



US 20090282162A1

(19) **United States**(12) **Patent Application Publication**
Mehrotra et al.(10) **Pub. No.: US 2009/0282162 A1**(43) **Pub. Date: Nov. 12, 2009**(54) **OPTIMIZED CLIENT SIDE RATE CONTROL
AND INDEXED FILE LAYOUT FOR
STREAMING MEDIA****Publication Classification**(51) **Int. Cl.**
G06F 15/173

(2006.01)

(52) **U.S. Cl. 709/233**(57) **ABSTRACT**(75) **Inventors:** **Sanjeev Mehrotra**, Kirkland, WA
(US); **Kishore Kotteri**, Bothell, WA
(US); **Bharath Siravara**, Issaquah,
WA (US); **Thomas W. Holcomb**,
Bothell, WA (US); **Hui Gao**,
Redmond, WA (US); **Serge**
Smirnov, Redmond, WA (US)

Correspondence Address:

KLARQUIST SPARKMAN LLP
121 S.W. SALMON STREET, SUITE 1600
PORTLAND, OR 97204 (US)(73) **Assignee:** **Microsoft Corporation**, Redmond,
WA (US)(21) **Appl. No.:** **12/119,364**(22) **Filed:** **May 12, 2008**

An indexed file layout, comprising index information, is defined for segmented streaming of multimedia content. The index information can comprise program description information and streaming segment index information. In addition, the layout can comprise files containing streaming segments of the program, where the streaming segments are each encoded at one or more bitrates independently of other streaming segments of the program. The layout supports client switching between different bitrates at segment boundaries. Optimized client-side rate control of streaming content can be provided by defining a plurality of states, selecting available paths based on constraint conditions, and selecting a best path through the states (e.g., based on a distortion measure). In one client-side rate control solution states correspond to a specific bitrate of a specific streaming segment, and in another client-side rate control solution states correspond to a measure of client buffer fullness.

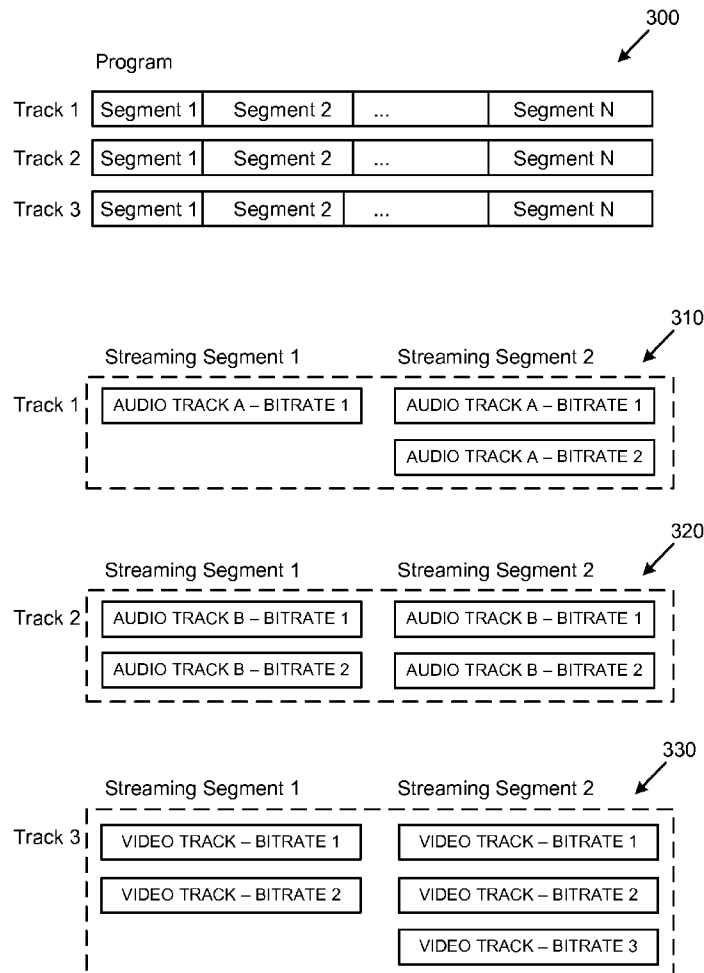


FIG. 1

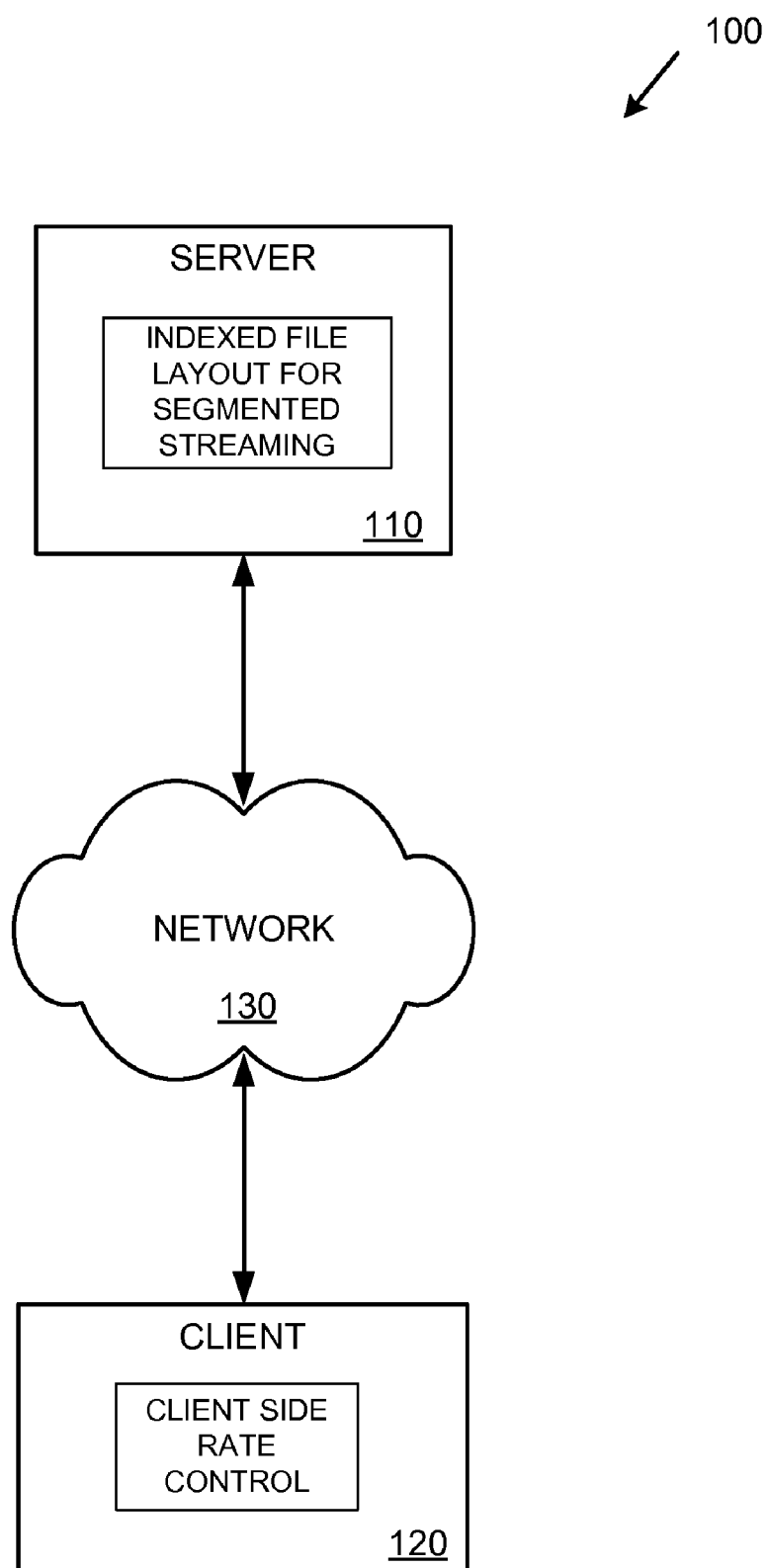


FIG. 2

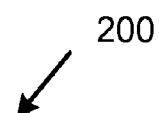
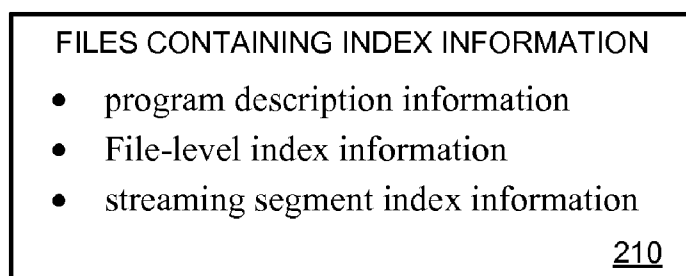
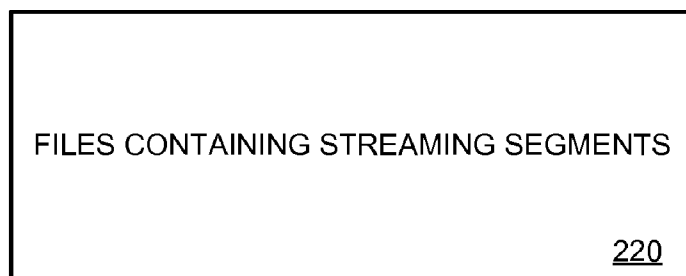


FIG. 3

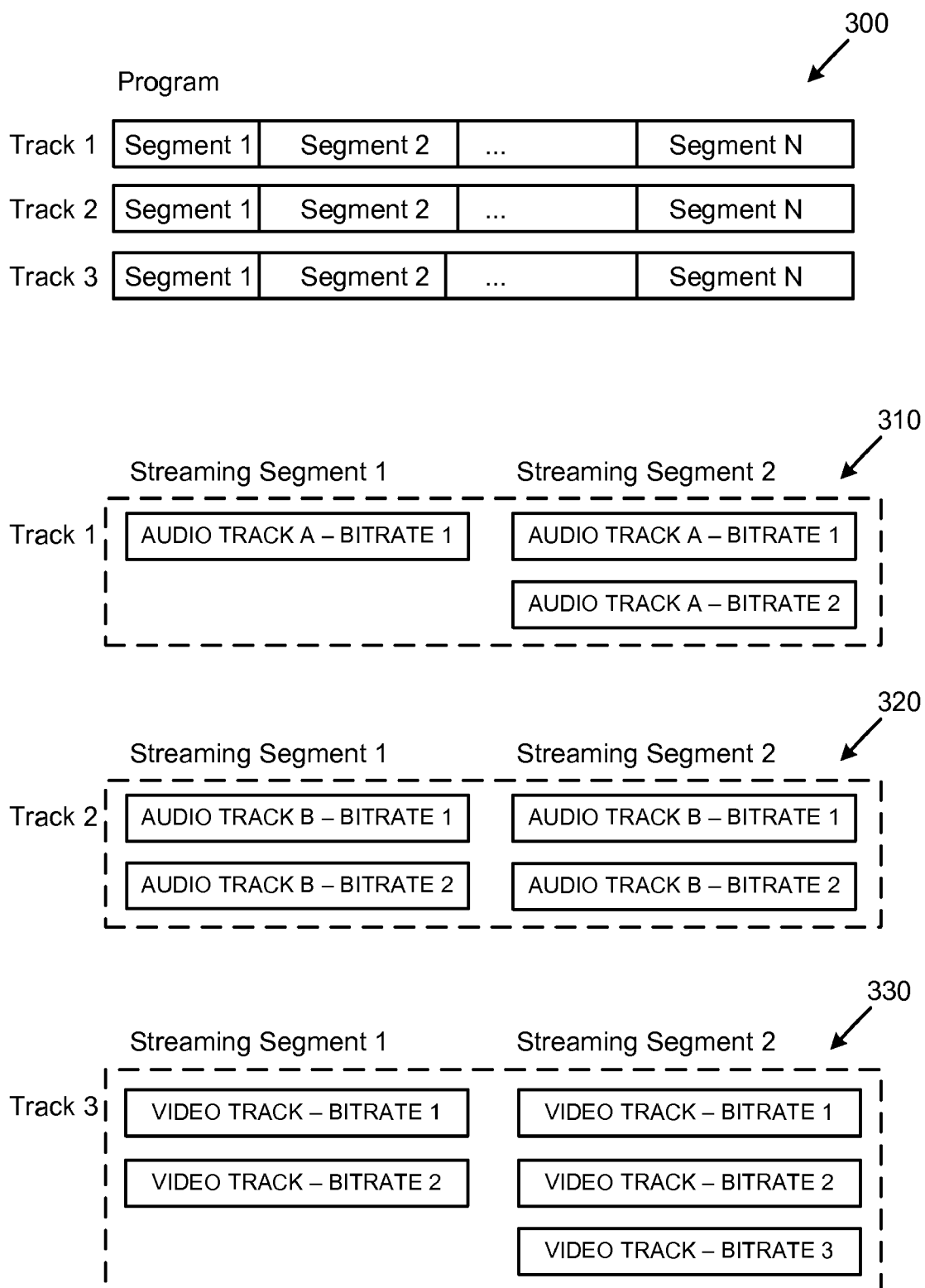


FIG. 4

400
↙

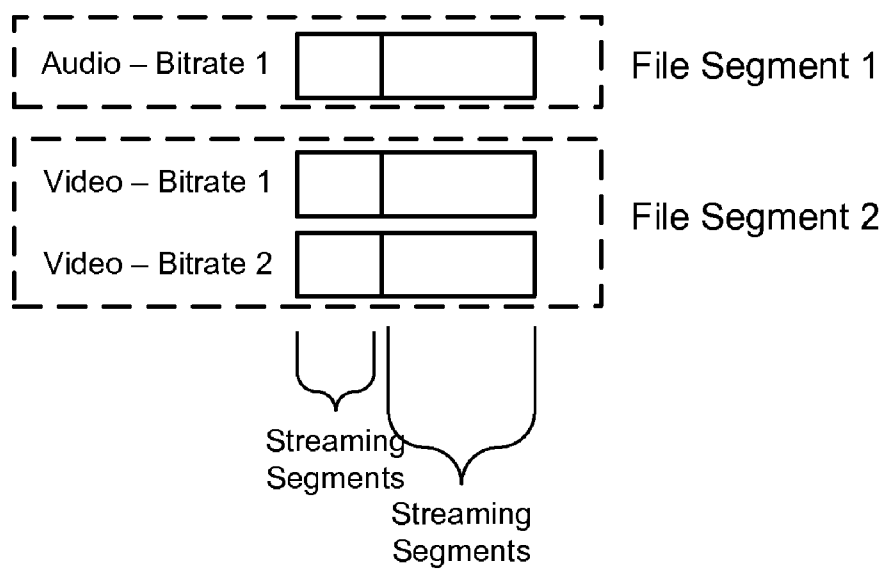


FIG. 5

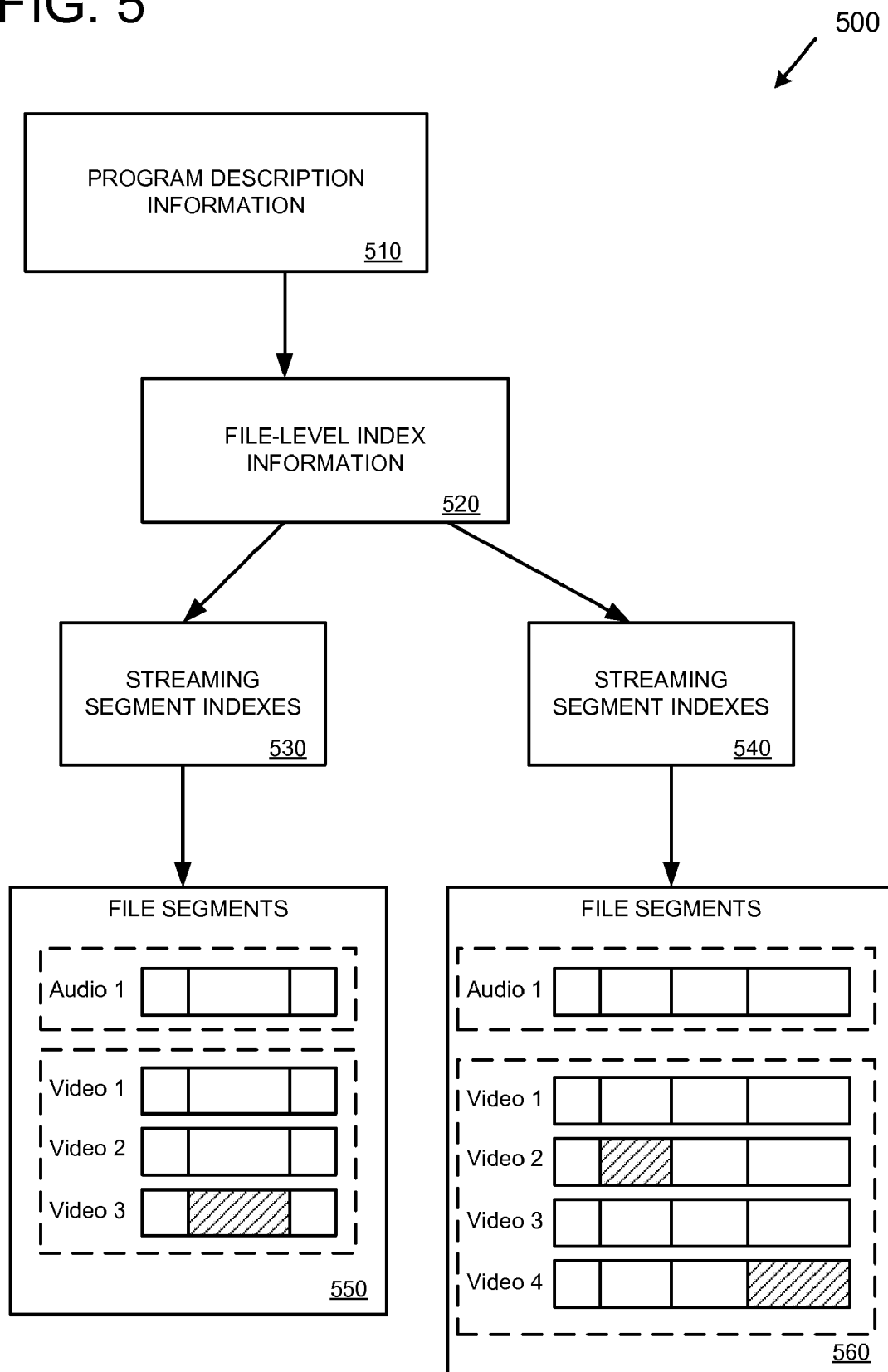


Figure 6

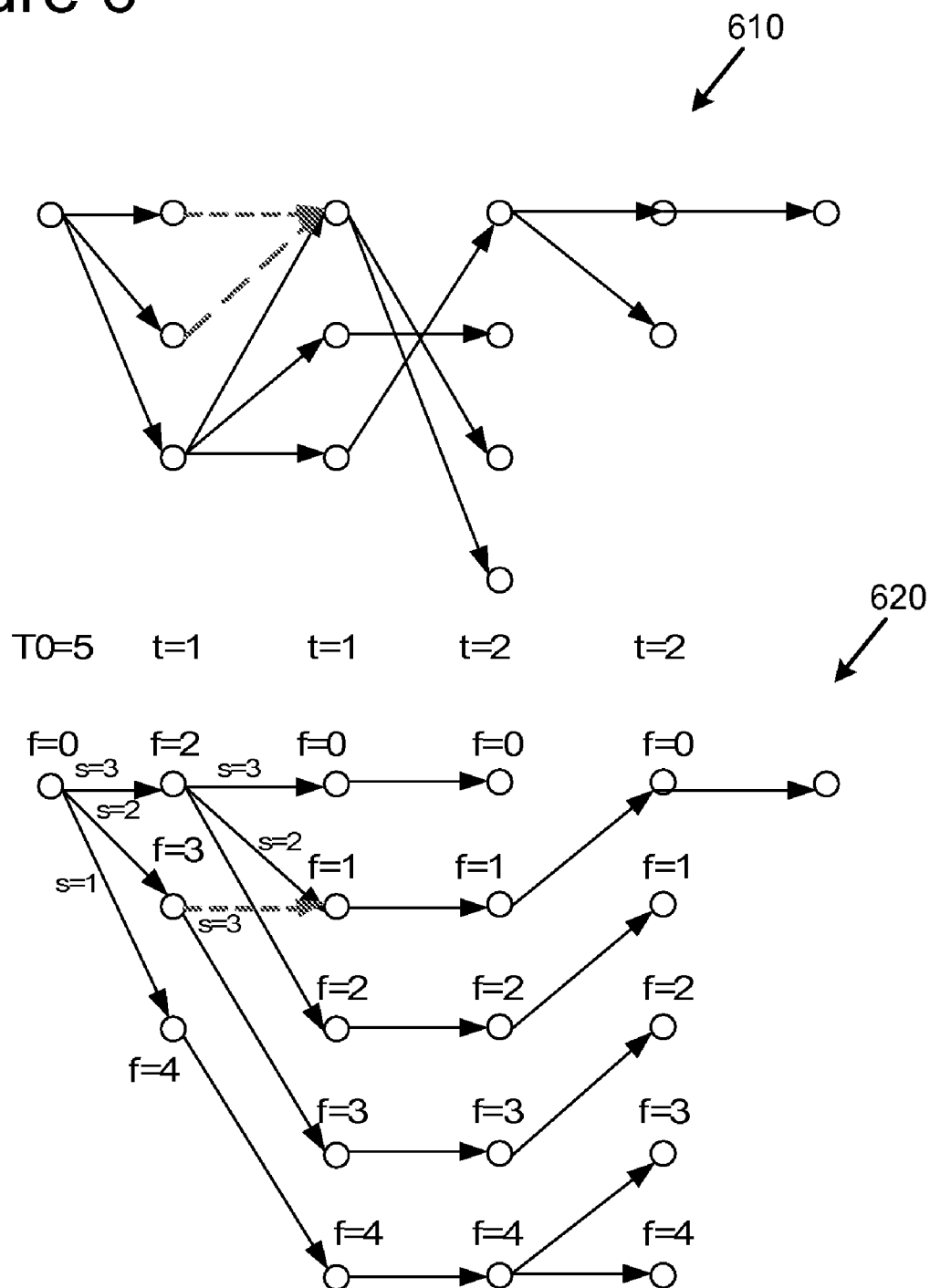
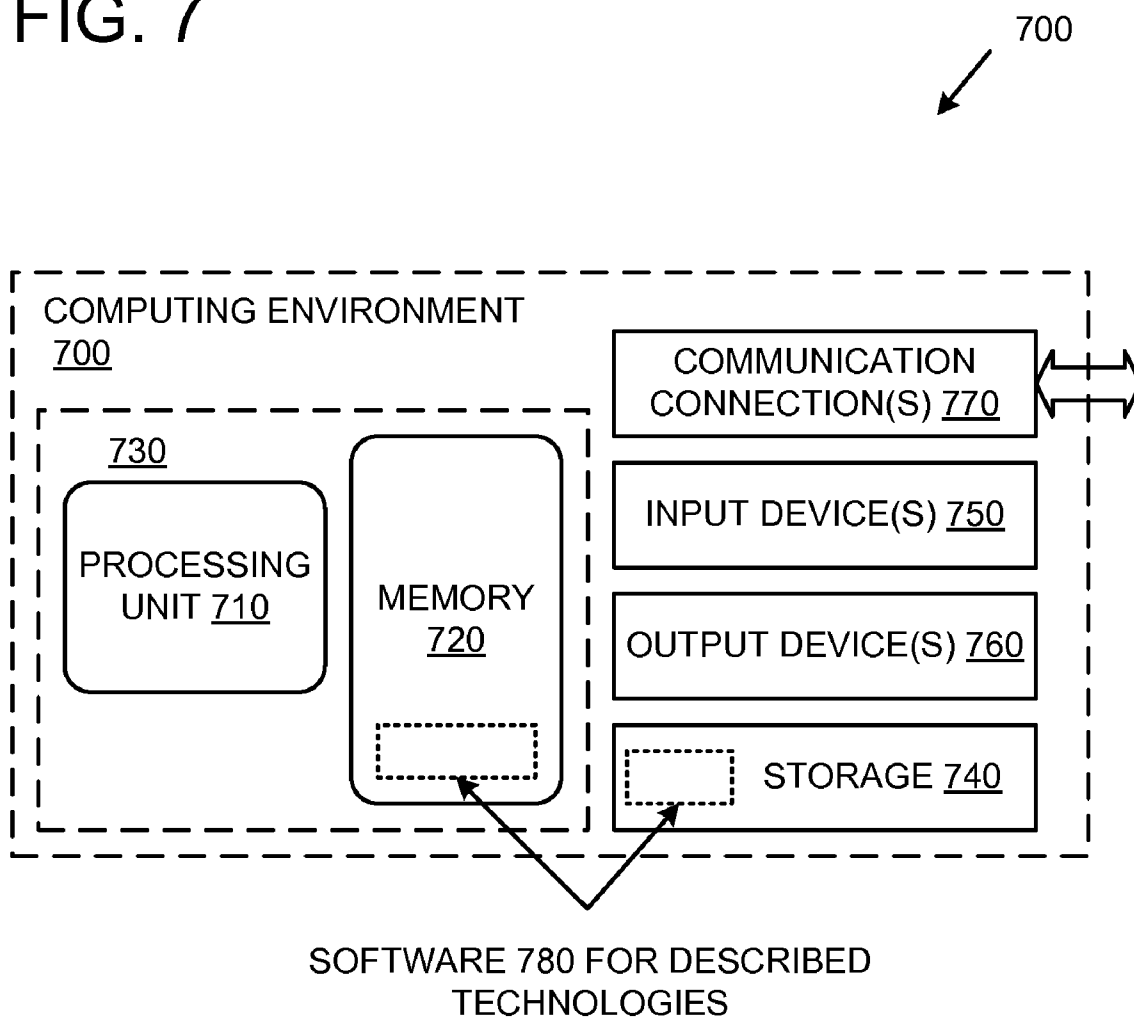


FIG. 7



OPTIMIZED CLIENT SIDE RATE CONTROL AND INDEXED FILE LAYOUT FOR STREAMING MEDIA

BACKGROUND

[0001] With the increasing popularity of viewing and/or listening to content, such as music, shows, and movies, over the Internet, there is a need for optimizing the client's viewing experience. Because the Internet is a variable bitrate channel without dedicated bandwidth between the server and client, it is important to adjust the bitrate of the content being sent to the client.

[0002] For bitrate-switching streaming solutions that rely on server-side awareness of the needs of each specific client, additional server-side resources may be required. For example, locating bitrate selection decision making at the server may require additional server-side software and/or hardware resources. In addition, servers may require information about clients, such as client capabilities (e.g., hardware and software), and network bandwidth information.

[0003] Client bitrate selection of streaming media can be a complex problem. One solution is for a client to adjust its buffer to account for startup-delay and changes in network bandwidth during downloading of a fixed-bitrate stream covering a program. However, this solution can lead to rebuffering situations which interrupt the client viewing experience.

[0004] Therefore, there exists ample opportunity for improvement in technologies related to client side rate control and indexed file layout for streaming media.

SUMMARY

[0005] In summary, the detailed description is directed to various techniques and solutions for optimized client side rate control and indexed file layout for streaming media.

[0006] According to one aspect of the techniques and solutions described herein, an indexed file layout for segmented streaming of multimedia content comprises one or more files containing index information for a program. The index information can comprise program description information describing the program and streaming segment index information describing each streaming segment of the program. In addition, the layout can comprise one or more file containing streaming segments of the program, where the streaming segments are each encoded at one or more bitrates independently of other streaming segments of the program. The layout supports client switching between different bitrates at segment boundaries.

[0007] According to another aspect of the techniques and solutions described herein, a method for optimizing client-side rate control of streaming content comprises, for each of a plurality of states, selecting one or more available paths that end in the state, where the one or more available paths are those paths ending in the state that satisfy a constraint condition, selecting a best path to the state from the one or more available paths (e.g., based on a cost measure such as a distortion measure), and eliminating as available all of the one or more available paths other than the best path. The method can also comprise downloading (e.g., by a client from a server) one or more streaming segments encoded at bitrates corresponding to the best path through the plurality of states, where each of the plurality of states corresponds to a specific bitrate of a specific streaming segment.

[0008] According to yet another aspect of the techniques and solutions described herein, a method for optimizing client-side rate control of streaming content comprises, for each of a plurality of states, selecting one or more available paths that end in the state, where the one or more available paths are those paths ending in the state that satisfy a constraint condition, and retaining the one or more available paths as possible paths to be included in a best end-to-end path. The method also comprises performing an end-to-end analysis of the possible paths to determine the best end-to-end path, where the best end-to-end path has the lowest overall distortion. The method can also comprise downloading (e.g., by a client from a server) one or more streaming segments encoded at bitrates corresponding to the best end-to-end path through the plurality of states, where each of the plurality of states corresponds to a measure of client buffer fullness for a specific bitrate of a specific streaming segment.

[0009] The described techniques and tools for solutions for optimized client side rate control and indexed file layout can be implemented separately or in combination.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] FIG. 1 is a block diagram of a system for segmented streaming of multimedia content using an indexed file layout.

[0011] FIG. 2 is a block diagram depicting an indexed file layout for segmented streaming.

[0012] FIG. 3 depicts an example diagram of streaming segments.

[0013] FIG. 4 depicts an example block diagram of file segments.

[0014] FIG. 5 depicts an example block diagram of an indexed file layout.

[0015] FIG. 6 depicts example rate control states.

[0016] FIG. 7 is a block diagram illustrating an example of a computing environment that can be used to implement any of the technologies described herein.

DETAILED DESCRIPTION

[0017] The following description is directed to techniques and solutions for optimized client side rate control and indexed file layout for segmented streaming of multimedia content. The various techniques and solutions can be used in combination or independently. Different embodiments can implement one or more of the described techniques and solutions.

Discussion of Terms

[0018] Program. A program is a self-contained piece of audio and/or video (e.g., multimedia) content. A program can contain multiple audio and/or video tracks. For example, a program could be a movie (e.g., comprising a stereo audio track, a multi-channel surround sound audio track, and a video track), a video clip (e.g., comprising one or more audio and video tracks), or an audio recording (comprising one or more audio tracks).

[0019] Streaming segment. A streaming segment is a section of a program for a specific track, and a program is divided into one or more streaming segments. Each streaming segment is self-contained, meaning a client can start playback at the beginning of a streaming segment without reference to prior (or subsequent) streaming segments. Streaming segments define the boundaries at which a client can change the bitrate of the content it is downloading.

[0020] In a specific implementation, a streaming segment has the following constraints:

- [0021]** every video encoding used for streaming contains a closed Group of Pictures (GOP) boundary (key frame) at every streaming segment boundary
- [0022]** each streaming segment touches (but does not overlap with) one or two other streaming segments
- [0023]** all streaming segments combine to exactly cover the presentation time range of the entire program (i.e., streaming segments represent a complete non-overlapping segmentation of the program presentation time range)

[0024] File Segment. A file segment is a file-level organization of streaming segments. A file segment contains one or more streaming segments.

[0025] In a specific implementation, a file segment has the following constraints:

- [0026]** each file segment touches (but does not overlap with) one or two other file segments
- [0027]** all file segments combine to exactly cover the presentation time range of the entire program
- [0028]** the presentation time range corresponding to any given streaming segment is fully contained within the time range corresponding to some single file segment (i.e., no streaming segment may span multiple file segments)

[0029] Index information. Index information is meta-data information other than actual program content (encoded audio and/or video data). Index information can include program description information (e.g., program length, number and type of tracks, etc.), streaming segment index information (e.g., comprising size, quality, and complexity of streaming segments), and file-level index information.

I. Indexed File Layout

[0030] This section describes indexed file layout techniques and solutions that can be applied to segmented streaming of multimedia content. For example, indexed file layout solutions can include one or more of the following features and techniques: dividing a program into multiple streaming segments, providing a variable number of encoded bitrates per streaming segment, providing index information covering all available bitrates to facilitate pure client-side bitrate selection, and organization of streaming segments into files to provide efficient streaming of content.

[0031] In general, the file layout described herein can be used by standard HTTP servers to serve multimedia content at multiple bitrates with bitrate selection (rate control) being performed client-side (e.g., exclusively client-side). Clients can perform rate control by first obtaining index information from the server describing the various bitrates available for streaming segments of a program. Based on the index information, and possibly other information (e.g., network bandwidth, buffer information, etc.), the client can decide which bitrate streaming segments to download from the server to provide a desired user experience (e.g., the best user experience possible based on the available bitrates and current network conditions).

[0032] Other types of computing devices (e.g., other than traditional HTTP servers) can provide files using the indexed file layout described herein. For example, a computing device (e.g., a personal computer, server computer, or special-purpose streaming media server) can use the indexed file layout to serve multimedia content using various file serving proto-

cols (e.g., File Transfer Protocol (FTP), Hypertext Transfer Protocol (HTTP), Real Time Streaming Protocol (RTSP), MMS (Microsoft Media Services), etc.).

[0033] In order to support bitrate switching, programs are divided into temporal chunks called streaming segments (self-contained units). The server stores each streaming segment at one or more bitrates (e.g., each streaming segment—bitrate combination is a separate streaming segment encoding). Each streaming segment includes one or more available bitrate encodings for a specific track (e.g., a specific audio track, such as an English audio track, or a specific video track) of a program. Clients then determine which bitrate, from the available bitrates (e.g., from the available streaming segment encodings), to download for each streaming segment. For example, a client may obtain a first streaming segment, for a video track, encoded at 250 Kb/sec (kilo-bits per second) (from one or more available streaming segment encodings for the first streaming segment), a second streaming segment, for the video track, encoded at 500 Kb/sec (from one or more available streaming segment encodings for the second streaming segment), and a third streaming segment, for the video track, encoded at 1 Mb/sec (mega-bit per second) (from one or more available streaming segment encodings for the third streaming segment). Each streaming segment encoding can be at a constant bitrate (CBR) or at a variable bitrate (VBR) (and the use of CBR and VBR can be mixed on a per streaming segment encoding basis).

[0034] FIG. 1 depicts a generalized block diagram 100 of a system for segmented streaming of multimedia content using an indexed file layout. In the diagram 100, a server 110 (e.g., a server computer system such as a standard HTTP server) provides multimedia content to a client 120 (e.g., a client computer system, such as a laptop or desktop computer, or another type of computing device, such as a PDA or mobile phone) via a network 130 (e.g., the Internet). In the diagram 100, the server 110 stores programs in an indexed file layout. The client 120 comprises client-side rate control software and/or hardware.

[0035] In a specific implementation, the server 110 is a standard HTTP server without any specialized streaming capability other than the ability to serve files. Because the server 110 does not support any specialized bitrate selection capability, the client 120 must perform all bitrate selection activities. In this implementation, the client 120 performs all bitrate selection activities. For example, the client 120 can perform rate control using the index information obtained from the server 110 (e.g., alone or in combination with other information, such as client buffer information, network bandwidth, etc.).

[0036] FIG. 2 is a block diagram 200 depicting an indexed file layout for segmented streaming. In the diagram 200, one or more files contain index information 210 and one or more files contain streaming segments 220. The index information 210 includes program description information, file-level index information, and streaming segment index information. In some implementations, some of the listed index information may not be present and/or other types of index information may be present in addition to the listed information. The streaming segments 220 are content files for streaming segments of a program (streaming segments of encoded audio and/or video tracks).

[0037] FIG. 3 is a block diagram 300 depicting the organization of streaming segments. At 300, a program (e.g., a multimedia program with a number of audio and/or video

tracks) is divided into streaming segments 1 through N for each track of the program. In general, a program can be divided into an arbitrary number of streaming segments. Each streaming segment includes the bitrate encodings for its audio or video tracks of the program for the period of the streaming segment. Each streaming segment can be encoded at a different number of bitrates, where each streaming segment encoded at a specific bitrate is a different streaming segment encoding. For example, streaming segment 1 of track 1 can be encoded at 128 Kb/sec (one available streaming segment encoding for streaming segment 1 of track 1). Streaming segment 1 of track 2 can be encoded at 96 Kb/sec and 192 Kb/sec (two available streaming segment encodings for streaming segment 1 of track 2).

[0038] At 310, example streaming segment encodings for track 1 are depicted. In the example, track 1 is an audio track (e.g., an English language audio track of the program). Streaming segment 1 of track 1 is encoded at one bitrate (e.g., 128 Kb/sec). Streaming segment 2 of track 1 is encoded at two bitrates (e.g., 96 Kb/sec and 192 Kb/sec), representing two available streaming segment encodings for streaming segment 2 of track 1.

[0039] At 320, example streaming segment encodings for track 2 are depicted. In the example, track 2 is a second audio track (e.g., a French language audio track of the program). Streaming segment 1 of track 2 is encoded at two bitrates (e.g., 128 Kb/sec and 192 Kb/sec), representing two available streaming segment encodings for streaming segment 1 of track 2. Streaming segment 2 of track 2 is also encoded at two bitrates (e.g., 128 Kb/sec and 256 Kb/sec), representing two available streaming segment encodings for streaming segment 2 of track 2.

[0040] At 330, example streaming segment encodings for track 3 are depicted. In the example, track 3 is video track. Streaming segment 1 of track 3 is encoded at two bitrates (e.g., 256 Kb/sec and 512 Kb/sec), representing two available streaming segment encodings for streaming segment 1 of track 3. Streaming segment 2 of track 3 is encoded at three bitrates (e.g., 200 Kb/sec, 400 Kb/sec, and 830 Kb/sec), representing three available streaming segment encodings for streaming segment 2 of track 3.

[0041] Allowing for different bitrate encodings between streaming segments can provide greater efficiency and quality of streaming multimedia content. For example, if the audio content of streaming segment 1 of track 1 is less complex, it can be encoded at fewer and/or lower bitrates. If the audio content of streaming segment 2 of track 1 is more complex, it can be encoded at more and/or higher bitrates.

[0042] Streaming segments can have the same start and stop time between tracks, or they can have different start and stop times. If streaming segments are generally aligned across tracks, then clients can easily switch bitrates at streaming segment boundaries. In some implementations, corresponding streaming segments of audio tracks and video tracks are closely (but not necessarily exactly) aligned. For example, in FIG. 3, streaming segment 2 of audio tracks 1 and 2 are closely (but not exactly) aligned with streaming segment 2 of track 3 (the video track).

[0043] FIG. 4 is a block diagram 400 depicting file segments. In a specific implementation, the streaming segments of a program are organized by file segment. Each file segment contains one or more consecutive streaming segments for one or more tracks. All file segments of a program combine to cover the entire program.

[0044] In the example 400, a program (with one audio and one video track) has been divided into two file segments. File segment 1 includes two streaming segments of the audio track, encoded at one bitrate (e.g., 128 Kb/sec). File segment 2 includes two streaming segments of the video track, encoded at two bitrates (e.g., 256 Kb/sec and 512 Kb/sec).

[0045] A file segment are used to organize content into physical files. A physical file can be created by combining one or more file segments from one or more track.

[0046] Streaming segments are stored in files based on file segment. In a specific implementation, a separate physical file is used to store each set of streaming segments of a file segment for a specific bitrate. Using file segment 1 depicted in 400, the following files can be used to store the streaming segments:

[0047] file1.aud (contains both streaming segments of audio bitrate 1)

[0048] file1_bitrate1.vid (contains both streaming segments of video bitrate 1)

[0049] file1_bitrate2.vid (contains both streaming segments of video bitrate 2)

[0050] In another implementation, a separate physical file is used to store each set of streaming segments of the file segments on a per-track basis. Using file segment 1 depicted in 400, the following physical files would be used to store the streaming segments:

[0051] file1_aud.seg (contains both streaming segments of audio bitrate 1)

[0052] file1_vid.seg (contains both streaming segments of video bitrates 1 and 2)

[0053] In yet another implementation, a separate physical file is used to store each set of streaming segments of the file segments for more than one track (e.g., for all tracks of the program). Using file segment 1 depicted in 400, the following physical files would be used to store the streaming segments:

[0054] file1.seg (contains both streaming segments of audio bitrate 1 and video bitrates 1 and 2)

[0055] In some implementations, streaming segments are aligned across all audio and/or video tracks of a program. Aligning streaming segments allows for switching between available bitrates at segment boundaries. In general, video segment boundaries occur such that video segments are self-contained (e.g., boundaries occur at group-of-frame (GOP) boundaries).

[0056] In other implementations, streaming segments for audio and video can differ. For example, an audio track could be encoded at a single bitrate for the duration of a program (or for the duration of a file segment), while a video track could be encoded at multiple bitrates for the duration of the program (or for the duration of the file segment). For example, file segment 1 depicted in 400 could contain one streaming segment for the audio track (instead of the two depicted), and file segment 2 could contain and two streaming segments for the video track (as depicted).

[0057] FIG. 5 is a block diagram 500 depicting an example indexed file layout. In general, the example layout 500 includes index information (510, 520, 530, and 540) describing the program, and encoded program content (streaming segments) 550 and 560.

[0058] The example file layout includes program description information 510 (e.g., general program information such as duration and track information). The example file layout includes file-level index information 520, describing the file segments of the program. The example file layout includes

streaming segment indexes for the file segments of the program. In the example, two sets of streaming segment indexes are depicted, **530** and **540**, for each of two sets of file segments. The example file layout also includes two sets of file segments **550** and **560**.

[0059] In the example file layout **500**, a first set of three streaming segments (**550**) is grouped into two file segments: one file segment covering an audio track at one bitrate and another file segment covering a video track at three bitrates. A second set of four streaming segments (**560**) is grouped into two file segments: one file segment covering the audio track at one bitrate and another file segment covering the video track at four bitrates.

[0060] The example indexed file layout **500** can be organized into files in various ways. For example, the program description information **510** and file-level index information **520** can be stored in a single file, or in two separate files. The streaming segment index information **530** and **540** can be stored in two separate files, or they can be stored in the same file as the file-level index information **520**. Listed below in Table 1 is one way of storing the index information and program content:

TABLE 1

File Name	Description
main.dat	Program description information 510
index.dat	File-level index information 520
index0.idx	Streaming segment index for file segments (530)
index1.idx	Streaming segment index for file segments (540)
file0.aud	Audio bitrate 1 for file segment (550)
file0.vid	Video bitrates 1-3 for file segment (550)
file1.aud	Audio bitrate 1 for file segment (560)
file1.vid	Video bitrates 1-4 for file segment (560)

[0061] As depicted in the file segments **550** and **560**, not all segment encodings must be present (those with hash marks have not been encoded and thus are not available to a client). For example, the second segment of video bitrate **3** (**550**) has not been encoded. One reason for not encoding the second segment of video at bitrate **3** could be that such an encoding would offer little improvement in quality over the encoding at bitrate **2** (e.g., the segment may cover a transition, such as a fade-in or fade-out). The fact that some encodings are not available can be indicated in the streaming segment indexes (**530** and **540**) so that the client will know that such bitrates are not available.

Example File Layout

[0062] In a specific implementation, the indexed file layout allows up to four types of files, as described below.

[0063] 1. A single Program Description file (for a given program) containing:

[0064] 1.1 Overall program description information includes information such as full duration and what kind of data the file contains (audio, video, what kind of audio—stereo, multichannel, how many languages, which streams are encoded using multiple bitrates, etc).

[0065] 1.2 For each high-level (user-exposed, such as audio or video) track, information about whether the Main Program Index for that track is contained in the program description file or in a separate file (in which case a partial file name is also specified, enabling the client to derive the full path to the main index file).

[0066] 1.3 Optionally, the Main Program Index(es) (unless 1.2 indicates that it is in a separate file). If present, this follows the same format as 2 below.

[0067] 2. Optionally (depending on 1.2), a separate file containing the Main Program Index containing:

[0068] 2.1 Optionally, a File Level index.

[0069] 2.2 Optionally a Streaming Segment Index describing all of the Streaming Segment encodings and bitrates available within the entire Program.

Note: At least one of {2.1, 2.2} must be present.

[0070] 3. Optionally, a number of Streaming Segment Index files (one for each File Segment) describing, for each File Segment, all of the Streaming Segment encodings and bitrates available within that File Segment. The filename of each Streaming Segment Index files must be derivable in some predictable fashion and contain the index (number) of the File Segment.

Note: Either 3 or 2.2 must be present.

[0071] 4. A number of Multimedia Data files containing compressed multimedia data. The filename of each Multimedia Data file must be derivable in some predictable fashion and contain the index (number) of the File Segment, as well as the index (number) of the bitrate contained in the file.

[0072] In this specific implementation, the File-Level Index (2.1) is a simple mapping of presentation time to File Segment. It is an array of start presentation times (or, equivalently, presentation durations) for each File Segment. In the event that all File Segments have equal durations, File-Level Index can be special-cased using a single value specifying the fixed File Segment Duration.

[0073] In the specific implementation, the Streaming Segment Index (2.2 or 3) contains the following information for each Streaming Segment. All fields are equally applicable in the context of 2.2 as well as in the context of 3, unless otherwise noted.

[0074] 2.2.1 Start presentation time (or, equivalently, duration) of the Streaming Segment. If presentation time, the base (frame of reference) for this presentation time is different depending on where the streaming-segment-level index is found. For a complete streaming-segment-level index found in a main program index file (2.2), the presentation time is relative to the beginning of the entire program. For a streaming-segment-level index found in File Segment index files (3), the presentation time is relative to the beginning of that File Segment.

[0075] 2.2.2 (Applicable to 2.2 only—n/a for 3) in a Streaming Segment index covering the entire program, the index (number) of the File Segment to which this Streaming Segment corresponds.

[0076] 2.2.3 The number of different bitrate encodings available for the streaming segment.

[0077] 2.2.4 For each bitrate encoding of a Streaming Segment, the following additional information:

[0078] 2.2.4.1 Some (possibly lossily compressed, approximate) representation of the size in bytes of that encoding.

[0079] 2.2.4.2 For video, the coded screen resolution (dimensions) used in that encoding.

[0080] 2.2.4.3 Some measure of the coded quality. This may be a scalar or a vector depending on whether “quality” is a single number appropriate for all use scenarios or a function of some parameters of the use scenario such as actual playback device screen resolution.

[0081] 2.2.2.4 Optionally, some measure of the decoding complexity (e.g., millions of instructions) of the encoding.

[0082] 2.2.2.5 File offset (into the audio or video data file) of the data corresponding to this Streaming Segment. This field may be omitted if the File Segment to Streaming Segment mapping is 1-to-1.

[0083] Both types of index files can be compressed to reduce index file sizes. At the very least, timestamps (if stored as presentation times as opposed to durations) and file offsets (2.2.2.5) should undergo differential coding. Some other fields (likely 2.2.4.2 and 2.2.4.3) should be coded using type-specific techniques (e.g., as enums) taking into account the set of possible values. All fields could further undergo entropy coding.

[0084] In alternative implementations, it may be advantageous to organize the index files differently (or store slightly different information) for different target client platforms, e.g., PC clients vs. portable devices. The Multimedia Data files do not have to be stored separately in such a case—i.e., multiple (independent) sets of Program Description and Index files could point to the same underlying set of Multimedia Data files. To make this possible, some portion of the Multimedia Data file naming scheme could be stored in the Program Description or Index files.

II. Client Side Rate Control

[0085] This section describes client side rate control (e.g., optimal client side rate control) techniques and solutions that can be applied to streaming of multimedia content (e.g., programs or pieces of programs). Client side rate control can be performed using the indexed file layout techniques and solutions described herein. For example, a client system (e.g., computer or other type of computing device) can download indexed information from a server and use the downloaded indexed information to select which bitrate to download for each segment. Using the techniques and solutions described in this section enables pure client-side bitrate selection without any special logic required at the server-side (other than standard file-serving capabilities, such as found on an HTTP server). More details regarding client side rate control techniques and solutions are described below.

[0086] Suppose we have M units of compressed data, each taking $b[c, n]$ bits where $c=0, 1, \dots, C[n]-1$, where $C[n]$ is the number of possible rates available for the n'th unit, and $n=0, 1, \dots, M-1$. Suppose for each of these units of compressed data, we also have a distortion function that we wish to minimize and that the distortion function is additive, given by $d[c, n]$. Suppose each unit of compressed data corresponds to $t[n]$ units of time and suppose that the average bitrate that can be transmitted across the channel over this time period is given by $R[c, n]$ bits/second which will be a function of network conditions, the amount of time, and also a function of the bitrate being requested. In many cases, we can assume the channel rate to not be a function of c, that is $R[c, n]=R[n]$ for all c. However, there may be certain cases where this is not

true. For example in a content delivery network (CDN), some bitrate versions of the units may be cached closer to the client in which case the bitrate between client and server can be a function of not only network conditions, but also a function of the content. In the most general case, we can assume it to be a function of the content. Corresponding to this rate we can compute the approximate time that it takes to download each unit. Let $s[c, n]=b[c, n]/R[c, n]$ be the time it takes to download the unit c'th unit at time n.

[0087] Suppose a client chooses to download unit $u[n]$ for the n'th unit. Then the cumulative distortion that the client will have is given by:

$$D = \sum_{i=0}^{M-1} d[u[i], i]. \quad \text{Equation 1}$$

The goal of an optimal rate control algorithm is to minimize D subject to the constraints that the decoder buffer never glitches for more than the desired time period given the start-up latency and network conditions. Suppose the maximum start up latency is given by T_0 seconds, and that the maximum glitch time is given by T_g seconds. If $T_g=0$, then the constraints essentially can be stated as:

[0088] 1. Time to download first unit is less than or equal to T_0

[0089] 2. Time to download first two units is less than or equal to $T_0+t[0]$

[0090] 3. Time to download first three units is less than or equal to $T_0+t[0]+t[1]$

[0091] 4. And so on

The cumulative value of the durations is also referred to as the “timestamp” of a given chunk which is often present in streams. That is $\sum_{i=0}^{n-1} t[i]$ is often called the timestamp of chunk n and is stored in the packet payload headers. This can be stated as:

$$T_0 + \sum_{i=0}^{n-1} t[i] \geq \sum_{i=0}^n s[u[i], i]. \quad \text{Equation 2}$$

This constraint has to be satisfied for all n, which can alternatively be written in terms of decoder buffer fullness given by:

$$f_d[n] = T_0 + \sum_{i=0}^n t[i-1] - s[u[i], i] \geq 0,$$

which has to be satisfied for all $n \geq 0$. The first unit constraint is for $n=0$, where $t[-1]=0$. An alternative representation for the decoder buffer fullness is given by the following recursive equation:

$$f_d[n] = f_d[n-1] + t[n-1] - s[u[n], n],$$

with the initial condition:

$$f_d[-1] = T_0.$$

If the n'th unit is allowed to glitch for $T_g[n]$ seconds, then we can write the constraint in a recursive way using the following decoder buffer fullness:

$$g_d[n] = f_d[n-1] + t[n-1] - s[u[n], n] \quad \text{Equation 3}$$

$$f_d[n] = \begin{cases} -\infty & \text{if } g_d[n] < -T_g[n] \\ 0 & \text{if } -T_g[n] \leq g_d[n] \leq 0 \\ g_d[n] & \text{if } g_d[n] > 0. \end{cases}$$

The client has to choose $u[n]$ to satisfy $f_d[n] \geq 0$ while minimizing D .

[0092] Also, the actual buffer fullness that the decoder has may be different simply because the decoder only knows the estimated download time for each chunk and does not know the actual download time. Suppose at any given instant the decoder knows the actual buffer fullness. Then, based upon the estimates for download time for each component it can make a decision into the future as to which units to download.

[0093] The problem of minimizing D subject to the buffer constraint $f_d[n] \geq 0$ for all n can be solved using dynamic programming. Suppose we find the minimum distortion possible that satisfies the buffer constraint and such that $u[M-1] = c$ for all possible $c=0, 1, \dots, C[M-1]-1$. The minimum distortion will be the minimum of all such distortions. Now the problem becomes finding the minimum possible distortion with the buffer constraint and with an additional constraint that $u[n]=c$. Such a minimization problem can be solved using a trellis to store relevant information needed in the optimization.

[0094] Since the distortion being minimized is assumed to be additive, a solution to find the optimal path can be done using a trellis. Suppose the decoder has a buffer fullness of $f_d[N_0]$ after downloading some number of units. In addition, suppose it knows distortion $d[c, n]$ and an estimate of the download time $s[c, n]$ for all c for $n=N_0+1, N_0+2, \dots, N_0+M$. Without loss of generality assume $N_0=0$. Define the following:

[0095] 1 Let $D[c, n]$ be the minimum distortion which can be achieved without glitches such that $u[n]=c$. If no such solution exists, let $D[c, n]$ be the minimum distortion that can be achieved while minimizing the total glitch time.

[0096] 2 Let $g[c, n]$ be the total glitch time achieved when doing 1.

[0097] 3 Let $f_{d,c}[C, n]$ be the decoder buffer fullness when doing 1.

[0098] 4 Let $u_c[c, n]$ be the set of optimal encodings chosen when doing 1. Let $C[n]$ be the number of rates available at time n .

[0099] 6 Let $t[n]$ be the duration of unit n .

[0100] 7 Let $d[c, n]$ be the distortion of the c 'th unit n .

[0101] 8 Let $s[c, n]$ be the estimated download time of the c 'th unit n .

Then it can try to make an optimal decision as to which units to download up to the N_0+M unit. Then, by definition, since the distortion is additive, we can write

$$D[c, n] = \begin{cases} \min_{l \in \gamma[c, n]} (D[l, n-1]) + d[c, n] & \text{if } |\gamma[c, n]| \geq 1 \\ \min_{l \in \sigma[c, n]} (D[l, n-1]) + d[c, n] & \text{if } |\gamma[c, n]| = 0, \end{cases} \quad \text{Equation 4}$$

where γ_n is the set of l from $l=0, 1, \dots, C[n-1]-1$ which satisfy the buffer constraint and which contain no glitches. That is:

$$\gamma[c, n] = \{l \mid l = 0, 1, \dots, C[n-1]-1 \cap f_{d,c}[l, n-1] + t[n-1] - s[c, n] \geq -T_g[n] \cap g[l, n-1] = 0\}. \quad \text{Equation 5}$$

If no such set exists, i.e. cardinality of it is 0, then we take the minimum over those l which satisfy the minimum glitch above the allowed glitches, that is:

$$\sigma[c, n] = \{l \mid l = 0, 1, \dots, C[n-1]-1 \cap (T_g[n] - \min(f_{d,c}[l, n-1] + t[n-1] - s[c, n], -T_g[n]) + g[l, n-1]) = g_{\min}\}. \quad \text{Equation 6}$$

$$g_{\min}[c, n] = \min_{l=0,1,\dots,C[n-1]-1} T_g[n] - \min(f_{d,c}[l, n-1] + t[n-1] - s[c, n] - T_g[n]) + g[l, n-1] \quad \text{Equation 7}$$

$$l_{\min}[c, n] = \begin{cases} \operatorname{argmin}_{l \in \gamma[c, n]} (D[l, n-1]) + d[c, n] & \text{if } |\gamma[c, n]| \geq 1 \\ \operatorname{argmin}_{l \in \sigma[c, n]} (D[l, n-1]) + d[c, n] & \text{if } |\gamma[c, n]| = 0 \end{cases} \quad \text{Equation 8}$$

Then, the updates for Equations 2, 3, and 4 can be done using:

$$g[c, n] = \begin{cases} 0 & \text{if } |\gamma[c, n]| \geq 1 \\ g_{\min}[c, n] & \text{if } |\gamma[c, n]| = 0 \end{cases} \quad \text{Equation 9}$$

$$f_{d,c}[c, n] = \max(f_{d,c}[l_{\min}[c, n], n-1] + t[n-1] - s[c, n], 0) \\ u_c[c, n] = \{u_c[l_{\min}[c, n], n-1], c\}$$

For the initial condition, we can assume:

$$\begin{aligned} C[0] &= 1 \\ D[0,0] &= 0 \\ g[0,0] &= 0 \\ f_{d,c}[0,0] &= f_d[N_0] \\ u[0,0] &= \{0\} \end{aligned} \quad \text{Equation 10}$$

Once the algorithm updates are run from $n=1, \dots, M$, the optimal distortion can be found using:

$$D = \begin{cases} \min_{c \in \gamma} D[c, M] & \text{if } |\gamma| \geq 1 \\ \min_{l \in \sigma} D[c, M] & \text{if } |\gamma| = 0 \end{cases} \quad \text{Equation 11}$$

where γ is the set of c from $c=0, 1, \dots, C[M-1]-1$ which satisfy the buffer constraint without any glitches.

$$\gamma = \{c \mid c=0, 1, \dots, C[M-1]-1 \cap g[c, M]=0\}. \quad \text{Equation 12}$$

If the cardinality of this set is 0 (i.e. no glitchless solution exists), then the minimum distortion solution is chosen which achieves the minimum glitch.

$$\sigma = \{c \mid c = 0, 1, \dots, C[M-1]-1 \cap g[c, M] = g_{\min}\}. \quad \text{Equation 13}$$

Where:

$$g_{\min} = \min_{c=0,1,\dots,C[M-1]-1} g[c, M]. \quad \text{Equation 14}$$

Let:

$$c_{\min} = \begin{cases} \operatorname{argmin}_{c \in \gamma} D[c, M] & \text{if } |\gamma| \geq 1 \\ \operatorname{argmin}_{c \in \sigma} D[c, M] & \text{if } |\gamma| = 0 \end{cases} \quad \text{Equation 15}$$

Then, the optimal selection of rates is given by:

$$u = \{u[n] \mid n=0, \dots, M\},$$

where:

$$u = u_c[c_{\min}, M]$$

$$u[n] = u_c[c_{\min}, M][n] \quad \text{Equation 16}$$

[0102] Note that in the equations above we could simply use the definition of γ for the set γ since if $|\gamma| \geq 1$, then the minimum glitch by definition will be 0, which would simplify the equations a bit. However, since different definitions for the “optimal” solution could be used when there is a glitch such as minimizing the number of glitches or minimizing the maximum glitch time, we have left the two sets as separate.

[0103] Pseudocode for the algorithm can be the following if for purposes of simplicity we assume $N_0=0$.

```

Set initial conditions as defined above in Equation 10
For n=1 to M
  For c=0 to C[n]-1
    Find optimal solution ending in state c at unit n using
    Equations 4-8 by looping over l
    Update variables for state (c,n) using Equation 9
  End
End
Find optimal solution ending in state M using Equations 11-16.

```

[0104] Using the above pseudocode, the optimal path can be found. If there are changes in decoder buffer fullness due to differences in the estimated time it takes to download a unit vs. the actual time it takes to download a unit, or if there are updates to the estimates themselves, the algorithm can be rerun with the current state.

[0105] Several other modifications could be done. If the player speed is different than the actual time, i.e. in trick play modes such as fast forward or fast play, then the time axis can be easily scaled when running the algorithm to compensate for this. For example, when playing at 2x, a 5 second decoder buffer can be thought of as only being 2.5 seconds, and a 10 second unit duration can be thought of as only being 5 seconds.

[0106] Also, since download times are usually only estimates, we can come up with a probability distribution on the download time, that is treat $s[u[n], n]$ as a random variable with some probability distribution. In the trellis, instead of

choosing whether the decoder will definitely starve or not, we can simply change the buffer constraint to be something like the following,

$$\text{Probability}(f_d[n] \geq 0) > 1 - \epsilon,$$

where ϵ is some small value.

[0107] This basically means that we will likely choose the worst case estimate for download time which would force the client to make very conservative decisions when the buffer is small but allow larger bitrates to be downloaded as the buffer grows. In the pseudocode, the only thing that would change is that

$$\text{Fullness} > -T_g[n]$$

Would be replaced by

$$\text{Probability}(\text{Fullness} > -T_g[n]) > 1 - \epsilon.$$

This is essentially the same as choosing a value S for $s[c, n]$ such that:

$$\text{Probability}(s[c, n] > S) < \epsilon$$

when running the trellis. However, based upon the actual value for $s[c, n]$, the decoder buffer should be correctly adjusted before making future decisions so as to not let the worst case estimate affect future decisions.

[0108] For the distortion metric, various metrics could be chosen such as mean square error. Simpler metrics such as lost bit throughput could also be used for the distortion metric.

Optimality of Solution

[0109] The distortion metric chosen can determine the optimality of the solution arrived at by using the dynamic programming approach described above. If the distortion $d[c, n]$ of units across bitrates $c=0, 1, \dots, C[n]-1$, is different by orders of magnitude, then the algorithm is likely to yield the optimal solution under practical conditions. If the distortions are not different by orders of magnitude, then the algorithm could yield a sub-optimal solution, because of the nature of the trellis.

[0110] In Equation 4, since γ is a function of c (i.e. the units in stage n that are available for download, depend on the buffer fullness $f_{d,c}[C, n-1]$ after downloading unit c in stage $n-1$), it is possible that a unit with low distortion in stage n becomes un-available to satisfy the no-glitch condition. This precludes this unit from being a part of the solution.

[0111] A potential solution to this problem is to not use just the distortion as a cost function at each stage, and instead use a function of the distortion and buffer fullness as the cost function. Such a cost function, and its impact on the optimal solution under practical network conditions is to be determined.

Practical Considerations

[0112] In practical implementations, network conditions are bound to keep varying as units of compressed data are downloaded. This means that the estimated time, $s[c, n]$, taken to download the c 'th unit n , will keep varying as new estimates of the network bandwidth are computed. Thus the optimal path needs to be re-computed using the algorithm described above each time a new network bandwidth estimate is computed. In order to reduce the computation required, the

loop in the above algorithm, need not be run from $n=1 \dots M$, and instead can be stopped at stage $n=n_{end}$ on equation 8, when it is found that

$$l_{min}[c,n]=k, \text{ where } \{c|c=0, 1, \dots, C[n]-1\} \text{ and } n=n_{end} \text{ and } k \text{ is a constant.}$$

[0113] In other words the dynamic programming loop should be run until only one of the candidate paths survives. Once the surviving path is identified, $u_c[c_{min}, 0]$, identifies the first unit to be downloaded.

Alternative Solution

[0114] As an alternate solution to solve the optimality issue raised in the above algorithm, with slightly increased complexity, the definition of a state is changed. Instead of defining the states to be the various bitrates available for a given streaming segment, define the states to the various levels of decoder buffer fullness at a given time instant. Define θ_n be the set of $P[n]$ quantized decoder buffer fullness states available at time n . For example, $\theta_n[0]=[0, 0.1)$ is the state where the decoder buffer fullness is between 0 and 0.1 seconds. Let $d[p, n]$ be the minimum distortion possible at time n when the decoder buffer fullness is contained in $\theta_n[p]$. Let $f_d[p, n]$ be the actual decoder buffer fullness corresponding to this. To run the optimization algorithm we can simply do:

$$D[p, n] = \min(D[l, n-1] + d[c, n])$$

Where $(l, c) \in \gamma_{p,n}$, where:

$$\gamma_{p,n} = \{(l, c) | l=0, 1, \dots, P[n-1]-1, c=0, 1, \dots, C[n]-1, f_d[l, n-1] + t[n-1] - s[c, n] \in \theta_n[p]\}.$$

Let:

$$(l_{min}[p, n], c_{min}[p, n]) = \arg \min(D[l, n-1] + d[c, n]).$$

Then the decoder buffer fullness is updated according to:

$$f_d[p, n] = f_d[l_{min}[p, n], n-1] + t[n-1] - s[c_{min}[p, n], n].$$

The optimal decoder path is updated according to:

$$u_c[p, n] = \{u_c[l_{min}[p, n], n-1], c\}.$$

[0115] As before, we do end up eliminating possible paths which satisfy the constraint due to taking the minimum distortion path into each state. However, if the quantization of the buffer states is done fine enough, the suboptimality is reduced. For example, if we did not quantize the set of possible buffer states, the algorithm would be optimal since we truly minimize the distortion for the given buffer constraints.

[0116] In addition, since smaller buffer states are more likely important in determining an optimal path (since if the buffer is large, the optimal thing to do is to try to simply take the largest bitrate, smallest distortion choice), it is recommended that the buffer states in this algorithm be unequally spaced. For example, the first buffer state can be $[0, 0.1)$, the second could be $[0.1, 0.3)$, the third could be $[0.3, 0.7)$, and so on, so that larger buffer states have larger quantization width.

EXAMPLE

[0117] Consider the following simple example. Suppose we have 4 streaming segments, with the following distortions, durations, and estimated download times.

$$T_0 = \text{initial latency} = 5$$

$$\text{Durations} = t[1] \ t[2] \ t[3] \ t[4] = 1 \ 1 \ 2 \ 2$$

$$\text{Number of bitrates available} = C[1] \ C[2] \ C[3] \ C[4] = 3 \ 3 \ 4 \ 2$$

Distortion for Each Streaming Segment (First Row Gives Distortion for Each of 3 Bitrates for First Segment)

[0118]

$$d[i, j] = \begin{matrix} & 3 & 2 & 1 \\ & 3 & 2 & 1 \\ 4 & 3 & 2 & 1 \\ 3 & 1 & & \end{matrix}$$

Estimated Download Time for Each Streaming Segment (First Row Gives Distortion for Each of 3 Bitrates for First Segment)

[0119]

$$s[i, j] = \begin{matrix} & 1 & 2 & 3 \\ & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 3 & & \end{matrix}$$

Total Distortion for Each State at Time $n=1, 2, 3, 4$

$$\begin{matrix} & 3 \\ D[c, 1] = & 2 \\ & 1 \\ & 4 \\ D[c, 2] = & 3 \\ & 2 \\ & 6 \\ & 6 \\ D[c, 3] = & 6 \\ & 5 \\ & 9 \\ D[c, 4] = & 7 \end{matrix}$$

Total Glitch Time for Each State at Time $n=1, 2, 3, 4$

$$\begin{matrix} & 0 \\ g[c, 1] = & 0 \\ & 0 \\ & 0 \\ g[c, 2] = & 0 \\ & 0 \\ & 0 \\ g[c, 3] = & 0 \\ & 1 \\ & 0 \\ g[c, 4] = & 1 \end{matrix}$$

Decoder Buffer Fullness

[0120]

$f_d[c, 1] = 3$
 $f_d[c, 2] = 1$
 $f_d[c, 3] = 0$
 $f_d[c, 4] = 0$

-continued

$f_d[p, 3] = 2$
 $f_d[p, 4] = 2$

Total Distortion

[0125]

Optimal Path Ending in State C

[0121]

$u[c, 1] = 1$
 $u[c, 2] = 2$
 $u[c, 3] = 2$
 $u[c, 4] = 2$

$D[p, 1] = 2$
 $D[p, 2] = 4$
 $D[p, 3] = 8$
 $D[p, 4] = 10$

[0122] Total minimum distortion by the algorithm is 9 since that is the minimum glitch free distortion returned. A full exhaustive search shows that the minimum distortion is 8 which can be found using the second solution.

[0123] Using the second solution, assume $P[n]=3, 5, 5, 5$ is the number of decoder buffer fullness states available.

[0124] Decoder buffer fullness and set of possible buffer fullness states θ_n (no buffer fullness quantization is done).

Optimal Path (State Here Refers to Buffer Fullness State)

[0126]

$f_d[p, 1] = 3$
 $f_d[p, 2] = 2$

$u[p, 1] = 1$
 $u[p, 2] = 0$

-continued

```

      0 0 0
      0 1 1
u[p, 3] = 0 2 2
      1 3 3
      2 4 4

      0 1 1 0
      0 2 2 1
u[p, 4] = 1 3 3 2
      2 4 4 3
      2 4 4 4

```

[0127] The two algorithms can be shown using the diagrams depicted in FIG. 6, where each dot represents a state, and the “optimal” state transitions are shown by solid lines. Some non-optimal paths are also shown as dashed lines. In the first case 610 (representing the first algorithm described above), there will always be a path from all states at $n-1$ to all states at n since the states represent possible bitrates. With the second algorithm 620, this will not be the case since there will not be possible path from all buffer states at $n-1$ to buffer states at n . The first dot in the first column represents “ $n=0$ ” and is only used for initialization. Similarly, the single dot in the last column is choosing of the optimal path. On the diagrams, “ t =duration”, “ s =estimated time to download”, “ T_0 =initial delay”, and “ f =decoder buffer fullness”.

III. Example Computing Environment

[0128] FIG. 7 illustrates a generalized example of a suitable computing environment 700 in which described embodiments, techniques, and technologies may be implemented. The computing environment 700 is not intended to suggest any limitation as to scope of use or functionality of the technology, as the technology may be implemented in diverse general-purpose or special-purpose computing environments. For example, the disclosed technology may be implemented with other computer system configurations, including hand held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers, and the like. The disclosed technology may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

[0129] With reference to FIG. 7, the computing environment 700 includes at least one central processing unit 710 and memory 720. In FIG. 7, this most basic configuration 730 is included within a dashed line. The central processing unit 710 executes computer-executable instructions and may be a real or a virtual processor. In a multi-processing system, multiple processing units execute computer-executable instructions to increase processing power and as such, multiple processors can be running simultaneously. The memory 720 may be volatile memory (e.g., registers, cache, RAM), non-volatile memory (e.g., ROM, EEPROM, flash memory, etc.), or some combination of the two. The memory 720 stores software 780 that can, for example, implement the technologies described herein. A computing environment may have additional fea-

tures. For example, the computing environment 700 includes storage 740, one or more input devices 750, one or more output devices 760, and one or more communication connections 770. An interconnection mechanism (not shown) such as a bus, a controller, or a network, interconnects the components of the computing environment 700. Typically, operating system software (not shown) provides an operating environment for other software executing in the computing environment 700, and coordinates activities of the components of the computing environment 700.

[0130] The storage 740 may be removable or non-removable, and includes magnetic disks, magnetic tapes or cassettes, CD-ROMs, CD-RWs, DVDs, or any other medium which can be used to store information and which can be accessed within the computing environment 700. The storage 740 stores instructions for the software 780, which can implement technologies described herein.

[0131] The input device(s) 750 may be a touch input device, such as a keyboard, keypad, mouse, pen, or trackball, a voice input device, a scanning device, or another device, that provides input to the computing environment 700. For audio, the input device(s) 750 may be a sound card or similar device that accepts audio input in analog or digital form, or a CD-ROM reader that provides audio samples to the computing environment 700. The output device(s) 760 may be a display, printer, speaker, CD-writer, or another device that provides output from the computing environment 700.

[0132] The communication connection(s) 770 enable communication over a communication medium (e.g., a connecting network) to another computing entity. The communication medium conveys information such as computer-executable instructions, compressed graphics information, or other data in a modulated data signal.

[0133] Computer-readable media are any available media that can be accessed within a computing environment 700. By way of example, and not limitation, with the computing environment 700, computer-readable media include memory 720, storage 740, communication media (not shown), and combinations of any of the above.

[0134] The indexed file layout and client side rate control techniques and solutions described in this application can be used in various combinations to provide more efficient organization and streaming of multimedia content. For example, a program can be divided into streaming segments, each streaming segment independently encoded at one or more bitrates (e.g., for each of one or more audio and/or video tracks of the program). Index information describing the program and available bitrates for the streaming segments can be stored. A client can download the index information and use it for client side rate control (e.g., to determine an efficient or optimal sequence of bitrates to download for each streaming segment).

[0135] Any of the methods described herein can be performed via one or more computer-readable media (e.g., storage or other tangible media) comprising (e.g., having or storing) computer-executable instructions for performing (e.g., causing a computing device, audio and/or video processing device, or computer to perform) such methods. Operation can be fully automatic, semi-automatic, or involve manual intervention.

[0136] Having described and illustrated the principles of our innovations in the detailed description and accompanying drawings, it will be recognized that the various embodiments can be modified in arrangement and detail without departing

from such principles. It should be understood that the programs, processes, or methods described herein are not related or limited to any particular type of computing environment, unless indicated otherwise. Various types of general purpose or specialized computing environments may be used with or perform operations in accordance with the teachings described herein. Elements of embodiments shown in software may be implemented in hardware and vice versa.

[0137] In view of the many possible embodiments to which the principles of our invention may be applied, we claim as our invention all such embodiments as may come within the scope and spirit of the following claims and equivalents thereto.

We claim:

1. An indexed file layout for segmented streaming of multimedia content, the layout comprising:

one or more files containing index information for a program, the index information comprising:
program description information describing the program, wherein the program description information comprises a duration of the program and track information for each audio and video track in the program;
and

streaming segment index information describing each streaming segment of the program; and

one or more files containing streaming segments of the program, wherein each streaming segment of the program is encoded at one or more bitrates, corresponding to one or more available streaming segment encodings, independently of other streaming segments of the program, and wherein the layout supports client switching between different bitrates when transitioning between different streaming segments.

2. The layout of claim 1 wherein the program comprises one or more audio tracks, one or more video tracks, or one or more audio tracks and one or more video tracks, and wherein at least one streaming segment of the program is encoded at a different number of bitrates than at least one other streaming segment of the program.

3. The layout of claim 1 wherein the streaming segment index information comprises, for each streaming segment:
a start position of the streaming segment; and
an indication of bitrates for each available streaming segment encoding of the streaming segment.

4. The layout of claim 3 wherein the streaming segment index information further comprises, for each available streaming segment encoding for each streaming segment:
a size measure of the available streaming segment encoding.

5. The layout of claim 3 wherein the streaming segment index information further comprises, for each available streaming segment encoding for each streaming segment:
a quality measure of the available streaming segment encoding; and
a decoding complexity measure of the available streaming segment encoding.

6. The layout of claim 1 wherein the index information further comprises:

file-level index information, wherein the file-level index information describes file segments of the program;

wherein the streaming segments of the program are organized by file segment, and wherein the layout comprises a separate streaming segment index information file for each file segment.

7. The layout of claim 1 wherein a client selects a streaming segment encoding of a streaming segment from available streaming segment encodings of the streaming segment using a client side rate control algorithm that uses, at least in part, the index information, wherein the client side rate control algorithm calculates a best path through states, wherein each state corresponds to a measure of client buffer fullness for a specific streaming segment encoding of a specific streaming segment.

8. The layout of claim 1 wherein a client selects a streaming segment encoding of a streaming segment from available streaming segment encodings of the streaming segment based only on the index information, available network bandwidth, desired startup delay, and client buffer information.

9. The layout of claim 1 wherein the program is segmented into the one or more streaming segments, wherein each of the one or more streaming segments covers a contiguous time range of corresponding tracks of the program, and wherein all of the one or more streaming segments of the program combine to exactly cover the whole program.

10. A method for optimizing client-side rate control of streaming content, the method comprising:

for each of a plurality of states:

selecting one or more available paths that end in the state, where the one or more available paths are those paths ending in the state that satisfy a constraint condition;

selecting a best path to the state from the one or more available paths, wherein the best path has the lowest distortion amongst the one or more available paths; and

eliminating, as available paths, all of the one or more available paths other than the best path; and

downloading, by a client from a server, one or more streaming segments encoded at bitrates corresponding to the best path to an ending state through the plurality of states, wherein each of the plurality of states corresponds to a specific bitrate encoding of a specific streaming segment.

11. The method of claim 10 wherein the constraint condition is satisfied when a sum of time it takes to download all streaming segments up to a current position is less than or equal to initial startup delay plus cumulative playback time up to the current position.

12. The method of claim 10 wherein the constraint condition is defined by the following equation:

$$T_0 + \sum_{i=0}^{n-1} t[i] \geq \sum_{i=0}^n s[u[i], i],$$

where T_0 is the initial delay, " $t[i]$ " is the playback duration of the i 'th position, " $u[i]$ " is the index of the chosen bitrate for the i 'th position, " $s[u[i], i]$ " is the download time for downloading the $u[i]$ bitrate for the i 'th position, and " n " is the current position.

13. The method of claim 10 further comprising:

receiving, by a client from a server, index information for a program, wherein the index information is used by the client to identify the plurality of states, select the one or more available paths, and select the best path.

- 14.** The method of claim **10** further comprising:
displaying, by the client, the decoded streaming segments
corresponding to the best path.
- 15.** A method for optimizing client-side rate control of
streaming content, the method comprising:
for each of a plurality of states:
selecting one or more available paths that end in the
state, where the one or more available paths are those
paths ending in the state that satisfy a constraint con-
dition; and
retaining the one or more available paths as possible
paths to be included in a best end-to-end path;
performing an end-to-end analysis of the possible paths to
determine the best end-to-end path, wherein the best
end-to-end path has the lowest overall distortion; and
downloading, by a client from a server, one or more stream-
ing segments encoded at bitrates corresponding to the
best end-to-end path through the plurality of states,
wherein each of the plurality of states corresponds to a
measure of client buffer fullness for a specific bitrate
encoding of a specific streaming segment.
- 16.** The method of claim **15** wherein the constraint condi-
tion is satisfied when a sum of time it takes to download all
streaming segments up to a current position is less than or
equal to initial startup delay plus cumulative playback time up
to the current position.

- 17.** The method of claim **15** wherein the constraint condi-
tion is defined by the following equation:

$$T_0 + \sum_{i=0}^{n-1} t[i] \geq \sum_{i=0}^n s[u[i], i],$$

where T_0 is the initial delay, “ $t[i]$ ” is the playback duration
of the i ’th position, “ $u[i]$ ” is the index of the chosen
bitrate for the i ’th position, “ $s[u[i], i]$ ” is the download
time for downloading the $u[i]$ bitrate for the i ’th position,
and “ n ” is the current position.

- 18.** The method of claim **15** further comprising:
receiving, by a client from a server, index information for a
program, wherein the index information is used by the
client to identify the plurality of states and select the one
or more available paths.
- 19.** The method of claim **15** further comprising:
displaying, by the client, the decoded streaming segments
corresponding to the best end-to-end path.
- 20.** The method of claim **15** wherein the plurality of states
are organized into a trellis.

* * * * *