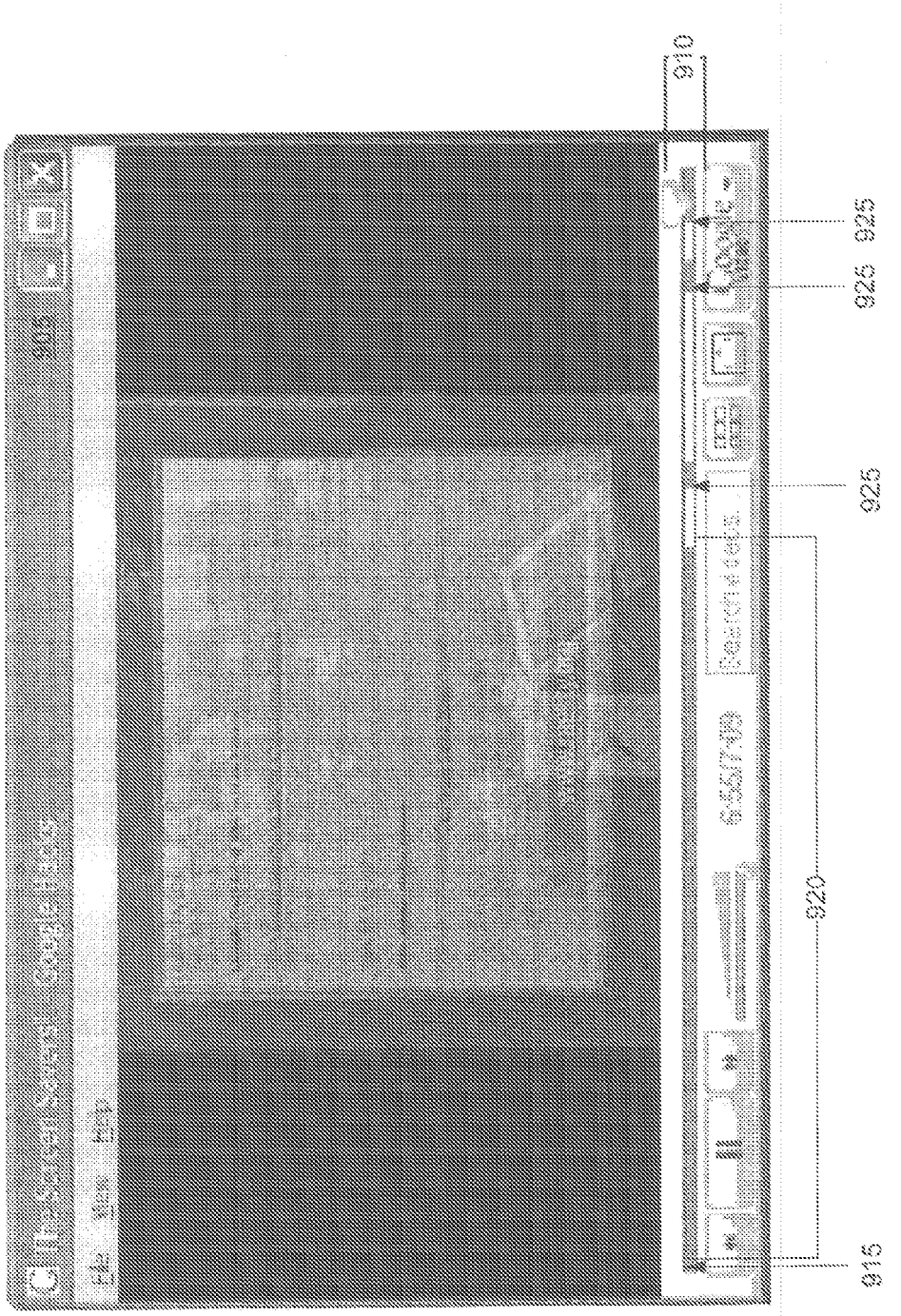


FIG. 9

DISCONTINUOUS DOWNLOAD OF MEDIA FILES

RELATED APPLICATIONS

This application claims the benefit under 35 U.S.C. §119 (e) of Provisional Patent Application Ser. No. 60/756,787, entitled "Discontinuous Download of Video Articles," filed Jan. 6, 2006. This application is a continuation-in-part under 35 U.S.C. §120 of U.S. patent application Ser. No. 11/428,319, now copending, entitled "Dynamic Media Serving Infrastructure," filed Jun. 30, 2006. This application is related to U.S. patent application Ser. No. 11/620,346, entitled "Combining and Serving Media Content", filed Jan. 5, 2007; U.S. patent application Ser. No. 11/620,354, now U.S. Pat. No. 7,840,693, entitled "Serving Media files with Altered Playback Speed", filed Jan. 5, 2007; and U.S. patent application Ser. No. 11/620,386, now copending, entitled "Media file Adaptation to Client Device", filed Jan. 5, 2007. All of the foregoing applications are incorporated herein in their entirety by reference for all purposes.

BACKGROUND

The present invention pertains in general to playing of media files, and in particular to downloading media files in segments.

Methods are known for streaming and downloading media content, for example, across the Internet from a server to a client device in response to a client request for media content. Existing technology uses a traditional static file serving interface, in which a complete file is served to a client. These technologies serve either entire files or client-requested byte ranges of files. The file is usually stored or cached on the server for playback. Typically, if a user wants to view media content in this manner, a specialized client player application for playing the media content must be downloaded and installed on a client system. The server and client player application then use specialized protocols, file formats, and video encodings to transmit, decode and playback the media content.

Media files, especially video files, are typically very large and thus take a long time to download to the end user. To avoid long delays to the end user caused by waiting for an entire media file to download, existing media player technology begins playing a media file as soon as the beginning of the media file has downloaded. FIG. **7** depicts a media player **705** using a standard media download process. The download proceeds continuously starting from the beginning of the media file, as indicated by the leftmost end **715** of the media file timeline bar **710**. The user of the media player cannot advance to a later portion of the media file, for example at time location **720**, because that portion has not yet been reached in the download process, shown by the dark bar **725**. This is particularly inconvenient in the case of a long file, which a user might prefer to preview portions of prior to allocating time and resources to download the entire video.

SUMMARY

The above problem is solved by a method for discontinuous download of media files. Using discontinuous download, the user can skip to any location in the media file and play from there, including sections of the media file that the initial download has not yet reached, and the media file will begin playing from that location with only a negligible delay. Thus, the downloading methods provide the user with the appearance and functionality of having the media file immediately available for viewing. To accomplish this, the present invention provides various embodiments of methods and systems for serving media content.

The method comprises downloading the media file in segments corresponding to the location in the media file that the user desires to view and merging the segments. The merging process may take place as the segments download, or may occur once at the end of the download. The segments are virtually merged by being stored in sequence in a preallocated storage location large enough to accommodate the entire media file. The method allows for tracking of which segments have been downloaded and which have not.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram illustrating a client-server architecture according to one embodiment of the present invention.

FIG. **2** is a block diagram illustrating the server in greater detail.

FIG. **3** is a flowchart illustrating a method for serving media content.

FIGS. **4**A-**4**C illustrate possible uniform resource locators.

FIG. **5**A is an illustration of a file format.

FIG. **5**B is an illustration of a sample AVI file.

FIG. **6** shows an example of a use case for the method described herein.

FIG. **7** is an illustration of a media player using a standard media download process.

FIG. **8** is an interaction diagram showing exchange of information between a client and a server.

FIG. **9** is an illustration of a media player according to the methods of the present invention.

The figures depict various embodiments of the present invention for purposes of illustration only. One skilled in the art will readily recognize from the following description that alternative embodiments of the structures and methods illustrated herein may be employed without departing from the principles of the invention described herein.

DETAILED DESCRIPTION OF THE EMBODIMENTS

Described herein is a method and system for serving media. Media content, as used herein, refers to any type of video and audio content formats, including movie files, i.e., files including both video and audio content, as well as audio-only formats, and may include any form of audio, video, metadata, or combination thereof. A common example is a video article including two content streams, one video stream and one audio stream. However, the techniques described herein can be used with any number of file portions or streams, and may include metadata.

The various embodiments of the invention can be implemented in the context of a standard-client server architecture. FIG. **1** is a block diagram illustrating a client server architecture suitable for this purpose Such as system comprises a client **105** and a server **110**, communicatively coupled, e.g., by a network **115**. The client **105** can be any type of client computing device, for example, a device executing a browser application or other application adapted to communicate over Internet related protocols (e.g., TCP/IP and HTTP) and/or display a user interface though which media content can be output. According to one embodiment, the user interface of the client **105** allows a user to view, manipulate, and select media content portions and sequence them to form the basis for an edit list as descried herein. The client **105** includes a

processor, an addressable memory, and other features (not illustrated) such as a display adapted to display video content, local memory, input/output ports, and a network interface. The network interface and a network communication protocol provide access to a network **115** and other computers, such as server **110** or third party computers, along with access to the Internet, via a TCP/IP type connection, or to other network embodiments, such as a LAN, a WAN, a MAN, a wired or wireless network, a private network, a virtual private network, via other networks, or other systems allowing for data communication between two or more computing systems, as well as through a combination of networks (e.g., accessing the server via a bridge from a cellular or wireless networking system to a TCP/IP system). In various embodiments the client **105** may be implemented on a computer running a Microsoft Corp. operating system, an Apple Computer Inc., Mac OS, various flavors of Linux, UNIX, Palm OS, and/or other operating systems. While only a single client **105** is shown, the system can support a large number of concurrent sessions with many clients **105**.

In one implementation, the system operates on high performance server class computers **110**. The details of the hardware aspects of servers **110** are well known to those of skill in the art and are not further described herein. The server **110** is depicted in FIG. **1** as a single computer system, however, the system may be implemented as a network of computer processors and associated storage devices. Examples of server devices **110** are servers, mainframe computers, networked computers, a processor-based device, and similar types of systems and devices. The server **110** and its associated storage devices need not be physically co-located, and there need not be any predefined mapping between storage devices and server computers. The server processor can be any of a number of well known computer processors, such as processors from Intel Corporation of Santa Clara, Calif. and Motorola Corporation of Schaumburg, Ill. In one embodiment, the server **110** has sufficient processing power and bandwidth to perform modifications of content prior to serving as described herein. The network **115** may comprise the Internet alone or in combination with other networks, wired and wireless, such as an intranet, a local area network, a wide area network, or a broadcast network according to various embodiments. Alternatively, the client and server may be located on a single device.

FIG. **2** is a block diagram illustrating one embodiment of the server **110** in greater detail. Here, the server **110** includes server software **120**, source file data storage **125**, a content index **130**, and a key data storage **135**. The source file data storage **125**, content index **130**, and key data storage **135** may also be separate from, but accessible by, the server **110**, and/or form a single data store. The server software **120** is comprised of a number of executable code portions and data files. These include code for enabling the methods described herein. The server software **120** is responsible for orchestrating the processes performed according to the methods of the present invention. The server software **120** includes a read module **140**, a write module **145**, and a buffer module **150**. The read module **140** is one means for enabling the system to use one or more read threads to read portions of a stored file. The read module **140** controls the source media file access, processing, header formation, content reading, content modification, footer construction, and writing to the output as described herein. The write module **145** is one means for enabling the server **110** to control the writing of data from the output (network transmit buffer) to the network **115**. The buffer module **150** is one means for enabling the system to prepare data for writing to the network by placing it in a network transfer buffer. The buffer module **150** controls the functioning of the network transfer buffer. The above software portions **140-150** need not be discrete software modules. The software configuration shown is meant only by way of example; other configurations are contemplated by and within the scope of the present invention. These aspects of the present invention allow the server **110** to serve media content directly from stored content, without the need to duplicate content for serving. In addition, the media files served are playable by standard media playback clients.

Stored media content can be previously received from a client **105** as a result of being uploaded by a user. In another embodiment, the media content is from one or more other sources, for example television content. Once the media content is received at the server **110**, the media file is transformed into a serving format. In this example, transforming into the serving file format includes processing the media file, identifying the indexable portions within the media file, enumerating and assigning each frame a timestamp, parsing the media file headers and footers, and computing format-specific "chunk" boundary points based on the indexable file portions, or chunks. The media content is stored in a source file data storage **125**, and data about the media content is stored out-of-band from the media content in a content index **130**, which can be queried using a content identifier. A content identifier is an arbitrary binary identifier that uniquely identifies a single piece of stored media content and data about the media content, e.g., a 64-bit hash value computed from attributes of the stored media content, any unique string of characters, or a unique number assigned to this content at the time it is created. The content identifier is used to anonymously reference media content instead of using file names, e.g., to provide greater privacy and security. The size of a portion of a file, or chunk, varies from the very small to the very large. The size of the portions of a file determines the granularity available for the editing of the file. The number of chunks stored per media stream is limited only by the size and speed of the storage mechanism used, and by the amount of acceptable metadata storage size per media file. Systems with smaller sized storage would choose a longer interval for chunks, thus reducing the storage overhead for each media file. Systems with large amounts of fast storage could choose chunk sizes down to the individual frame granularity. Preferably, the ratio of the size of the metadata to the size of the media file is small, e.g., less than 1%. Alternatively, a static size for the metadata, such as 64 kB, could be used.

The source file data storage **125** may be a relational database or any other type of database that stores the media content data according to one embodiment. Media content on the server may be received from various sources in various (e.g., compressed or uncompressed) forms. A media article can be stored in various ways and formats. For example, a media article can be stored as a file or other data in a database or other storage medium. A media article can comprise, for example, video content stored in any format, such as Moving Picture Experts Group (MPEG-2, MPEG-4), Windows Media 9 (Windows Media High Definition Video), Video for Windows (AVI), QuickTime, Indeo, etc., as well as in any resolution (e.g., HD or SD), or signal format (e.g., National Television Standards Committee (NTSC), Phase Alternating Line (PAL), System Electronique pour Couleur avec Mémorie (SECAM)), or any other suitable video content of any format and standard whatsoever. The media articles may be suitable for broadcast and/or made available on a network (such as the Internet), a personal computer, or other computing or storage means, and may include other data associated with the video content such as program guide information,

metadata, and closed captioning text. The embodiments described herein are described generally in relation to video content, and thus are referred to as video articles, but embodiments may operate on any suitable type of content, including audio-only content.

A generic container file format is used to encapsulate the underlying video or audio data. An example container file format according to one embodiment is shown in FIG. **5**A. The example generic file format includes an optional file header followed by file contents **502** and an optional file footer. The file contents **502** comprise a sequence of zero or more chunks **504**, and each chunk is a sequence of "frames" **506** (or groups, as described herein). Each frame **506** includes an optional frame header followed by frame contents **508** and an optional frame footer. A frame **506** (or group) can be of any type, either audio, video, or metadata according to one embodiment. In other embodiments, this model may be used for other file types, for example, Adobe Post Script™, PDF, mp3, or other custom file types. For temporal media, e.g., most audio or video, frames are defined by a specific (e.g., chronological) timestamp. For non-temporal media, frames are defined by any monotonically increasing value, e.g., page numbers of a PDF document. A frame or group **506** includes one or more streams of content. A stream refers to a set of frames identified by the containing file format. For example, a file may contain zero or more video streams, and zero or more audio streams. Each frame **506** has an identifier that marks it as being part of at least one stream. Some formats implicitly define streams based on frame format types.

Each frame **506** may be a keyframe according to one embodiment, as defined by the video encoding format and encoding parameters used. The media file must include one or more keyframes, and chunks may contain zero or more keyframes. Key frames are used, in part, to define segments of media content from which playback can begin. However, many audio formats do not have this feature, and can be segmented on individual audio frame boundaries. For each frame **506**, a timestamp can be computed. A timestamp need not necessarily correspond to a physical time, and should be thought of as an arbitrary monotonically increasing value that is assigned to each frame of each stream in the file. For example, as referenced above, the monotonically increasing value as applied to a PDF file would correspond to a page number. Thus, the use of the term "timestamp" is used herein in this broader context. If a timestamp is not directly available, the timestamp can be synthesized through interpolation according to the parameters of the video file. Each frame **506** is composed of data, typically compressed audio, compressed video, text meta data, binary meta data, or of any other arbitrary type of compressed or uncompressed data. Various container file formats may be used. An AVI file is described below as one possible container file format, however, other container formats, e.g., ASF, etc., may be used.

In one embodiment, an AVI file is used to store the media article. FIG. **5**B shows sample AVI file contents according to one embodiment. In this example, the AVI file includes a header **505** including basic information about the segment, e.g., width, height, video format, audio format, number of video frames, number of audio frames, length of video frame, etc. The bulk of the file lists byte content information **510**. In addition, the file ends with a footer **515** including information about byte offsets, etc. In this example every frame is a key frame.

The content index **130** may be a relational database or any other type of database that stores the indexed data corresponding to the source file media according to one embodiment. Here, the content index **130** includes a content identifier, the number of file portions or streams and stream types within the media file, the file content offset in bytes, the file footer offset in bytes, stream-specific information (e.g., frame rate, format, width, height, duration, etc.), file-format specific flags, and a file format identifier. In this example, the indexed data is available as a result of the above-referenced transforming operations. In addition, other information from the received video file (e.g., width, height, audio and video frame rate, audio and video duration, file format, audio and video codec, etc.) may also be stored in the content index **130**. Media content is stored in source file(s) in a plurality of separately indexed portions. "Portions," as used herein refer to any segments or fragments of a file, and includes frames, chunks, groups, segments, or other portions defined herein.

The content index **130** is a file-format specific data structure that includes relevant information from the source media file headers and footers, as well as overall file information, such as described above. The content index **130** may be structured as a set of sorted lists, tables, trees, or other implementations. The content index **130** stores the media segmented into chunks, which are logical units representing indexed portions of files and identifying the location of the frames in the chunk. In other words, each media file is associated with some number of chunks, which can then be arbitrarily and independently accessed and manipulated. Similar to the file structure depicted in FIG. **5**A, each chunk has its own file name for internal data access, and includes a set of frames and/or groups. Each chunk is associated with a row or portion thereof in the content index **130** that represents a single media file. Each row has a maximum size of 64 k bytes, and is identified by a row identifier, which is a hash of the content identifier and an identifier for the video encoding format and resolution. In this example, the row identifier includes multiple columns to specify the content index **130** data as discussed herein. Alternatively, all data for all chunks of a media file is stored in a single database row, using one or more columns. For each chunk, a chunk index stores a chunk identifier number, chunk byte offset (from the start of the file associated with the chunk), chunk length in bytes, chunk start timestamp, chunk end timestamp, and for each stream in the chunk, a stream frame number, stream frame count, stream starting timestamp, and stream ending timestamp. This aspect of the present invention allows a given media file to be represented as a number of individual chunks, which can then be stored separately or together, so as to optimize read and write performance. Further separation of chunks into frames, groups, or other file portions allows the various file operations (transcoding, resolution, etc.) to be performed on the frame or group level, allowing the system to make changes at this level of granularity.

The content index **130** maps from content identifier and time to file name, byte offset, and (video, audio) frame number and/or group number. The mapping is created when media content is received by the server; the mapping is stored in the content index **130** in association with the row for a given media file. The mapping maps a specific time index point in both the video and audio feeds to a specific byte in the source file. Byte offsets are relative to the start of an audio or video frame according to one embodiment, and the content index **130** itself is stored in a format that is file-agnostic with the specific file format and codec stored as metadata in the index **130**. Thus, the content index **130** provides an indication of the relative location of a time index point in the original file, and exactly what content (e.g., audio and video) is stored at that byte.

Time to byte mappings exist for every video key frame in one embodiment. In other embodiments, time to byte map-

pings exist for every audio frame and video frame in the video article, for every N frames in the video article, or for every N video key frames, without limitation as to size (number of items in the mapping) of the information. The mapping from time to offset preferably maps to the nearest group. Most compressed video formats have the notion of a group of pictures ("GOP," "group"), which can be thought of an atomic set of video frames. This set of video frames cannot be cut at any point without decoding and re-encoding the video. Therefore, the media content may take advantage of this design, and be addressed at the group level rather than the frame level. Typically, audio formats do not have groups, and can be cut on individual audio frame boundaries. Each frame or group is sorted by time, and includes a start time, audio and/or video frame number, and byte offset. In other words, for any given time point (or range of time) in a media file, the content index **130** can be used to determine the specific chunk(s) that are associated with that time, and the specific frames thereof, and thereby retrieve them as needed.

The key data storage **135** may be a relational database or any other type of database that stores the data corresponding to keys for encrypting data within a URL. The key data storage **135** may be accessible by the server software **120** through a user interface.

The server software **120**, the source file data storage **125**, content index **130**, and key data storage **135** may be stored and operated on a single computer or on separate computer systems communicating with each other through a network.

The server **110** can store the media content either locally, or on remotely accessible systems, or a combination of both.

One skilled in the art will recognize that the system architecture illustrated in FIGS. **1** and **2** is merely exemplary, and that the invention may be practiced and implemented using many other architectures and environments.

As mentioned above, the present invention operates in the context of an information retrieval system for processing queries for video content. FIG. **3** is a flowchart illustrating one method for serving media content. The method begins when a request from a client **105** is received **310** for media content stored on a server **110**. The request is received at a network interface to the server **110**. The request to the server **110** includes the content identifier for the content that the user desires to play back, the resolution at which to play the media content, and a time range, specified as the starting time (in milliseconds, or another time unit, into the content) and length (in milliseconds or other units). The content identifier is an arbitrary binary identifier that uniquely identifies a single piece of stored media content and data about the media content, e.g., a 64-bit hash value computed from attributes of the stored media content, any unique string of characters, or a unique number assigned to this content at the time it is created. The content identifier is used to anonymously reference media content instead of using file names, e.g., to provide greater privacy and security. This aspect of the present invention is advantageous because it allows for time-based requests from clients, rather than traditional byte-based requests.

The request also includes an edit list. An edit list, as used herein, refers to a list of dynamic or static instructions for serving and editing stored media content. Edit list instructions can be added either by the client **105**, the server **110**, or both. For example, given an edit list received from the client **105**, the server **110** can pre-pend, append, or insert additional instructions into the client-requested content, or can modify the length, rate, number of streams, file wrapper format of the content. The edit list instructions can range from the very simple serving of an entire source file, to more complicated modifications, including serving a portion of a source file, serving more than one portion of one or more source files, changing the rate at which a source file or portion is served (speeding up or slowing down), adapting the rate at which a source file or portion is served to the client capabilities, encrypting a file or portion, adding or removing streams from the file or portion, interleaving streams from different files or portions, adding a data stream to a file or portion, transforming a file wrapper format, transforming from a proprietary file format to a standards-compliant format, inserting metadata into the file or portion, and various other modifications according to one embodiment. Other possible modifications include changing stereo audio into monophonic audio, changing color video into black and white video, reducing the resolution of a video stream, reducing the resolution of a still image stream, performing various visual algorithms on video or still images streams (sharpening, blurring, distorting, crossfade, etc.), and performing various audio algorithms (changing sample rate, pitch shifting, rate adjusting, echo, flange, etc.) according to various embodiments. Examples of ways to accomplish these modifications are included in the following description. The edit list that is provided in the request also may include an identifier which indicates that further instructions should be generated and included. These extra instructions may be generated by the server, or received from any machine connected to the server. Thus, the complete edit list need not be specified in the request, but can be generated internally at the time the request is received. This process allows every user to be provided with a unique media stream in response to the unique request.

The details of the request can be generated automatically or programmatically. The user is not required to specifically input the request details.

A request may be for content from a single media source file, an altered (e.g., truncated) version of the media content, or a combination of pieces from different media source files. For a request that includes more than one portion of one or more files, the request includes more than one time range, and the server **110** queries the content index to find the chunks that lie nearest to the requested timestamps according to one embodiment.

The request is a hyper text transfer protocol (HTTP) request (or similar transport protocol), and takes the form of a uniform resource locator (URL) **400**, as shown in FIG. **4**A. The URL **400** is in the typical format, including a protocol **405**, a host (and port, if applicable) **410**, and a resource path and/or query string **415**. In addition, an edit list **420**, in the form of instructions for performing operations on one or more files, is included. FIG. **4**A shows a simple example, in which a time range is the only parameter. In some embodiments, part of the URL **400** is encrypted as descried herein. FIG. **4**B shows an example of an encrypted URL **425**. In embodiments in which the URL is encrypted, the client cannot modify the encrypted portion, but can add modifications on to the URL, for example, to speed up the rate or to request a smaller subset of the media content. FIG. **4**C shows an example of an encrypted URL with additional edit list instructions **430** appended. In this example, an instruction has been added to speed up the rate for the content (to 1.5 times normal speed) and to select the time range 10 to 5000. The modification permitted by the client **105** does not alter the encrypted portion according to one embodiment. For example, if the file size is 1000, the request for 10 to 5000 may automatically be adjusted by the server to the available time range.

Once a request is received **310**, at least one source media file corresponding to the media content requested is opened **320**. To do this, the request is passed to a read thread, which does asynchronous lookups to the content index **130** using the

content identifier, computes starting byte offsets and frame lengths from the specified time range, and then opens **320** the source media file. In one embodiment, if more than one portion will be combined or more than one stream interleaved, the server **110** sorts the content items in the order in which they will be transmitted prior to reading the content index for all content items associated with all portions. In addition, the server **110** stores the relevant header and footer information, such as video resolution and frame rate, byte range of the portion of the file that is being composed, and other file format specific information. If a data stream is being added, the server **110** also reads content index information for a data stream, which includes a list of time-stamped data objects to be inserted in the stream. The time-stamped data objects can be arbitrary binary data, text data, or any other format.

In addition, a new file header is created from the information stored in the content index **130** according to one embodiment, which is written to the output. In one embodiment, the server **110** looks at the content index **130**, reads the header information, and constructs a segment media header representing the requested time range. The server **110** computes the number of audio and video frames that will be transmitted during this request. If more than one portion is being used, the server **110** computes the total number as the sum of frames from all content items that will be transmitted. If the content is being sped up or slowed down, the number of frames that will be transmitted will be different from the number of frames in the source media file(s). The server **110** also appends data specifying the original byte range and (video and audio) frame numbers according to one embodiment. If the data is being encrypted, the header may contain encryption keys, user identifying marks, or other information that allows the client to decrypt the encrypted data that follows. If the file wrapper is being transformed, the server **110** composes a valid file header, according to the "new" wrapper format (e.g., the format into which the wrapper is being transformed). If the wrapper transformation is from a proprietary format to a standards-compliant format, e.g., the original wrapper format is a format that is not standards compliant, and the new wrapper format is a standards compliant one.

Once the file is opened **320**, some or all of the file contents are read.

To reconstruct a media file in the same file format as the original source, the byte offsets of the first and last video and audio frames (or groups) of interest, as well as the number of video and audio frames in the segment, as well as any other metadata that may be useful to clients, are located. In addition, a new header (and footer, if applicable for the format) is created and spliced together to construct the output file.

A separately indexed portion of the media content is then selected **330** for transmitting based upon an instruction in the edit list. In one embodiment, the selecting **330** is part of a loop of reading from the source media file(s), and writing that data to the output. Once selected **330**, the server **110** finds the byte range in the original media source file and reads that region. In one embodiment, the output is a network transmit buffer. The network transmit buffer stores data ready for output to the network, for serving to the client according to one embodiment.

Optionally, the server **110** next modifies **340** the portion. Various modifications are possible; each modification is listed as an instruction in the edit list. Each modification step may or may not be length preserving, and thus the frames and other portions may be of any length after each step of the modification process. One possible data modification is data encryption. The contents of each frame or portion are encrypted with a per-request generated key. To accomplish this, the server **110** looks at each frame of this content, applies a standard encryption algorithm to some or all of these frames, and modifies the frames in a way that clearly delineates them as being encrypted before writing the encrypted data to the output. The keys are stored, e.g., in key data storage **135**, such that they can be retrieved later for decryption purposes. The use of encryption allows for insertion of additional data, e.g., ad targeting information. Another possible data modification is a rate change to a rate different from the original encoding rate. For example, a user can request a "fast forward" stream or portion that sends a file that will play back faster than the original content. To accomplish fast forward, in one embodiment frames are selected to be "dropped" from the source file, while the remaining frames are written to the output as normal, or with their timestamps adjusted accordingly. For example, for each video frame, the frame is either discarded or written to the output. The requested speedup factor and the specifics of the video encoding determine how many frames need to be discarded, and which frames may be discarded. The server **110** also needs to construct a valid audio stream that matches the rate of the video stream. The audio stream is constructed, e.g., by using a precomputed "silence" audio track in place of the original audio, doing more sophisticated audio processing to adjust audio rate to match the new video rate, or discarding a fraction of the original audio frames.

Another example is a "slow motion" stream or portion that will send data to be played back at a lower rate than the original source. To accomplish slow motion, frames are selected for duplication, or "padding" frames are added to automatically duplicate frames. For example, for each video frame, the frame is either transmitted intact, or a format-specific marker is placed in the file to duplicate the previously transmitted frame. The requested slow down factor and the specifics of the video encoding format determine how many frames need to be duplicated, and which frames may be duplicated, as well as the bytes that should be sent to mark duplicate frames. As with speeding up the content, the server **110** also needs to construct a valid audio stream that matches the rate of the video stream, e.g., as described above. Another possible rate change modification is an adaptive bit rate. To accomplish this modification, the server **110** determines whether the client **105** is able to receive the video at a rate sufficient to be decoded in real time. For example, the rate at which the client reads from the output is monitored throughout the request. If the rate at which the client is reading is less than the encoded rate of the video, the server **110** may replace the original encoded video frame with a significantly smaller, dynamically created frame that tells the video decoder in the client to duplicate the previous frame onscreen. In another embodiment, the server **110** applies an algorithm to modify the original encoded video frame to create a new frame of a smaller size. This algorithm may include a full or partial decode of the original video frame, as well as a full or partial re-encode of the video frame. This process reduces the overall transmitted audio and video rate.

Another possible data modification is selection of data streams or other file portions from within a file. For a data file that includes multiple streams of content, the server **110** can dynamically choose not to send some of the streams. For example, a data file may contain one video stream and three audio streams. At the time the file is sent, one of the three audio streams is chosen for sending with the video stream. For example, the server **110** reads correct byte range from the first file, and writes this to the output, and then repeats this process for subsequent files until each content item has been read and written to the output. Yet another possible data

modification is stream interleaving or data interleaving. The server **110** interleaves data from one or more files together to create a new, dynamic piece of content. For example, the server could choose to overlay an audio stream from one file with video stream from another file. In this example, the server **110** reads chunks simultaneously from each file, then takes each video and audio stream pair, and applies a file format specific stream marker to each matched pair of audio and video. It then sorts all of the frames by their representative timestamps, and writes this data to the output. If the number of audio and video frames is different between the streams, then the server **110** may pad the video with empty video frames, black video frames, or some other file format specific construct. A similar padding operation may be used for audio streams. This process ensures that the total duration of video and audio content is identical in each stream. Another example of data interleaving is mixing a text or metadata stream, e.g., closed captioning or promotional data, with an existing audio and/or video stream. To add a data stream, the index information for the data stream, e.g., the byte range, is gathered and transformed into a format suitable for inclusion in the resultant media file. The video frames and data stream frames are then sorted by their internal timestamps, and each frame is subsequently written to the output. This process results in an effective interleaving of the data and video streams, and the data stream will be synchronized correctly with the video and audio streams. In one embodiment, the added data is metadata dynamically computed by the server. The metadata may be data that is not useful to the user, and is consumed by the playback client.

Another possible data modification is data wrapper transformation. Different computers and operating systems support different file wrapper and frame wrapper formats. For each source file, the frame header and footer specific to the original format are discarded, and generate new headers and footers that are specific to a desired file format. For each frame contained in this data, the server **110** parses the data according to the original wrapper format, extracts the compressed video or audio data, and then encodes this data using the "new" wrapper format, and writes this byte range to the output. Yet another possible data modification is calculated data insertion, wherein data is generated on the fly and inserted into the served file according to one embodiment. For example, a new "stream" is created and specified in the file header, and this stream is transmitted by the server. In this example, the new stream includes data transmission statistics, popularity statistics, or other computed metadata according to various embodiments.

In addition, for each frame, various types of audio and/or video processing may be applied, e.g., using one or more processing algorithms. The algorithm(s) may operate directly on the encoded video to create the desired video effect. Or the server **110** may do a partial or full decode of each video frame, apply an arbitrary video transformation on the partially or fully decoded video and then do a partial or full encode to produce the resultant frame. Each audio frame may be processed in a similar manner. Video processing and enhancement algorithms are well known, and include such things as color to black and white transformation, sharpening, blurring, arbitrary distortions, resolution reduction, and blending effects between one or more video streams. Audio processing steps include such things as sample rate changes, pitch shifting, bitrate adjustments, echo, flange, crossfade, converting to monaural, and dynamic range compression.

In some embodiments, a per-frame transformation is applied to make the file appear consistent. The timestamp of the output video article is updated so that the first frame is timestamp **0** and increases over time according to some embodiments. Although the time durations of video and audio frames are often unequal, because separate video and audio time offsets are stored in the content index **130**, there is as little offset as possible between video and audio frames. Thus, no re-muxing of video and audio frames is required (but is possible, if desired). According to one embodiment, the server **110** also has the ability to dynamically modify data by chaining together multiple modifications for the same piece media content. These aspects allow insertions, deletions, and other modifications to be made at the time of serving the media file portion.

Once any modifications **340** are made to the portion, the portion is written **350** to the output. Then, optionally it is determined **360** whether more instructions apply for the requested media. If the edit list includes additional instructions, steps **320-350** repeat according to the next instruction in the edit list.

In one embodiment, the server **110** next constructs a footer that includes an index of relevant information from the original media source file and writes the footer to the output. In one embodiment, a region of the source file footer is read, modified, and placed in the network transmit buffer. If greater than one file portion is to be included in the served content, a region of the first source media file footer is read first, then subsequent files in series. If the data wrapper is being transformed, the server **110** parses the footer according to the original wrapper format, and modifies the footer to be consistent with the new wrapper format. As with the header, if the transformation is from a proprietary format to a standards-compliant format, the original wrapper format is a format that is not standards compliant, and the new wrapper format is a standards-compliant one.

Up to this point, the steps at the server **110** have been performed by a single read thread according to one embodiment. A separate thread, a write thread, controls the writing of data from the output (network transmit buffer) to the network **115** for serving to the client. The write thread determines whether there is data in the network transfer buffer. If there is, the data is written to the network. If there is no data in the network transfer buffer, or the buffer is nearly empty, the write thread asks for more data, i.e., from the above process by the read thread.

FIG. **6** shows an example of a use case for the method described herein according to one embodiment. In this example, streams from different file portions are interleaved. As a preliminary step, a user uploads one or more pieces of media content ("shots") to a server, e.g., via a user interface. In this example, the user uploads media content including shots **1-4**. The media content is transcoded on the server into the serving format. Then, using a user interface, the user selects, edits, and arranges the content into the format desired for viewing. In this example, the user wants to create a new piece of media content that includes a portion of the video from the content associated with shots **1-3**, i.e., the portion shown as shots **1-3**, with a portion of the audio associated with shot **4**, i.e., the portion shown as shot **4**. The user interface software converts the instructions into an edit list as part of a request from the client for the edited content, which is sent to the server. After the steps detailed below, the client will receive from the server a single piece of content edited per the request.

The server receives the request from the client, in the form of a URL as described in conjunction with FIGS. **4**A-C, which includes the content identifier for each piece of content that the user desires to play back, in this example shots **1-4**, and the resolution at which to play the content. The request

also includes a time range for each shot, specified as the starting time (e.g., in seconds into the content) and length (e.g., in seconds). For example, for shot **1**, the starting time is 10 seconds (begin=10) and the length is 8 seconds (len=8). The request also includes an edit list that refers to a list of instructions for interleaving the four streams from shots **1-4**, such that the new piece of content includes the video of each of shots **1-3** and the audio of shot **4**.

As discussed in conjunction with FIG. **3**, the source media file corresponding to the media content requested, in this example, corresponding to each of shots **1-4**, is opened. To accomplish this step, the request is passed to a read thread, which does asynchronous lookups to the content index using the content identifiers, computes starting byte offsets and frame lengths from the specified time ranges, and opening each file. The read thread also creates a new header for the new content.

Following the instructions in the edit list sequentially, a next frame of the media content is selected for transmitting. In this example, the server interleaves data from four files together to create a new, dynamic piece of content. Specifically, the server selects an audio stream from one file (shot **4**) to interleave with the video streams from three other files (shots **1-3**).

In this example, the audio frame of shot **4** with timestamp "41" is selected **605** for transmitting and is written to the network transfer buffer, as illustrated in FIG. **6**. Next, the video frame of shot **1** with timestamp "11" is selected **610** and written to the output buffer. The read thread continues to select the next frame in sequence until completing the last instruction in the edit list. The frames are renumbered such that the timestamps associated with the newly created piece of content increase monotonically from the new starting time. The frames of the new shot may include any of the various modifications described herein as compared with the frames from the original shots, as indicated by the fact that the audio and video frames respectively are designated A' and V' rather than A and V. The read buffer then constructs the new footer. The new header, content, and footer are spliced together to construct the new output file for serving to the client. When the write thread detects that there is data in the buffer, the data is transmitted to the network for serving to the client.

FIG. **8** is an interaction diagram showing exchange of information between a client **105** and a server **110**.

The process begins when the server **110** receives a request **805** for a media file from client **105**. The server stores each media file as a plurality of chunks (sometimes referred to as or "portions" or "segments" herein), each corresponding to a predetermined time interval of the media file. Each chunk is separately indexed and is accessible and capable of being separately served to the client. The form of the request includes a content identifier and is similar to the requests described herein, for example in conjunction with the method of FIG. **3**.

The client **105** request is for download of the first segment (corresponding to the first time interval) of the media file from the server **110**. The request may comprise a time range, specified as the beginning millisecond and length in milliseconds for the segment. The details of the request can be generated automatically or programmatically. The user is not required to specifically input the request details. He user need only select the media file for play.

The media file is stored on the server **110** in segments corresponding to time intervals of fixed length as described herein. The segments may also (or alternatively) be stored on the server **110** in varying-length segments that begin and end with a key frame. In the context of audio content, for most audio formats there is no such thing as a "key frame" and thus no correlation between the audio frame number and the video frame number. A typical file might have several audio frames for every video frame. In one embodiment, groups of audio frames may be indexed in sets approximating a selected time interval, e.g., 250 ms. In another embodiment, every audio frame is indexed. In addition, audio frames may have a variable size (in both bytes and time duration), but because there are no key frames, any indexing interval could be used, e.g., a constant number of index points (say, 1000 per file) or a constant number of audio frames, etc.

Alternatively, the server **110** stores the file in a continuous or "chunked" format, with the chunks aligned on video key frame boundaries, or not so aligned. A container file format is used to encapsulate the underlying video or audio data, e.g., as described in conjunction with FIG. **5**A.

Using the content identifier, the server **110** opens the file corresponding to the requested content and selects **810** a first media file portion for serving to the client. Initially, this is the first segment, corresponding to the first time interval.

As part of this process, the server **110** may look at the index for the media file, read the header information, and construct a segment header representing the requested time range. The server **110** also may append data specifying the original byte range and (video and audio) frame numbers. The server **110** then finds the byte range in the original file, and reads that region. In some embodiments, a per-frame transformation is applied to make the file appear consistent. The server **110** may also apply encryption to the data. The server **110** may construct a footer that includes an index of relevant information from the original file. The server **110** also may append or prepend a portion of non-requested content, for example an advertisement. If prepended, the non-requested content may be transmitted to the client prior to transmitting the requested content.

The following is an example of this process using a standard AVI file, e.g., as described in conjunction with FIG. **5**B. When a request is received by the server for video starting at time 250 ms and ending at 333 ms, the server examines the index to find the frame before 250 ms (video frame **2**) and the frame that goes beyond 333 ms (video frame **3**, as it is played for time 300-399 ms). The server computes that it will send two frames total, starting at frame **2**, covering byte range **4234** (start of frame **2**) through byte **4788** (end of frame **3** and surrounding audio).

The server constructs a segment header for this information, e.g.:

{width=320, height=240, video_format=mpeg-4, audio_format=mp3, video_frames=2, audio_frames=3, etc.}

The server also sends bytes **4234** through **4788** from the original media file (i.e., the segment) in response to the request, and reads the footer from the media file (byte **5499** and beyond) to extract the information for the relevant frames, e.g.:

{{video frame, offset=4234}
{video frame, offset=4567}},

adapting the "offset=" values (using simple subtraction) such that they match the offsets in the segment.

Various modifications and transformations of the media file portion, such as described herein, may also be made before serving the content to the client **105**.

Next, the server serves **815** the first portion of the media file to client **105**. As part of this process, the server **110** responds to the download request from the client **105** with a time range response corresponding to the requested segment, and the segment header (and/or footer) information.

The client **105** receives the segment and preallocates **820** hard drive space for the aggregate file. The preallocation may be the result of an instruction and overall file size information received from the server **110**, either as part of the initial time range response, or previously received in conjunction with other information about the media file, such for displaying to the user, e.g., as part of a list of media files resulting from a query. Alternatively, the entire file may be stored in memory until completed, and then written to disk.

Next, the client writes **825** the first segment or portion to the file and maintains a list of intervals not yet downloaded. The client uses the above-described information to preallocate space on a local storage device (e.g., an internal hard drive) for the final, aggregate file corresponding to the complete media file. The client **105** may merge each segment into the final, aggregate file as each is received. In this case, as the client downloads each audio and/or video segment, it writes the segments to the aggregate file and updates the download index. Even if the download is cancelled (e.g., when the user jumps to another location), the client retains any segment that has already been downloaded. Alternatively, the segments are merged when the entire media file is downloaded, as described in conjunction with step **890** below.

The time range response includes information indicating the location within the media file where the segment occurs. As the media file is downloaded, the audio and/or video segments are written to a file by the client **105**. The client maintains a download index that lists which segments have been downloaded and which have not yet been downloaded. The index contains an entry for each video and audio segment, and each entry contains a bit that specifies whether the segment has been downloaded or not. The download index lists a segment as not downloaded if either audio frames or video frames for the segment have not been downloaded, e.g., such that if download resumes the index indicates that the segment needs to be downloaded. Because the audio stream and video stream may be interlaced in an uneven way, it is possible that only one of the two streams (either audio or video) has been downloaded.

Once the first time range response corresponding to a segment is received, the client **105** plays media file from the beginning **830**. When the download of one segment completes, the client continues downloading **835** segments in sequence with the next closest time segment after the current position, so that the user may continue viewing the media file.

When the user wants to view a different location in the media file, the user so indicates using a widget or other control in the user interface. For example, the user may move the cursor to a location on a time bar in the user interface representing a different segment in the media file. As a result, the client **105** chooses the closest segment beginning at or before the location chosen by the user. The server **110** receives **840** the request to play a different part of the media file from the client.

For each segment requested, the first video frame in the file must be a key frame. Key frames are segments of the media file from which play can begin. In one embodiment, key frames are intra-coded frames, i.e., frames that do not depend on previous frames for coding. Key frames may be labeled as such in the original media file as received by the server, and/or markers indicating that the frame is a key frame may be added by to frames by the server before sending a segment to the client. As a result of sending from the nearest key frame, the server **110** may return a segment of the media file that is larger than the requested segment. However, going to the closest preceding key frame ensures that the segment returned by the server includes the entire requested interval. The location within the media file (particular time interval beginning with the key frame) from which the download proceeds is shown on the user interface, but the difference may be invisible to the user because of the short duration between the beginning of the key frame and the location the user requested. In one embodiment, the media files are sent with zero preload encoding, i.e., the audio and video frames are sent such that there is minimal difference between the timestamps of adjacent audio and video frames.

Thus, the server **110** again opens the file and chooses **845** the next segment and sends **850** it to the client.

The client **105** again writes **855** the portion to the file, and updates the list of intervals. The client **105** then can play **860** the media file from new portion. The download then continues **835** from this segment. The steps in box **865** repeat every time the user selects a segment of the media file that has not yet been downloaded. As a result, the user may start and stop the stream download as desired, and may start a new stream.

However, if the user selects a segment of the media file that already has been downloaded, the user can view the media file from that point and the download continues from the end of the last downloaded segment of the media file following the selected segment.

When the server **110** receives **870** a request for an already downloaded portion of the media file and/or once the client **105** has downloaded everything between the current position and the end of the media file, the client **105** then downloads the first segment in any area of the media file that has not been downloaded yet, starting from the beginning of the media file. Alternatively, the client **105** allows the user to choose to download only selected segments of the media file or only the entire media file. Because the portion requested already was downloaded, the client **105** begins play **875** from the beginning of the selected portion already downloaded without the need to receive a new segment from the server **110**.

To continue the download process, the server **110** opens the file and selects **880** the first portion not downloaded, either following the selected portion, or the next lowest time stamped segment, starting from the beginning of the media file. The server **110** then sends **885** that next portion. Again, download continues **835** from that point onward.

In the case in which the client **105** not write each segment to the preallocated storage space as each is received, once all segments are downloaded, the client **105** then combines **890** the portions and saves the media file on the client hard drive. The video and audio streams may need to be matched up if the streams are separate.

If the download is cancelled (e.g., when the user selects another video for download), the client **105** may retain the segments of the media file that have already been downloaded. E.g., the client **105** may continue to download the remaining segments of the media file in the background. Alternatively, the client **105** may halt the download unless the user selects another segment of the media file for download. In yet another embodiment, the behavior of the client **105** in response to a cancelled download may be determined by a user's default preferences or input selections.

If the media player for any reason closes during the download, the disconnect is treated as if the connection between the client **105** and server **110** has terminated. In this scenario, the client **105** saves the segments already downloaded, freezing the state of the background download. If the download needs to be restarted, the client **105** uses the index to determine which segments it needs to download and downloads only those segments.

FIG. **9** is an illustration of a media player user interface **905** according to the methods of the present invention. In an

example of the above process, the user begins by selecting the media file for play. As a result, the client begins playing the media file from the first segment, e.g., at location **915** on timeline bar **910**. Download proceeds from this point, downloading segments that follow the first segment, as shown by the darkened bar **920**. Thus, the user can continue to view the media file from the beginning.

When the user wants to view a different location in the media file, the user moves the cursor to a location on the user interface, e.g., any of locations **925** on the timeline bar **910**, and download of that segment begins. Then, the client continues downloading segments in sequence as described above, beginning with the next closest segment after the current position, as indicated by the filled-in bar following locations **925**. Thus, the user can continue viewing the media file at each point to which the user jumps, without the need to wait or the entire download to catch up to the selected location in the media file.

The present invention has been described in particular detail with respect to one possible embodiment. Those of skill in the art will appreciate that the invention may be practiced in other embodiments. First, the particular naming of the components, capitalization of terms, the attributes, data structures, or any other programming or structural aspect is not mandatory or significant, and the mechanisms that implement the invention or its features may have different names, formats, or protocols. Further, the system may be implemented via a combination of hardware and software, as described, or entirely in hardware elements. Also, the particular division of functionality between the various system components described herein is merely exemplary, and not mandatory; functions performed by a single system component may instead be performed by multiple components, and functions performed by multiple components may instead performed by a single component.

Some portions of above description present the features of the present invention in terms of algorithms and symbolic representations of operations on information. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. These operations, while described functionally or logically, are understood to be implemented by computer programs. Furthermore, it has also proven convenient at times, to refer to these arrangements of operations as modules or by functional names, without loss of generality.

Unless specifically stated otherwise as apparent from the above discussion, it is appreciated that throughout the description, discussions utilizing terms such as "determining" or "displaying" or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system memories or registers or other such information storage, transmission or display devices.

Certain aspects of the present invention include process steps and instructions described herein in the form of an algorithm. It should be noted that the process steps and instructions of the present invention could be embodied in software, firmware or hardware, and when embodied in software, could be downloaded to reside on and be operated from different platforms used by real time network operating systems.

The present invention also relates to an apparatus for performing the operations herein. This apparatus may be specially constructed for the required purposes, or it may comprise a general-purpose computer selectively activated or reconfigured by a computer program stored on a computer readable medium that can be accessed by the computer. Such a computer program may be stored in a computer readable storage medium, such as, but is not limited to, any type of disk including floppy disks, optical disks, CD-ROMs, magnetic-optical disks, read-only memories (ROMs), random access memories (RAMs), EPROMs, EEPROMs, magnetic or optical cards, application specific integrated circuits (ASICs), or any type of media suitable for storing electronic instructions, and each coupled to a computer system bus. Furthermore, the computers referred to in the specification may include a single processor or may be architectures employing multiple processor designs for increased computing capability.

The algorithms and operations presented herein are not inherently related to any particular computer or other apparatus. Various general-purpose systems may also be used with programs in accordance with the teachings herein, or it may prove convenient to construct more specialized apparatus to perform the required method steps. The required structure for a variety of these systems will be apparent to those of skill in the art, along with equivalent variations. In addition, the present invention is not described with reference to any particular programming language. It is appreciated that a variety of programming languages may be used to implement the teachings of the present invention as described herein, and any references to specific languages are provided for invention of enablement and best mode of the present invention.

The present invention is well suited to a wide variety of computer network systems over numerous topologies. Within this field, the configuration and management of large networks comprise storage devices and computers that are communicatively coupled to dissimilar computers and storage devices over a network, such as the Internet.

Finally, it should be noted that the language used in the specification has been principally selected for readability and instructional purposes, and may not have been selected to delineate or circumscribe the inventive subject matter. Accordingly, the disclosure of the present invention is intended to be illustrative, but not limiting, of the scope of the invention, which is set forth in the following claims.

What is claimed is:

1. A method of enabling discontinuous download of media files, the method comprising:

storing at a server a media file comprising a plurality of chunks of the media file corresponding to separate, contiguous time intervals of the media file, wherein the plurality of chunks comprise a frame selected from a group consisting of a video stream, an audio stream, a text stream, and a metadata stream, wherein the chunks are not stored as TCP/IP packets, and the server comprises a server computer system;

receiving at the server from a client a request for a first of the plurality of chunks of the media file corresponding to a first time interval of the media file, wherein the request comprises at least one content identifier and instructions corresponding to the at least one content identifier, the instructions further comprising a content transformation instruction;

transmitting from the server to the client a portion of non-requested content;

transmitting the first chunk from the server to the client to enable playback of the first chunk at the client;

tracking which of the plurality of chunks have been transmitted to the client;

in response to receiving a request from the client for one or more of the plurality of chunks that have not been transmitted to the client, the one or more chunks correspond-

ing to one or more time intervals of the media file not contiguous with the first time interval, transmitting the requested one or more chunks to the client to enable playback of the one or more chunks at the client;

continuing to transmit remaining of the plurality of chunks from the server to the client;

transmitting overall file size information for the media file from the server to the client for preallocation of storage space for the media file on the client; and

wherein header and footer information are transmitted from the server to the client with each transmitted chunk.

2. The method of claim **1**, wherein the non-requested content is transmitted prior to transmitting the first chunk from the server to the client.

3. The method of claim **1**, further comprising, in response to receiving a request from the client for one or more of the plurality of chunks that have been transmitted from the server to the client, transmitting from the server to the client a next of the one or more of the plurality of chunks that have not been transmitted.

4. The method of claim **1**, wherein each segment is written to the preallocated storage as it is received by the client.

5. The method of claim **1**, wherein the client and the server reside on a single device.

6. A computer program product for enabling discontinuous download of media files, comprising:

a non-transitory computer-readable storage medium; and

computer program code, coded on the medium, for:

storing at a server a media file comprising a plurality of chunks of the media file corresponding to separate, contiguous time intervals of the media file, wherein the plurality of chunks comprise a frame selected from a group consisting of a video stream, an audio stream, a text stream, and a metadata stream, wherein the chunks are not stored as TCP/IP packets, and the server comprises a server computer system;

receiving at the server from a client a request for a first of the plurality of chunks of the media file corresponding to a first time interval of the media file, wherein the request comprises at least one content identifier and instructions corresponding to the at least one content identifier, the instructions further comprising a content transformation instruction;

transmitting from the server to the client a portion of non-requested content;

transmitting the first chunk from the server to the client to enable playback of the first chunk at the client;

tracking which of the plurality of chunks have been transmitted to the client;

in response to receiving a request from the client for one or more of the plurality of chunks that have not been transmitted to the client, the one or more chunks corresponding to one or more time intervals of the media file not contiguous with the first time interval, transmitting the requested one or more chunks from the server to the client to enable playback of the one or more chunks at the client;

continuing to transmit remaining of the plurality of chunks from the server to the client;

transmitting overall file size information for the media file from the server to the client for preallocation of storage space for the media file on the client; and

wherein header and footer information are transmitted from the server to the client with each transmitted chunk.

7. The computer program product of claim **6**, wherein the client and the server reside on a single device.

8. The computer program product of claim **6**, wherein the non-requested content is transmitted prior to transmitting the first chunk from the server to the client.

9. The computer program product of claim **6**, further comprising computer program code, coded on the medium, for, in response to receiving a request from the client for one or more of the plurality of chunks that have been transmitted from the server to the client, transmitting from the server to the client a next of the one or more of the plurality of chunks that have not been transmitted.

10. The computer program product of claim **6**, wherein each segment is written to the preallocated storage as it is received by the client.

11. A system for enabling discontinuous download of media files, the system comprising:

means for storing at a server a media file comprising a plurality of chunks of the media file corresponding to separate, contiguous time intervals of the media file, wherein the plurality of chunks comprise a frame selected from a group consisting of a video stream, an audio stream, a text stream, and a metadata stream, wherein the chunks are not stored as TCP/IP packets, and the server comprises a server computer system;

means for receiving from a client a request for a first of the plurality of chunks of the media file corresponding to a first time interval of the media file, wherein the request comprises at least one content identifier and instructions corresponding to the at least one content identifier, the instructions further comprising a content transformation instruction;

means for transmitting from the server to the client a portion of non-requested content;

means for transmitting the first chunk from the server to the client to enable playback of the first chunk at the client;

means for tracking which of the plurality of chunks have been transmitted to the client;

means for transmitting the requested one or more chunks to the client to enable playback of the one or more chunks at the client in response to receiving a request at the server from the client for one or more of the plurality of chunks that have not been transmitted to the client, the one or more chunks corresponding to one or more time intervals of the media file not contiguous with the first time interval;

means for continuing to transmit remaining of the plurality of chunks from the server to the client;

means for transmitting overall file size information for the media file from the server to the client for preallocation of storage space for the media file on the client; and

wherein header and footer information are transmitted from the server to the client with each transmitted chunk.

12. The system of claim **11**, wherein the client and the server reside on a single device.

13. The system of claim **11**, wherein the non-requested content is transmitted prior to transmitting the first chunk from the server to the client.

14. The system of claim **11**, further comprising means for, in response to receiving a request from the client for one or more of the plurality of chunks that have been transmitted from the server to the client, transmitting from the server to the client a next of the one or more of the plurality of chunks that have not been transmitted.

15. The system of claim **11**, wherein each segment is written to the preallocated storage as it is received by the client.

* * * * *