# REAL-TIME OR NEAR REAL-TIME STREAMING

## RELATED APPLICATIONS

**[0001]** This application claims the benefit of the filing dates of the following U.S. provisional applications:

(1) Application No. 61/142,110 filed on December 31, 2008 (Docket No. P7437Z);

(2) Application No. 61/160,693 filed on March 16, 2009 (Docket No. P7437Z2);

(3) Application No. 61/161,036 filed on March 17, 2009 (Docket No. P7437Z3);

(4) Application No. 61/167,524 filed on April 7, 2009 (Docket No. P7437Z4);

(5) Application No. 61/240,648 filed on September 8, 2009 (Docket No. P7437Z5); and

(6) Application No. 61/288,828 filed on December 21, 2009 (Docket No. P7437Z6). All of these U.S. provisional applications are incorporated herein by reference to the extent that they are consistent with this disclosure.

**[0002]** The present U.S. Patent application is related to the following U.S. Patent applications, each of which is incorporated herein by reference:

(1) Application No. 12/479,690 (Docket No. P7437US1), filed June 5, 2009, entitled "REAL-TIME OR NEAR REAL-TIME STREAMING;"

(2) Application No. 12/479,698 (Docket No. P7437US2), filed June 5, 2009, entitled "VARIANT STREAMS FOR REAL-TIME OR NEAR REAL-TIME STREAMING;"

(3) Application No. 12/479,732 (Docket No. P7437US3), filed June 5, 2009, entitled "UPDATABLE REAL-TIME OR NEAR REAL-TIME STREAMING;" and

(4) Application No. 12/479,735 (Docket No. P7437US4), filed June 5, 2009, entitled "PLAYLISTS FOR REAL-TIME OR NEAR REAL-TIME STREAMING."

TECHNICAL FIELD

**[0003]** Embodiments of the invention relate to data transmission techniques. More particularly, embodiments of the invention relate to techniques that allow streaming of data using non-streaming protocols such as, for example, HyperText Transfer Protocol (HTTP).

BACKGROUND

**[0004]** Streaming of content generally refers to multimedia content that is constantly transmitted from a server device and received by a client device. The content is usually presented to an end-user while it is being delivered by the streaming server. The name refers to the delivery method of the medium rather than to the medium itself.

**[0005]** Current streaming services generally require specialized servers to distribute "live" content to end users. In any large scale deployment, this can lead to great cost, and requires specialized skills to set up and run. This results in a less than desirable library of content available for streaming.

SUMMARY OF THE DESCRIPTION

**[0006]** In one embodiment, a server device stores at least a portion of content to be streamed. The content is typically a time based stream of images or audio (e.g. sounds or music) or both; an example of a time based stream is a movie in which the order and presentation of images is based on time, and hence it can be considered a time based stream. The server includes a segmenter agent to decompose the content to be streamed into segments to be transmitted via packets according to a network protocol and an indexer agent to generate one or more playlist files that can facilitate a client in presenting the segmented user data. A client device is coupled with the server device (or another server which stores the segments and playlists and transmits them but does not generate them) via a network. The client device has an assembler agent to receive the one or more playlist files and facilitate retrieval of the segmented media files into the content according to the one or more playlist files. The client device can also

have an output generator agent to output the content via one or more output components of the client device.

**[0007]** In one embodiment, the server device acquires data to be transmitted to the client device. The server device divides the data to be transmitted into multiple media files with a segmenter agent. The server device also stores the multiple segments as individual media files in a memory. The server device further generates one or more playlist files having references to the multiple media files. In response to requests for the data from the client device, the server device (or another server device) transmits the one or more playlist files and at least a subset of the multiple media files over a network to the client device. The multiple media files can be transmitted using a non-streaming transfer protocol in response to requests from the client device; this protocol may be, for example, HTTP.

**[0008]** In one embodiment, the client device can receive and store the one or more playlist files. The client then can request the segmented media files identified in the playlist file(s) and download the linked media files. The client device (or another client device) can then generate an audio and/or video output representing the stream of content.

**[0009]** In one embodiment, an updated playlist can be dynamically generated by a server and then retrieved by a client. The updated playlist can include ancillary material (e.g. advertisements in a sidebar user interface, related content, alternative versions, etc.) shown in addition to the program in the original playlist or can include additional portions of the program (e.g. the second half of a program which is beyond the first half identified in the original playlist). In one implementation, a server can use a rolling method, described herein, to update the playlist which is then retrieved by the client as an updated playlist.

**[0010]** In one embodiment, a playlist can specify a plurality of alternative streams representing the same content; these alternative streams may be the same program transmitted at different visual resolutions (and hence transmitted at different bit rates) or with other different attributes. A server can generate multiple playlists, each for one of the alternative streams and can generate a

variant playlist which refers to or otherwise specifies the alternative streams. The server (or another server) can then transmit the variant playlist to a client device, and the client device can decide, based on current network conditions (e.g. the current throughput rate on a network used to transfer the media files), which playlist to select from the variant playlist, and the client device can download the selected playlist (and further download the media files specified by that selected playlist).

**[0011]** In one embodiment, a client device can switch from a first playlist in the variant playlist to a second playlist in that variant playlist while receiving and presenting content. For example, a client device can be receiving a program, using the first playlist, and a first bit rate and can determine through measurements of the throughput rate of the network that it can receive content of the same program at a higher, second bit rate, that content being specified by the second playlist. In this case, the client device can request the second playlist, receive the second playlist and begin retrieving the media files specified in the second playlist while continuing to present the content specified by the first playlist. The client device can store the media files and the resulting decompressed content in buffers for both playlists, and the client device can perform an automatic operation to determine when and how to switch or transition between the two versions of the content. For example, a client device can use pattern matching of the audio content in the two versions of the content to find a matching point in the two versions and then cause a switch after identifying a transition in the new content from the second playlist.

**[0012]** In one embodiment, a method includes providing multiple redundant locations that provide media content to client devices using alternative streams. To implement failover protection, a first server device or first content distribution service creates a stream, or multiple alternate bandwidth streams and generates playlist file(s). A second server device or second content distribution service creates a parallel stream, or set of streams. A client attempts to download the playlist file(s) from a first uniform resource locator (URL) using a first stream associated with the first server device or the first content distribution

service. If a client is unable to download the playlist file(s) from the first URL, the client attempts to switch to an alternate stream associated with another URL.

**[0013]** In one embodiment, a method includes providing a request for a playlist and specifying in the request that the playlist can be compressed, and receiving, in response to the request, the playlist in a compressed form if a server can provide the playlist in the compressed form (otherwise, the playlist can be provided by the server in uncompressed format). The playlist can be compressed (also referred to as encoded) using a compression technique or format supported by an existing HTTP standard compression technique such as deflate or gzip. The sending and receiving of the playlist in a compressed format can significantly reduce the size of the data being transmitted and received, especially when the playlist grows over time (e.g. the playlist is for a long baseball game). In one embodiment, the use of compression for a playlist can be optional for both the client (the system requesting the playlist) and the server (the system responding to the request by sending the playlist). The use of a compression technique or format that is part of the HTTP standard allows any compliant web server to be able to supply a compressed playlist and any compliant client will also be able to decompress and use the playlist.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0014]** The invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings in which like reference numerals refer to similar elements.

**[0015]** **Figure 1** is a block diagram of one embodiment of a server and clients that can send and receive real-time, or near real-time, content.

**[0016]** **Figure 2A** is a flow diagram of one embodiment of a technique for one or more server devices to support media content using non-streaming protocols.

**[0017]** **Figure 2B** is a flow diagram of one embodiment of a technique for one or more server devices to provide dynamically updated playlists to one or more client devices.

**[0018] Figure 2C** is a flow diagram of one embodiment of a technique for one or more server devices to provide media content to client devices using multiple bit rates.

**[0019] Figure 3A** is a flow diagram of one embodiment of a technique for a client device to support streaming of content using non-streaming protocols.

**[0020] Figure 3B** is a flow diagram of one embodiment of a technique for a client device to support streaming of content using multiple bit rates.

**[0021] Figure 4** is a block diagram of one embodiment of a server stream agent.

**[0022] Figure 5** is a block diagram of one embodiment of a client stream agent.

**[0023] Figure 6** illustrates on embodiment, of a playlist file with multiple tags.

**[0024] Figure 7** is a flow diagram of one embodiment of a playback technique for assembled streams as described herein.

**[0025] Figure 8** is a block diagram of one embodiment of an electronic system.

**[0026] Figure 9A** is a flowchart showing an example of how a client device can switch between alternative content in a variant playlist.

**[0027] Figure 9B** is a further flowchart showing how a client device can switch between content in two playlists.

**[0028] Figure 9C** is a further flowchart showing an example of how a client device can switch between content using audio pattern matching.

**[0029] Figure 9D** shows diagrammatically how the method of Figure 9C is implemented with audio pattern matching.

**[0030] Figure 10** is a flow diagram of one embodiment of a technique for providing multiple redundant locations that provide media content to client devices using alternative streams.

**[0031] Figure 11** illustrates a network in which a client 1102 communicates bi-directionally with one or more URLs in accordance with one embodiment.

## DETAILED DESCRIPTION

**[0032]** In the following description, numerous specific details are set forth. However, embodiments of the invention may be practiced without these specific details. In other instances, well-known circuits, structures and techniques have

not been shown in detail in order not to obscure the understanding of this description.

**[0033]** The present description includes material protected by copyrights, such as illustrations of graphical user interface images. The owners of the copyrights, including the assignee of the present invention, hereby reserve their rights, including copyright, in these materials. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office file or records, but otherwise reserves all copyrights whatsoever. Copyright Apple Inc. 2009.

**[0034]** In one embodiment, techniques and components described herein can include mechanisms to deliver streaming experience using non-streaming protocols (e.g., HTTP) and other technologies (e.g., Motion Picture Expert Group (MPEG) streams). For example, near real-time streaming experience can be provided using HTTP to broadcast a "live" musical or sporting event, live news, a Web camera feed, etc. In one embodiment, a protocol can segment incoming media data into multiple media files and store those segmented media files on a server. The protocol can also build a playlist file that includes Uniform Resource Identifiers (URIs) that direct the client to the segmented media files stored on a server. When the segmented media files are played back in accordance with the playlist file(s), the client can provide the user with a near real-time broadcast of a "live" event. Pre-recorded content can be provided in a similar manner.

**[0035]** In one embodiment, the server can dynamically introduce supplementary or alternative media content (e.g., advertisements, statistics related to a sporting event, additional media content to the main presentation) into the broadcast event. For example, during client playback of a media event, the server can add additional URIs to the playlist file, the URIs may identify a location from which a client can download a supplementary media file. The client can be instructed to periodically retrieve from the server one or more updated playlist file(s) in order to access any supplementary or additional (or both) media content the server has introduced.

**[0036]** In one embodiment, the server can operate in either cumulative mode or in rolling mode. In cumulative mode, the server can create a playlist file and append media file identifiers to the end of the playlist file. The client then has access to all parts of the stream from a single playlist file (e.g., a user can start at the middle of a show) when downloaded. In rolling mode, the server may limit the availability of media files by removing media file identifiers from the beginning of the playlist file on a rolling basis, thereby providing a sliding window of media content accessible to a client device. The server can also add media file identifiers to the playlist and, in rolling mode, the server can limit the availability of media files to those that have been most recently added to the playlist. The client then repeatedly downloads updated copies of the playlist file to continue viewing. The rolling basis for playlist downloading can be useful when the content is potentially unbounded in time (e.g. content from a continuously operated web cam). The client can continue to repeatedly request the playlist in the rolling mode until it finds an end tag in the playlist.

**[0037]** In one embodiment, the mechanism supports bit rate switching by providing variant streams of the same presentation. For example, several versions of a presentation to be served can be stored on the server. Each version can have substantially the same content but be encoded at different bit rates. This can allow the client device to switch between bit rates depending on, for example, a detection of the available bandwidth, without compromising continuity of playback.

**[0038]** In one embodiment, protection features may be provided to protect content against unauthorized use. For example, non-sequential media file numbering may be used to prevent prediction. Encryption of media files may be used. Partial media file lists may be used. Additional and/or different protection features may also be provided.

**[0039]** **Figure 1** is a block diagram of one embodiment of a server and clients that can send and receive real-time, or near real-time, content. The example of Figure 1 provides a simple server-client connection with two clients coupled with a server via a network. Any number of clients may be supported utilizing the

techniques and mechanisms described herein. Further, multiple servers may provide content and/or may operate together to provide content according to the techniques and mechanisms described herein. For example, one server may create the content, create the playlists and create the multiple media (e.g. files) and other servers store and transmit the created content.

**[0040]** Network 110 may be any type of network whether wired, wireless (e.g., IEEE 802.11, 802.16) or any combination thereof. For example, Network 100 may be the Internet or an intranet. As another example, network 110 may be a cellular network (e.g., 3G, CDMA). In one embodiment, client devices 150 and 180 may be capable of communicating over multiple network types (e.g. each device can communicate over a WiFi wireless LAN and also over a wireless cellular telephone network). For example, client devices 150 and 180 may be smart phones or cellular-enabled personal digital assistants that can communicate over cellular radiotelephone networks as well as data networks. These devices may be able to utilize the streaming mechanisms described herein over either type of network or even switch between networks as necessary.

**[0041]** Server 120 may operate as a HTTP server in any manner known in the art. That is server 120 includes a HTTP server agent 145 that provides content using HTTP protocols. While the example of Figure 1 is described in terms of HTTP, other protocols can be utilized in a similar manner. Segmenter 130 and indexer 135 are agents that reside on server 120 (or multiple servers) to provide content in media files with a playlist file as described herein. These media files and playlist files may be provided over network 110 via HTTP server agent 145 (or via other servers) using HTTP protocols. Agents as discussed herein can be implemented as hardware, software, firmware or a combination thereof.

**[0042]** Segmenter 130 may function to divide the stream of media data into multiple media files that may be transmitted via HTTP protocols. Indexer 135 may function to create a playlist file corresponding to the segmented media files so that client devices can reassemble the media files to provide real-time, or near real-time, transmission of the content provided by server 120. In response to one or more requests from a client device, HTTP server agent 145 (or other servers)

may transmit one or more playlist files as generated by indexer 135 and media files of content as generated by segmenter 130. Server 120 may further include optional security agent 140 that provides one or more of the security functions (e.g. encryption) discussed herein. Server 120 may also include additional components not illustrated in Figure 1.

**[0043]** Client devices 150 and 180 may receive the playlist files and media files from server 120 over network 110. Client devices may be any type of electronic device that is capable of receiving data transmitted over a network and generate output utilizing the data received via the network, for example, wireless mobile devices, PDAs, entertainment devices, consumer electronic devices, etc. The output may be any media type of combination of media types, including, for example, audio, video or any combination thereof.

**[0044]** Client device 150 can include assembler agent 160 and output generator agent 165. Similarly, client device 180 can include assembler agent 190 and output generator agent 195. Assembler agents 160 and 180 receive the playlist files from server 120 and use the playlist files to access and download media files from server 120. Output generator agents 165 and 195 use the downloaded media files to generate output from client devices 150 and 160, respectively. The output may be provided by one or more speakers, one or more display screens, a combination of speakers and display screens or any other input or output device. The client devices can also include memory (e.g. flash memory or DRAM, etc.) to act as a buffer to store the media files (e.g. compressed media files or decompressed media files) as they are received; the buffer can provide many seconds worth of presentable content beyond the time of content currently being presented so that the buffered content can later be displayed while new content is being downloaded. This buffer can provide presentable content while the client device is attempting to retrieve content through an intermittently slow network connection and hence the buffer can hide network latency or connection problems.

**[0045]** Client devices 150 and 180 may further include optional security agents 170 and 185, respectively that provide one or more of the security functions

discussed herein. Client devices 150 and 180 may also include additional components not illustrated in Figure 1.

**[0046]** In one embodiment, the techniques that are described in this application may be used to transmit an unbounded stream of multimedia data over a non-streaming protocol (e.g., HTTP). Embodiments can also include encryption of media data and/or provision of alternate versions of a stream (e.g., to provide alternate bit rates). Because media data can be transmitted soon after creation, the data can be received in near real-time. Example data formats for files as well as actions to be taken by a server (sender) and a client (receiver) of the stream of multimedia data are provided; however, other formats can also be supported.

**[0047]** A media presentation that can be transmitted as a simulated real-time stream (or near real-time stream) is specified by a Universal Resource Indicator (URI) that indicates a playlist file. In one embodiment, the playlist file is an ordered list of additional URIs. Each URI in the playlist file refers to a media file that is a segment of a stream, which may be a single contiguous stream of media data for a particular program.

**[0048]** In order to play the stream of media data, the client device obtains the playlist file from the server. The client also obtains and plays each media data file indicated by the playlist file. In one embodiment, the client can dynamically or repeatedly reload the playlist file to discover additional and/or different media segments.

**[0049]** The playlist files may be, for example, Extended M3U Playlist files. In one embodiment, additional tags that effectively extend the M3U format are used. M3U refers to Moving Picture Experts Group Audio Layer 3 Uniform Resource Locator (MP3 URL) and is a format used to store multimedia playlists. A M3U file is a text file that contains the locations of one or more media files for a media player to play.

**[0050]** The playlist file, in one embodiment, is an Extended M3U-formatted text file that consists of individual lines. The lines can be terminated by either a single LF character or a CR character followed by a LF character. Each line can be a URI, a blank line, or start with a comment character (e.g. '#'). URIs identify

media files to be played. Blank lines can be ignored.

**[0051]** Lines that start with the comment character can be either comments or tags. Tags can begin with #EXT, while comment lines can begin with #. Comment lines are normally ignored by the server and client. In one embodiment, playlist files are encoded in UTF-8 format. UTF-8 (8-bit Unicode Transformation Format) is a variable-length character encoding format. In alternate embodiments, other character encoding formats can be used.

**[0052]** In the examples that follow, an Extended M3U format is utilized that includes two tags: EXTM3U and EXTINF. An Extended M3U file may be distinguished from a basic M3U file by a first line that includes "#EXTM3U".

**[0053]** EXTINF is a record marker that describes the media file identified by the URI that follows the tag. In one embodiment, each media file URI is preceded by an EXTINF tag, for example:

#EXTINF: <duration>,<title>

where "duration" specifies the duration of the media file and "title" is the title of the target media file.

**[0054]** In one embodiment, the following tags may be used to manage the transfer and playback of media files:

EXT-X-TARGETDURATION
EXT-X-MEDIA-SEQUENCE
EXT-X-KEY
EXT-X-PROGRAM-DATE-TIME
EXT-X-ALLOW-CACHE
EXT-X-STREAM-INF
EXT-X-ENDLIST

These tags will each be described in greater detail below. While specific formats and attributes are described with respect to each new tag, alternative embodiments can also be supported with different attributes, names, formats, etc.

**[0055]** The EXT-X-TARGETDURATION tag can indicate the approximate duration of the next media file that will be added to the presentation. It can be included in the playback file and the format can be:

#EXT-X-TARGETDURATION:<seconds>

where "seconds" indicates the duration of the media file. In one embodiment, the actual duration may differ slightly from the target duration indicated by the tag. In one embodiment, every URI indicating a segment will be associated with an approximate duration of the segment; for example, the URI for a segment may be prefixed with a tag indicating the approximate duration of that segment.

**[0056]** Each media file URI in a playlist file can have a unique sequence number. The sequence number, if present, of a URI is equal to the sequence number of the URI that preceded it, plus one in one embodiment. The EXT-X-MEDIA-SEQUENCE tag can indicate the sequence number of the first URI that appears in a playlist file and the format can be:

#EXT-X-MEDIA-SEQUENCE:<number>

where "number" is the sequence number of the URI. If the playlist file does not include a #EXT-X-MEDIA-SEQUENCE tag, the sequence number of the first URI in the playlist can be considered 1. In one embodiment, the sequence numbering can be non-sequential; for example, non-sequential sequence numbering such as 1, 5, 7, 17, etc. can make it difficult to predict the next number in a sequence and this can help to protect the content from pirating. Another option to help protect the content is to reveal only parts of a playlist at any given time.

**[0057]** Some media files may be encrypted. The EXT-X-KEY tag provides information that can be used to decrypt media files that follow it and the format can be:

#EXT-X-KEY:METHOD=<method>[,URI="<URI>"]

The METHOD parameter specifies the encryption method and the URI parameter, if present, specifies how to obtain the key.

**[0058]** An encryption method of NONE indicates no encryption. Various encryption methods may be used, for example AES-128, which indicates encryption using the Advance Encryption Standard encryption with a 128-bit key and PKCS7 padding [see RFC3852]. A new EXT-X-KEY tag supersedes any prior EXT-X-KEY tags.

**[0059]** An EXT-X-KEY tag with a URI parameter identifies the key file. A key file may contain the cipher key that is to be used to decrypt subsequent media files listed in the playlist file. For example, the AES-128 encryption method uses 16-octet keys. The format of the key file can be a packed array of 16 octets in binary format.

**[0060]** Use of AES-128 normally requires that the same 16-octet initialization vector (IV) be supplied when encrypting and decrypting. Varying the IV can be used to increase the strength of the cipher. When using AES-128 encryption, the sequence number of the media file can be used as the IV when encrypting or decrypting media files.

**[0061]** The EXT-X-PROGRAM-DATE-TIME tag can associate the beginning of the next media file with an absolute date and/or time and can include or indicate a time zone. In one embodiment, the date/time representation is ISO/IEC 8601:2004. The tag format can be:

EXT-X-PROGRAM-DATE-TIME:<YYYY-MM-DDThh:mm:ssZ>

**[0062]** The EXT-X-ALLOW-CACHE tag can be used to indicate whether the client may cache the downloaded media files for later playback. The tag format can be:

EXT-X-ALLOW-CACHE:<YES|NO>

**[0063]** The EXT-X-ENDLIST tag indicates in one embodiment that no more media files will be added to the playlist file. The tag format can be:

EXT-X-ENDLIST

In one embodiment, if a playlist contains the final segment or media file then the playlist will have the EXT-X-ENDLIST tag.

**[0064]** The EXT-X-STREAM-INF tag can be used to indicate that the next URI in the playlist file identifies another playlist file. The tag format can be, in one embodiment:

EXT-X-STREAM-INF:[attribute=value][,attribute=value]*<URI>

where the following attributes may be used. The attribute BANDWIDTH=<n> is an approximate upper bound of the stream bit rate expressed as a number of bits per second. The attribute PROGRAM-ID=<i> is a number that uniquely

identifies a particular presentation within the scope of the playlist file. A playlist file may include multiple EXT-X-STREAM-INF URIs with the same PROGRAM-ID to describe variant streams of the same presentation. Variant streams and variant playlists are described further in this disclosure (e.g. see Figures 9A-9D).

**[0065]** The foregoing tags and attributes can be used by the server device to organize, transmit and process the media files that represent the original media content. The client devices use this information to reassemble and present the media files in a manner to provide a real-time, or near real-time, streaming experience (e.g. viewing of a live broadcast such as a music or sporting event) to a user of the client device.

**[0066]** Each media file URI in a playlist file identifies a media file that is a segment of the original presentation (i.e., original media content). In one embodiment, each media file is formatted as a MPEG-2 transport stream, a MPEG-2 program stream, or a MPEG-2 audio elementary stream. The format can be specified by specifying a CODEC, and the playlist can specify a format by specifying a CODEC. In one embodiment, all media files in a presentation have the same format; however, multiple formats may be supported in other embodiments. A transport stream file should, in one embodiment, contain a single MPEG-2 program, and there should be a Program Association Table and a Program Map Table at the start of each file. A file that contains video SHOULD have at least one key frame and enough information to completely initialize a video decoder. Clients SHOULD be prepared to handle multiple tracks of a particular type (e.g. audio or video) by choosing a reasonable subset. Clients should, in one embodiment, ignore private streams inside Transport Streams that they do not recognize. The encoding parameters for samples within a stream inside a media file and between corresponding streams across multiple media files SHOULD remain consistent. However clients SHOULD deal with encoding changes as they are encountered, for example by scaling video content to accommodate a resolution change.

**[0067]** **Figure 2A** is a flow diagram of one embodiment of a technique for one or more server devices to support media content using non-streaming protocols. The example of Figure 2A is provided in terms of HTTP; however, other non-streaming protocols can be utilized in a similar manner. The example of Figure 2A is provided in terms of a single server performing certain tasks. However, any number of servers may be utilized. For example, the server that provides media files to client devices may be a different device than a server that segments the content into multiple media files.

**[0068]** The server device receives content to be provided in operation 200. The content may represent live audio and/or video (e.g., a sporting event, live news, a Web camera feed). The content may also represent pre-recorded content (e.g., a concert that has been recorded, a training seminar, etc.). The content may be received by the server according to any format and protocol known in the art, whether streamed or not. In one embodiment, the content is received by the server in the form of a MPEG-2 stream; however, other formats can also be supported.

**[0069]** The server may then store temporarily at least portions of the content in operation 210. The content or at least portions of the content may be stored temporarily, for example, on a storage device (e.g., hard disk in a Storage Area Network, etc.) or in memory. Alternatively, the content may be received as via a storage medium (e.g., compact disc, flash drive) from which the content may be transferred to a storage device or memory. In one embodiment, the server has an encoder that converts, if necessary, the content to one or more streams (e.g., MPEG-2). This conversion can occur without storing permanently the received content, and in some embodiments, the storage operation 210 may be omitted or it may be a longer term storage (e.g. an archival storage) in other embodiments.

**[0070]** The content to be provided is segmented into multiple media files in operation 220. In one embodiment, the server converts a stream into separate and distinct media files (i.e., segments) that can be distributed using a standard web server. In one embodiment, the server segments the media stream at points that support effective decode of the individual media files (e.g., on packet and key

frame boundaries such as PES packet boundaries and i-frame boundaries). The media files can be portions of the original stream with approximately equal duration. The server also creates a URI for each media file. These URIs allow client devices to access the media files.

**[0071]** Because the segments are served using HTTP servers, which inherently deliver whole files, the server should have a complete segmented media file available before it can be served to the clients. Thus, the client may lag (in time) the broadcast by at least one media file length. In one embodiment, media file size is based on a balance between lag time and having too many files.

**[0072]** In one embodiment, two session types (live session and event session) are supported. For a live session, only a fixed size portion of the stream is preserved. In one embodiment, content media files that are out of date are removed from the program playlist file, and can be removed from the server. The second type of session is an event session, where the client can tune into any point of the broadcast (e.g., start from the beginning, start from a mid-point). This type of session can be used for rebroadcast, for example.

**[0073]** The media files are stored in the server memory in operation 230. The media files can be protected by a security feature, such as encryption, before storing the files in operation 230. The media files are stored as files that are ready to transmit using the network protocol (e.g., HTTP or HTTPS) supported by the Web server application on the server device (or supported by another device which does the transmission).

**[0074]** One or more playlist files are generated to indicate the order in which the media files should be assembled to recreate the original content in operation 240. The playlist file(s) can utilize Extended M3U tags and the tags described herein to provide information for a client device to access and reassemble the media files to provide a streaming experience on the client device. A URI for each media file is included in the playlist file(s) in the order in which the media files are to be played. The server can also create one or more URIs for the playlist file(s) to allow the client devices to access the playlist file(s).

**[0075]** The playlist file(s) can be stored on the server in operation 250. While the

creation and storing of media files and playlist file(s) are presented in a particular order in Figure 2A, a different order may also be used. For example, the playlist file(s) may be created before the media files are created or stored. As another example, the playlist file(s) and media files may be created before either are stored.

**[0076]** If media files are to be encrypted the playlist file(s) can define a URI that allows authorized client devices to obtain a key file containing an encryption key to decrypt the media files. An encryption key can be transmitted using a secure connection (e.g., HTTPS). As another example, the playlist file(s) may be transmitted using HTTPS. As a further example, media files may be arranged in an unpredictable order so that the client cannot recreate the stream without the playlist file(s).

**[0077]** If the encryption method is AES-128, AES-128 CBC encryption, for example, may be applied to individual media files. In one embodiment, the entire file is encrypted. Cipher block chaining is normally not applied across media files in one embodiment. The sequence of the media files is use as the IV as described above. In one embodiment, the server adds an EXT-X-KEY tag with the key URI to the end of the playlist file. The server then encrypts all subsequent media files with that key until a change in encryption configuration is made.

**[0078]** To switch to a new encryption key, the server can make the new key available via a new URI that is distinct from all previous key URIs used in the presentation. The server also adds an EXT-X-KEY tag with the new key URI to the end of a playlist file and encrypts all subsequent media files with the new key.

**[0079]** To end encryption, the server can add an EXT-X-KEY tag with the encryption method NONE at the end of the playlist file. The tag (with "NONE" as the method) does not include a URI parameter in one embodiment. All subsequent media files are not encrypted until a change in encryption configuration is made as described above. The server does not remove an EXT-X-KEY tag from a playlist file if the playlist file contains a URI to a media file encrypted with that key. The server can transmit the playlist file(s) and the media files over the network in response to client requests in operation 270, as described

in more detail with respect to Figure 3A.

**[0080]** In one embodiment, a server transmits the playlist file to a client device in response to receiving a request from a client device for a playlist file. The client device may access/request the playlist file using a URI that has been provided to the client device. The URI indicates the location of the playlist file on the server. In response, the server may provide the playlist file to the client device. The client device may the utilize tags and URIs (or other identifiers) in the playlist file to access the multiple media files.

**[0081]** In one embodiment, the server may limit the availability of media files to those that have been most recently added to the playlist file(s). To do this, each playlist file can include only one EXT-X-MEDIA-SEQUENCE tag and the value can be incremented by one for every media file URI that is removed from the playlist file. Media file URIs can be removed from the playlist file(s) in the order in which they were added. In one embodiment, when the server removes a media file URI from the playlist file(s) the media file remains available to clients for a period of time equal to the duration of the media file plus the duration of the longest playlist file in which the media file has appeared.

**[0082]** The duration of a playlist file is the sum of the durations of the media files within that playlist file. Other durations can also be used. In one embodiment, the server can maintain at least three main presentation media files in the playlist at all times unless the EXT-X-ENDLIST tag is present.

**[0083]** **Figure 2B** is a flow diagram of one embodiment of a technique for one or more server devices to provide dynamically updated playlists to one or more client devices. The playlists can be updated using either of the cumulative mode or the rolling mode described herein. The example of Figure 2B is provided in terms of HTTP; however, other non-streaming protocols (e.g. HTTPS, etc.) can be utilized in a similar manner. The example of Figure 2B is provided in terms of a server performing certain tasks. However, any number of servers may be utilized. For example, the server that provides media files to client devices may be a different device than the server that segments the content into multiple media files.

**[0084]** The server device receives content to be provided in operation 205. The server may then temporarily store at least portions of the content in operation 215. Operation 215 can be similar to operation 210 in Figure 2A. The content to be provided is segmented into multiple media files in operation 225. The media files can be stored in the server memory in operation 235. The media files can be protected by a security feature, such as encryption, before storing the files in operation 235.

**[0085]** One or more playlist files are generated to indicate the order in which the media files should be assembled to recreate the original content in operation 245. The playlist file(s) can be stored on the server in operation 255. While the creation and storing of media files and playlist file(s) are presented in a particular order in Figure 2B, a different order may also be used.

**[0086]** The server (or another server) can transmit the playlist file(s) and the media files over the network in response to client requests in operation 275, as described in more detail with respect to Figures 3A-3B.

**[0087]** The playlist file(s) may be updated by a server for various reasons. The server may receive additional data to be provided to the client devices in operation 285. The additional data can be received after the playlist file(s) are stored in operation 255. The additional data may be, for example, additional portions of a live presentation, or additional information for an existing presentation. Additional data may include advertisements or statistics (e.g. scores or data relating to a sporting event). The additional data could be overlaid (through translucency) on the presentation or be presented in a sidebar user interface. The additional data can be segmented in the same manner as the originally received data. If the additional data constitutes advertisements, or other content to be inserted into the program represented by the playlist, the additional data can be stored (at least temporarily) in operation 215, segmented in operation 225 and stored in operation 235; prior to storage of the segmented additional data, the segments of the additional data can be encrypted. Then in operation 245 an updated playlist, containing the program and the additional data, would be generated. The playlist is updated based on the additional data and stored again in

operation 255. Changes to the playlist file(s) should be made atomically from the perspective of the client device. The updated playlist replaces, in one embodiment, the previous playlist. As discussed below in greater detail, client devices can request the playlist multiple times. These requests enable the client devices to utilize the most recent playlist. In one embodiment, the additional data may be metadata; in this case, the playlist does not need to be updated, but the segments can be updated to include metadata. For example, the metadata may contain timestamps which can be matched with timestamps in the segments, and the metadata can be added to segments having matching timestamps.

**[0088]** The updated playlist may also result in the removal of media files. In one embodiment, a server should remove URIs, for the media files, from the playlist in the order in which they were added to the playlist. In one embodiment, if the server removes an entire presentation, it makes the playlist file(s) unavailable to client devices. In one embodiment, the server maintains the media files and the playlist file(s) for the duration of the longest playlist file(s) containing a media file to be removed to allow current client devices to finish accessing the presentation. Accordingly, every media file URI in the playlist file can be prefixed with an EXT-X-STREAM-INF tag to indicate the approximate cumulative duration of the media files indicated by the playlist file. In alternate embodiments, the media files and the playlist file(s) may be removed immediately.

**[0089]** Subsequent requests for the playlist from client devices result in the server providing the updated playlist in operation 275. In one embodiment, playlists are updated on a regular basis, for example, a period of time related to the target duration. Periodic updates of the playlist file allow the server to provide access to servers to a dynamically changing presentation.

**[0090]** **Figure 2C** is a flow diagram of one embodiment of a technique for one or more server devices to provide media content to client devices using multiple bit rates, which is one form of the use of alternative streams. The example of Figure 2C is provided in terms of HTTP; however, other non-streaming protocols can be utilized in a similar manner. The example of Figure 2C is provided in

terms of a server performing certain tasks. However, any number of servers may be utilized. For example, the server that provides media files to client devices may be a different device than a server that segments the content into multiple media files.

**[0091]** In one embodiment, the server can offer multiple playlist files or a single playlist file with multiple media file lists in the single playlist file to provide different encodings of the same presentation. If different encodings are provided, playlist file(s) may include each variant stream providing different bit rates to allow client devices to switch between encodings dynamically (this is described further in connection with Figures 9A-9D). Playlist files having variant streams can include an EXT-X-STREAM-INF tag for each variant stream. Each EXT-X-STREAM-INF tag for the same presentation can have the same PROGRAM-ID attribute value. The PROGRAM-ID value for each presentation is unique within the variant streams.

**[0092]** In one embodiment, the server meets the following constraints when producing variant streams. Each variant stream can consist of the same content including optional content that is not part of the main presentation. The server can make the same period of content available for all variant streams within an accuracy of the smallest target duration of the streams. The media files of the variant streams are, in one embodiment, either MPEG-2 Transport Streams or MPEG-2 Program Streams with sample timestamps that match for corresponding content in all variant streams. Also, all variant streams should, in one embodiment, contain the same audio encoding. This allows client devices to switch between variant streams without losing content.

**[0093]** Referring to Figure 2C, the server device receives content to be provided in operation 202. The server may then at least temporarily store the content in operation 212. The content to be provided is segmented into multiple media files in operation 222. Each media file is encoded for a selected bit rate (or a selected value of other encoding parameters) and stored on the server in operation 232. For example, the media files may be targeted for high-, medium- and low-bandwidth connections. The media files can be encrypted prior to storage. The

encoding of the media files targeted for the various types of connections may be selected to provide a streaming experience at the target bandwidth level.

**[0094]** In one embodiment, a variant playlist is generated in operation 242 with tags as described herein that indicate various encoding levels. The tags may include, for example, an EXT-X-STREAM-INF tag for each encoding level with a URI to a corresponding media playlist file.

**[0095]** This variant playlist can include URIs to media playlist files for the various encoding levels. Thus, a client device can select a target bit rate from the alternatives provided in the variant playlist indicating the encoding levels and retrieve the corresponding playlist file. In one embodiment, a client device may change between bit rates during playback (e.g. as described with respect to Figures 9A-9D). The variant playlist indicating the various encoding levels is stored on the server in operation 252. In operation 242, each of the playlists referred to in the variant playlist can also be generated and then stored in operation 252.

**[0096]** In response to a request from a client device, the server may transmit the variant playlist that indicates the various encoding levels in operation 272. The server may receive a request for one of the media playlists specified in the variant playlist corresponding to a selected bit rate in operation 282. In response to the request, the server transmits the media playlist file corresponding to the request from the client device in operation 292. The client device may then use the media playlist to request media files from the server. The server provides the media files to the client device in response to requests in operation 297.

**[0097]** **Figure 3A** is a flow diagram of one embodiment of a technique for a client device to support streaming of content using non-streaming protocols. The example of Figure 3A is provided in terms of HTTP; however, other non-streaming protocols can be utilized in a similar manner. The methods shown in Figures 3A-3B can be performed by one client device or by several separate client devices. For example, in the case of any one of these methods, a single client device may perform all of the operations (e.g. request a playlist file, request media files using URIs in the playlist file, assemble the media files to

generate and provide a presentation/output) or several distinct client devices can perform some but not all of the operations (e.g. a first client device can request a playlist file and request media files using URIs in the playlist file and can store those media files for use by a second client device which can process the media files to generate and provide a presentation/output).

**[0098]** The client device may request a playlist file from a server in operation 300. In one embodiment, the request is made according to an HTTP-compliant protocol. The request utilizes a URI to an initial playlist file stored on the server. In alternate embodiments, other non-streaming protocols can be supported. In response to the request, the server will transmit the corresponding playlist file to the client over a network. As discussed above, the network can be wired or wireless and can be any combination of wired or wireless networks. Further, the network may be a data network (e.g., IEEE 802.11, IEEE 802.16) or a cellular telephone network (e.g., 3G).

**[0099]** The client device can receive the playlist file in operation 310. The playlist file can be stored in a memory of the client device in operation 320. The memory can be, for example, a hard disk, a flash memory, a random-access memory. In one embodiment, each time a playlist file is loaded or reloaded from the playlist URI, the client checks to determine that the playlist file begins with a #EXTM3U tag and does not continue if the tag is absent. As discussed above, the playlist file includes one or more tags as well as one or more URIs to media files.

**[0100]** The client device can include an assembler agent that uses the playlist file to reassemble the original content by requesting media files indicated by the URIs in the playlist file in operation 330. In one embodiment, the assembler agent is a plug-in module that is part of a standard Web browser application. In another embodiment, the assembler agent may be a stand-alone application that interacts with a Web browser to receive and assemble the media files using the playlist file(s). As a further example, the assembler agent may be a special-purpose hardware or firmware component that is embedded in the client device.

**[0101]** The assembler causes media files from the playlist file to be downloaded from the server indicated by the URIs. If the playlist file contains the EXT-X-ENDLIST tag, any media file indicated by the playlist file may be played first. If the EXT-X-ENDLIST tag is not present, any media file except for the last and second-to-last media files may be played first. Once the first media file to play has been chosen, subsequent media files in the playlist file are loaded, in one embodiment, in the order that they appear in the playlist file (otherwise the content is presented out of order). In one embodiment, the client device attempts to load media files in advance of when they are required (and stores them in a buffer) to provide uninterrupted playback and to compensate for temporary variations in network latency and throughput.

**[0102]** The downloaded media file(s) can be stored in a memory on the client device in operation 340. The memory in which the content can be stored may be any type of memory on the client device, for example, random-access memory, a hard disk, or a video buffer. The storage may be temporary to allow playback or may be permanent. If the playlist file contains the EXT-X-ALLOW-CACHE tag and its value is NO, the client does not store the downloaded media files after they have been played. If the playlist contains the EXT-X-ALLOW-CACHE tag and its value is YES, the client device may store the media files indefinitely for later replay. The client device may use the value of the EXT-X-PROGRAM-DATE-TIME tag to display the program origination time to the user. In one embodiment, the client can buffer multiple media files so that it is less susceptible to network jitter, in order to provide a better user experience.

**[0103]** In one embodiment, if the decryption method is AES-128, then AES-128 CBC decryption is applied to the individual media files. The entire file is decrypted. In one embodiment, cipher block chaining is not applied across media files. The sequence number of the media file can be used as the initialization vector as described above.

**[0104]** From the memory, the content can be output from the client device in operation 350. The output or presentation may be, for example, audio output via built-in speakers or head phones. The output may include video that is output via

a screen or projected from the client device. Any type of output known in the art may be utilized. In operation 351, the client device determines whether there are any more media files in the stored, current playlist which have not been played or otherwise presented. If such media files exist (and if they have not been requested) then processing returns to operation 330 in which one or more media files are requested and the process repeats. If there are no such media files (i.e., all media files in the current playlist have been played), then processing proceeds to operation 352, which determines whether the playlist file includes an end tag.

**[0105]** If the playlist includes an end tag (e.g., EXT-X-ENDLIST) in operation 352, playback ceases when the media files indicated by the playlist file have been played. If the end tag is not in the playlist, then the client device requests a playlist again from the server and reverts back to operation 300 to obtain a further or updated playlist for the program.

**[0106]** As discussed in greater detail with respect to Figure 2B, a server may update a playlist file to introduce supplementary content (e.g., additional media file identifiers corresponding to additional media content in a live broadcast) or additional content (e.g. content further down the stream). To access the supplementary content or additional content, a client can reload the updated playlist from the server. This can provide a mechanism by which playlist files can be dynamically updated, even during playback of the media content associated with a playlist file. A client can request a reload of the playlist file based on a number of triggers. The lack of an end tag is one such trigger.

**[0107]** In one embodiment, the client device periodically reloads the playlist file(s) unless the playlist file contains the EXT-X-ENDLIST tag. When the client device loads a playlist file for the first time or reloads a playlist file and finds that the playlist file has changed since the last time it was loaded, the client can wait for a period of time before attempting to reload the playlist file again. This period is called the initial minimum reload delay. It is measured from the time that the client began loading the playlist file.

**[0108]** In one embodiment, the initial minimum reload delay is the duration of the last media file in the playlist file or three times the target duration, whichever

is less. The media file duration is specified by the EXTINF tag. If the client reloads a playlist file and finds that it has not changed then the client can wait for a period of time before retrying. The minimum delay in one embodiment is three times the target duration or a multiple of the initial minimum reload delay, whichever is less. In one embodiment, this multiple is 0.5 for a first attempt, 1.5 for a second attempt and 3.0 for subsequent attempts; however, other multiples may be used.

**[0109]** Each time a playlist file is loaded or reloaded, the client device examines the playlist file to determine the next media file to load. The first file to load is the media file selected to play first as described above. If the first media file to be played has been loaded and the playlist file does not contain the EXT-X-MEDIA-SEQUENCE tag then the client can verify that the current playlist file contains the URI of the last loaded media file at the offset where it was originally found, halting playback if the file is not found. The next media file to load can be the first media file URI following the last-loaded URI in the playlist file.

**[0110]** If the first file to be played has been loaded and the playlist file contains the EXT-X-MEDIA-SEQUENCE tag, then the next media file to load can be the one with the lowest sequence number that is greater than the sequence number of the last media file loaded. If the playlist file contains an EXT-X-KEY tag that specifies a key file URI, the client device obtains the key file and uses the key inside the key file to decrypt the media files following the EXT-X-KEY tag until another EXT-X-KEY tag is encountered.

**[0111]** In one embodiment, the client device utilizes the same URI as previously used to download the playlist file. Thus, if changes have been made to the playlist file, the client device may use the updated playlist file to retrieve media files and provide output based on the media files.

**[0112]** Changes to the playlist file may include, for example, deletion of a URI to a media file, addition of a URI to a new media file, replacement of a URI to a replacement media file. When changes are made to the playlist file, one or more tags may be updated to reflect the change(s). For example, the duration tag may

be updated if changes to the media files result in a change to the duration of the playback of the media files indicated by the playlist file.

**[0113]** **Figure 3B** is a flow diagram of one embodiment of a technique for a client device to support streaming of content using multiple bit rates which is one form of alternative streams. The example of Figure 3B is provided in terms of HTTP; however, other non-streaming protocols can be utilized in a similar manner.

**[0114]** The client device can request a playlist file in operation 370. As discussed above, the playlist file may be retrieved utilizing a URI provided to the client device. In one embodiment, the playlist file includes listings of variant streams of media files to provide the same content at different bit rates; in other words, a single playlist file includes URIs for the media files of each of the variant streams. The example shown in Figure 3B uses this embodiment. In another embodiment, the variant streams may be represented by multiple distinct playlist files separately provided to the client that each provide the same content at different bit rates, and a variant playlist can provide a URI for each of the distinct playlist files. This allows the client device to select the bit rate based on client conditions.

**[0115]** The playlist file(s) can be retrieved by the client device in operation 375. The playlist file(s) can be stored in the client device memory in operation 380. The client device may select the bit rate to be used in operation 385 based upon current network connection speeds. Media files are requested from the server utilizing URIs included in the playlist file corresponding to the selected bit rate in operation 390. The retrieved media files can be stored in the client device memory. Output is provided by the client device utilizing the media files in operation 394 and the client device determines whether to change the bit rate.

**[0116]** In one embodiment, a client device selects the lowest available bit rate initially. While playing the media, the client device can monitor available bandwidth (e.g. current network connection bit rates) to determine whether the available bandwidth can support use of a higher bit rate for playback. If so, the client device can select a higher bit rate and access the media files indicated by

the higher bit rate media playlist file. The reverse can also be supported. If the playback consumes too much bandwidth, the client device can select a lower bit rate and access the media files indicated by the lower bit rate media playlist file.

**[0117]** If the client device changes the bit rate in operation 394, for example, in response to a change in available bandwidth or in response to user input, the client device may select a different bit rate in operation 385. In one embodiment, to select a different bit rate the client device may utilize a different list of URIs included in the playlist file that corresponds to the new selected bit rate. In one embodiment, the client device may change bit rates during access of media files within a playlist.

**[0118]** If the bit rate does not change in operation 394, then the client device determines whether there are any more unplayed media files in the current playlist which have not been retrieved and presented. If such media files exist, then processing returns to operation 390 and one or more media files are retrieved using the URIs for those files in the playlist. If there are no such media files (i.e. all media files in the current playlist haven been played), then processing proceeds to operation 396 in which it is determined whether the playlist includes an end tag. If it does, the playback of the program has ended and the process has completed; if it does not, then processing reverts to operation 370, and the client device requests to reload the playlist for the program, and the process repeats through the method shown in Figure 3B.

**[0119]** **Figure 4** is a block diagram of one embodiment of a server stream agent. It will be understood that the elements of server stream agent 400 can be distributed across several server devices. For example, a first server device can include the segmenter 430, the indexer 440 and security 450 but not the file server 460 and a second server device can include the file server 450 but not the segmenter 430, the indexer 440 and security 450. In this example, the first server device would prepare the playlists and media files but would not transmit them to client devices while one or more second server devices would receive and optionally store the playlists and media files and would transmit the playlists and media files to the client devices. Server stream agent 400 includes control logic

410, which implements logical functional control to direct operation of server stream agent 400, and hardware associated with directing operation of server stream agent 400. Logic may be hardware logic circuits or software routines or firmware. In one embodiment, server stream agent 400 includes one or more applications 412, which represent code sequence and/or programs that provide instructions to control logic 410.

**[0120]** Server stream agent 400 includes memory 414, which represents a memory device or access to a memory resource for storing data or instructions. Memory 414 may include memory local to server stream agent 400, as well as, or alternatively, including memory of the host system on which server stream agent 400 resides. Server stream agent 400 also includes one or more interfaces 416, which represent access interfaces to/from (an input/output interface) server stream agent 400 with regard to entities (electronic or human) external to server stream agent 400.

**[0121]** Server stream agent 400 also can include server stream engine 420, which represents one or more functions that enable server stream agent 400 to provide the real-time, or near real-time, streaming as described herein. The example of Figure 4 provides several components that may be included in server stream engine 420; however, different or additional components may also be included. Example components that may be involved in providing the streaming environment include segmenter 430, indexer 440, security 450 and file server 460. Each of these components may further include other components to provide other functions. As used herein, a component refers to routine, a subsystem, etc., whether implemented in hardware, software, firmware or some combination thereof.

**[0122]** Segmenter 430 divides the content to be provided into media files that can be transmitted as files using a Web server protocol (e.g., HTTP). For example, segmenter 430 may divide the content into predetermined, fixed-size blocks of data in a pre-determined file format.

**[0123]** Indexer 440 may provide one or more playlist files that provide an address or URI to the media files created by segmenter 430. Indexer 440 may,

for example, create one or more files with a listing of an order for identifiers corresponding to each file created by segmenter 430. The identifiers may be created or assigned by either segmenter 430 or indexer 440. Indexer 440 can also include one or more tags in the playlist files to support access and/or utilization of the media files.

**[0124]** Security 450 may provide security features (e.g. encryption) such as those discussed above. Web server 460 may provide Web server functionality related to providing files stored on a host system to a remote client device. Web server 460 may support, for example, HTTP-compliant protocols.

**[0125]** **Figure 5** is a block diagram of one embodiment of a client stream agent. It will be understood that the elements of a client stream agent can be distributed across several client devices. For example, a first client device can include an assembler 530 and security 550 and can provide a decrypted stream of media files to a second client device that includes an output generator 540 (but does not include an assembler 530 and security 550). In another example, a primary client device can retrieve playlists and provide them to a secondary client device which retrieves media files specified in the playlist and generates an output to present these media files. Client stream agent 500 includes control logic 510, which implements logical functional control to direct operation of client stream agent 500, and hardware associated with directing operation of client stream agent 500. Logic may be hardware logic circuits or software routines or firmware. In one embodiment, client stream agent 500 includes one or more applications 512, which represent code sequence or programs that provide instructions to control logic 510.

**[0126]** Client stream agent 500 includes memory 514, which represents a memory device or access to a memory resource for storing data and/or instructions. Memory 514 may include memory local to client stream agent 500, as well as, or alternatively, including memory of the host system on which client stream agent 500 resides. Client stream agent 500 also includes one or more interfaces 516, which represent access interfaces to/from (an input/output

interface) client stream agent 500 with regard to entities (electronic or human) external to client stream agent 500.

**[0127]** Client stream agent 500 also can include client stream engine 520, which represents one or more functions that enable client stream agent 500 to provide the real-time, or near real-time, streaming as described herein. The example of Figure 5 provides several components that may be included in client stream engine 520; however, different or additional components may also be included. Example components that may be involved in providing the streaming environment include assembler 530, output generator 540 and security 550. Each of these components may further include other components to provide other functions. As used herein, a component refers to routine, a subsystem, etc., whether implemented in hardware, software, firmware or some combination thereof.

**[0128]** Assembler 530 can utilize a playlist file received from a server to access the media files via Web server protocol (e.g., HTTP) from the server. In one embodiment, assembler 530 may cause to be downloaded media files as indicated by URIs in the playlist file. Assembler 530 may respond to tags included in the playlist file.

**[0129]** Output generator 540 may provide the received media files as audio or visual output (or both audio and visual) on the host system. Output generator 540 may, for example, cause audio to be output to one or more speakers and video to be output to a display device. Security 550 may provide security features such as those discussed above.

**[0130]** **Figure 6** illustrates one embodiment of a playlist file with multiple tags. The example playlist of Figure 6 includes a specific number and ordering of tags. This is provided for description purposes only. Some playlist files may include more, fewer or different combinations of tags and the tags can be arranged in a different order than shown in Figure 6.

**[0131]** Begin tag 610 can indicate the beginning of a playlist file. In one embodiment, begin tag 610 is a #EXTM3U tag. Duration tag 620 can indicate the duration of the playback list. That is, the duration of the playback of the

media files indicated by playback list 600. In one embodiment, duration tag 620 is an EXT-X-TARGETDURATION tag; however, other tags can also be used.

**[0132]** Date/Time tag 625 can provide information related to the date and time of the content provided by the media files indicated by playback list 600. In one embodiment, Date/Time tag 625 is an EXT-X-PROGRAM-DATE-TIME tag; however, other tags can also be used. Sequence tag 630 can indicate the sequence of playlist file 600 in a sequence of playlists. In one embodiment, sequence tag 630 is an EXT-X-MEDIA-SEQUENCE tag; however, other tags can also be used.

**[0133]** Security tag 640 can provide information related to security and/or encryption applied to media files indicated by playlist file 600. For example, the security tag 640 can specify a decryption key to decrypt files specified by the media file indicators. In one embodiment, security tag 640 is an EXT-X-KEY tag; however, other tags can also be used. Variant list tag 645 can indicate whether variant streams are provided by playlist 600 as well as information related to the variant streams (e.g., how many, bit rate). In one embodiment, variant list tag 645 is an EXT-X-STREAM-INF tag.

**[0134]** Media file indicators 650 can provide information related to media files to be played. In one embodiment, media file indicators 650 include URIs to multiple media files to be played. In one embodiment, the order of the URIs in playlist 600 corresponds to the order in which the media files should be accessed and/or played. Subsequent playlist indictors 660 can provide information related to one or more playback files to be used after playback file 600. In one embodiment, subsequent playlist indicators 660 can include URIs to one or more playlist files to be used after the media files of playlist 600 have been played.

**[0135]** Memory tag 670 can indicate whether and/or how long a client device may store media files after playback of the media file content. In one embodiment, memory tag 670 is an EXT-X-ALLOW-CACHE tag. End tag 680 indicates whether playlist file 600 is the last playlist file for a presentation. In one embodiment, end tag 680 is an EXT-X-ENDLIST tag.

**[0136]** The following section contains several example playlist files according to one embodiment.

Simple Playlist file

```
#EXTM3U
#EXT-X-TARGETDURATION:10
#EXTINF:5220,
http://media.example.com/entire.ts
#EXT-X-ENDLIST
```

---

Sliding Window Playlist, using HTTPS

```
#EXTM3U
#EXT-X-TARGETDURATION:8
#EXT-X-MEDIA-SEQUENCE:2680

#EXTINF:8,
https://priv.example.com/fileSequence2680.ts
#EXTINF:8,
https://priv.example.com/fileSequence2681.ts
#EXTINF:8,
https://priv.example.com/fileSequence2682.ts
```

---

Playlist file with encrypted media files

```
#EXTM3U
#EXT-X-MEDIA-SEQUENCE:7794
#EXT-X-TARGETDURATION:15

#EXT-X-KEY:METHOD=AES-128,URI="
https://priv.example.com/key.php?r=52"
#EXTINF:15,
http://media.example.com/fileSequence7794.ts
#EXTINF:15,
http://media.example.com/fileSequence7795.ts
#EXTINF:15,
http://media.example.com/fileSequence7796.ts
#EXT-X-KEY:METHOD=AES-128,URI="
https://priv.example.com/key.php?r=53"
#EXTINF:15,
http://media.example.com/fileSequence7797.ts
```

---

Variant Playlist file

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1280000
http://example.com/low.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=2560000
http://example.com/mid.m3u8
```

```
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=7680000
http://example.com/hi.m3u8
#EXT-X-STREAM-INF:PROGRAM-
ID=1,BANDWIDTH=65000,CODECS="mp4a.40.5"
http://example.com/audio-only.m3u8
```

---

**[0137]** **Figure 7** is a flow diagram of one embodiment of a playback technique for assembled streams as described herein. In one embodiment, playback of the received media files can be controlled by the user to start, stop, rewind, etc. The playlist file is received by the client device in operation 700. The media files indicated by the playlist file are retrieved in operation 710. Output is generated based on the received media files in operation 720. Receiving and generating output based on media files can be accomplished as described above.

**[0138]** If control input is detected in operation 730, the client device can determine if the input indicates a stop in operation 740. If the input is a stop, the process concludes and playback stops. If the input indicates a rewind or forward request in operation 750, the client device can generate output based on previously played media files still stored in memory in operation 760. If these files are no longer in a cache, then processing reverts to operation 710 to retrieve the media files and repeats the process. In an alternate embodiment, playback can support a pause feature that halts playback without concluding playback as with a stop input.

**[0139]** Methods for transitioning from one stream to another stream are further described with reference to Figures 9A-9D. One client device can perform each of these methods or the operations of each of these methods can be distributed across multiple client devices as described herein; for example, in the distributed case, one client device can retrieve the variant playlist and the two media playlists and provide those to another client device which retrieves media files specified by the two media playlists and switches between the two streams provided by the retrieved media files. It will also be understood that, in alternative embodiments, the order of the operations shown may be modified or there can be more or fewer operations than shown in these figures. The methods

can use a variant playlist to select different streams. A variant playlist can be retrieved and processed in operation 901 to determine available streams for a program (e.g. a sporting event). Operation 901 can be done by a client device. A first stream can be selected from the variant playlist in operation 903, and a client device can then retrieve a media playlist for the first stream. The client device can process the media playlist for the first stream in operation 905 and also measure or otherwise determine a bit rate of the network connection for the first stream in operation 907. It will be appreciated that the sequence of operations may be performed in an order which is different than what is shown in Figure 9A; for example, operation 907 may be performed during operation 903, etc. In operation 911 the client device selects an alternative media playlist from the variant playlist based on the measured bit rate from operation 907; this alternative media playlist may be at a second bit rate that is higher than the existing bit rate of the first stream. This typically means that alternative stream will have a higher resolution than the first stream. The alternative media playlist can be selected if it is a better match than the current playlist for the first stream based on current conditions (e.g. the bit rate measured in operation 907). In operation 913, the alternative media playlist for an alternate stream is retrieved and processed. This typically means that the client device can be receiving and processing both the first stream and the alternative stream so both are available for presentation; one is presented while the other is ready to be presented. The client device then selects a transition point to switch between the versions of the streams in operation 915 and stops presenting the first stream and begins presenting the alternative stream. Examples of how this switch is accomplished are provided in conjunction with Figures 9B-9D. In some embodiments, the client device can stop receiving the first stream before making the switch.

**[0140]** Figure 9B shows that the client device retrieves, stores and presents content specified by the first media playlist (e.g. the first stream) in operations 921 and 923, and while the content specified by the first playlist is being presented the client device in operation 925 also retrieves and stores content specified by the second media playlist (e.g. the second stream). The retrieval and

storage (e.g. in a temporary buffer) of the content specified by the second media playlist while presenting the content obtained from the first media playlist creates an overlap 955 in time of the program's content (shown in Figure 9D) that allows the client device to switch between the versions of the program without a substantial interruption of the program. In this way, the switch between the versions of the program can be achieved in many cases without the user noticing that a switch has occurred (although the user may notice a higher resolution image after the switch in some cases) or without a substantial interruption in the presentation of the program. In operation 927, the client device determines a transition point at which to switch from content specified by the first media playlist to content specified by the second media playlist; an example of a transition point (transition point 959) is shown in Figure 9D. The content specified by the second media playlist is then presented in operation 931 after the switch.

**[0141]** The method shown in Figures 9C and 9D represents one embodiment for determining the transition point; this embodiment relies upon a pattern matching on audio samples from the two streams 951 and 953 to determine the transition point. It will be appreciated that alternative embodiments can use pattern matching on video samples or can use the timestamps in the two streams, etc. to determine the transition point. The method can include, in operation 941, storing content (e.g. stream 951) specified by the first media playlist in a buffer; the buffer can be used for the presentation of the content and also for the pattern matching operation. The stream 951 includes both audio samples 951A and video samples 951B. The video samples can use a compression technique which relies on i-frames or key frames which have all necessary content to display a single video frame. The content in stream 951 can include timestamps specifying a time (e.g. time elapsed since the beginning of the program), and these timestamps can mark the beginning of each of the samples (e.g. the beginning of each of the audio samples 951A and the beginning of each of the video samples 951B). In some cases, a comparison of the timestamps between the two streams may not be useful in determining a transition point because they

may not be precise enough or because of the difference in the boundaries of the samples in the two streams; however, a comparison of the timestamps ranges can be used to verify there is an overlap 955 in time between the two streams. In operation 943, the client device stores in a buffer content specified by the second media playlist; this content is for the same program as the content obtained from the first media playlist and it can include timestamps also. In one embodiment, timestamps, if not present in a stream, can be added to a playlist for a stream; for example, in one embodiment an ID3 tag which includes one or more timestamps can be added to an entry in a playlist, such as a variant playlist or a media playlist. The entry may, for example, be in a URI for a first sample of an audio stream. Figure 9D shows an example of content 953 obtained from the second media playlist, and this includes audio samples 953A and video samples 953B. In operation 945, the client device can perform a pattern matching on the audio samples in the two streams 951 and 953 to select from the overlap 955 the transition point 959 which can be, in one embodiment, the next self contained video frame (e.g. i-frame 961) after the matched audio segments (e.g. segments 957). Beginning with i-frame 961 (and its associated audio sample), presentation of the program uses the second stream obtained from the second media playlist. The foregoing method can be used in one embodiment for both a change from a slower to a faster bit rate and for a change from a faster to a slower bit rate, but in another embodiment the method can be used only for a change from a slower to a faster bit rate and another method (e.g. do not attempt to locate a transition point but attempt to store and present content from the slower bit rate stream as soon as possible) can be used for a change from a faster to a slower bit.

**[0142] Figure 10** is a flow diagram of one embodiment of a technique for providing multiple redundant locations that provide playlists or media content or both to client devices using alternative streams. If a playlist contains alternate streams as discussed above, then alternate streams can not only operate as bandwidth or device alternates, but also as failure fallbacks. For example, if the client is unable to reload the playlist file for a stream (due to a 404 error or a network connection error, for example), the client can attempt to switch to an

alternate stream. Referring to Figure 10, to implement failover protection, a first server device or first content distribution service is configured to create a stream, or multiple alternate bandwidth streams in operation 1002 as discussed in conjunction with the description of Figure 2C. In operation 1004, the first server device or first content distribution service generates playlist file(s) from the stream(s) generated in operation 1002. A second server device or second content distribution service can create a parallel stream, or set of streams, in operation 1006 and also create a playlist. These parallel stream(s) can be considered backup streams. Next, the list of backup streams is added to the playlist file(s) in operation 1008 so that the backup stream(s) at each bandwidth is listed after the primary stream. For example, if the primary stream comes from server ALPHA, and the backup stream is on server BETA, then a playlist file might be as follows:

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=200000
http://ALPHA.mycompany.com/low/prog_index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=200000
http://BETA.mycompany.com/low/prog_index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=500000
http://ALPHA.mycompany.com/mid/prog_index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=500000
http://BETA.mycompany.com/mid/prog_index.m3u8
```

**[0143]** Note that the backup streams are intermixed with the primary streams in the playlist with the backup at each bandwidth is listed after the primary for that bandwidth. A client is not limited to a single backup stream set. In the example above, ALPHA and BETA could be followed by GAMMA, for instance. Similarly, it is not necessary to provide a complete parallel set of streams. A single low-bandwidth stream may be provided on a backup server, for example.

**[0144]** In operation 1010, the client attempts to download playlist file(s) from a first URL using a first stream associated with the first server device or the first

content distribution service. **Figure 11** illustrates a network in which a client 1102 communicates bi-directionally with one or more URLs, server devices or content distribution services, in accordance with one embodiment. The playlist file(s) may be transmitted from the first URL, server device or content distribution service in operation 1012 to the client 1102. If a client is unable to download the playlist file(s) from the first URL, server device, or content distribution service (e.g., due to an error in reloading the index file for a stream), the client attempts to switch to an alternate stream. In the event of a failure (e.g., index load failure) on one stream (e.g., operation 1010), the client chooses the highest bandwidth alternate stream that the network connection supports in operation 1014. If there are multiple alternates at the same bandwidth, the client chooses among them in the order listed in the playlist. For example, if the client 1102 is not able to successfully download from URL 1, it may download from URL 2 or another URL in which case the playlist file(s) are transmitted from the alternative URL to the client. This feature provides redundant streams that will allow media to reach clients even in the event of severe local failures, such as a server crashing or a content distributor node going down.

**[0145]** The failover protection provides the ability to provide multiple redundant locations from which clients can retrieve playlists and media files. Thus, if the client cannot retrieve a stream from a first location, it can attempt to access the stream from a secondary, tertiary, etc. location.

**[0146]** In one embodiment, to indicate the additional locations from which the client can retrieve a playlist, the same variant playlist tag would be provided with the same bandwidth, but a new URI of the redundant location. The client initially can attempt to access the first URL associated with the desired bandwidth. If it cannot download the playlist from the first URL, it then can attempt to access the next URL presented for the bandwidth, and so on until it has exhausted all the possibilities.

**[0147]** An example below includes 1 redundant location for the 2560000 bandwidth and 2 redundant locations for the 7680000 bandwidth.

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1280000
http://example.com/low.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=2560000
http://example.com/mid.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=2560000
http://example1.com/mid-redundant2.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=7680000
http://example.com/hi.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=7680000
http://example2.com/hi-redudant2.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=7680000
http://example3.com/hi-redudant3.m3u8
#EXT-X-STREAM-INF:PROGRAM-
ID=1,BANDWIDTH=65000,CODECS="mp4a.40.5"
http://example.com/audio-only.m3u8
```

**[0148]** Note that in this example both the filenames (e.g., mid-redundant2.m3u8) and the actual URL (e.g., http://example2.com <http://example2.com/> , http://example3.com <http://example3.com/> ) change. However, in one embodiment, a redundant location can be a change only to the filename or only to the website.

**[0149]** In one embodiment, a playlist can be compressed by a server device and sent to a client device in a compressed form. The compressed playlist normally requires fewer bits to represent the playlist than an uncompressed playlist, and hence a compressed playlist uses less available bandwidth of a network, such as a wireless cellular telephone network, when being transmitted or received. In one embodiment, the playlist can be compressed by a web server according to a built-in compression technique or facility that is used by a web server that is compliant with or compatible with a transfer protocol such as the HTTP 1.1 standard protocol; an example of such a compression technique or facility is the deflate or

the gzip compression facility of HTTP 1.1. Other standards based compression facilities which are part of a standards based transfer protocol can be used in other embodiments. The use of compressed playlists can be, in one embodiment, an optional feature of server devices and client devices. In one embodiment, the playlist can be textual content (e.g. a text file) and be compressed efficiently with deflate or gzip by a standards based web server and then decompressed automatically by a client device. A description of a version of the gzip compression facility can be found at www.ietf.org/rfc/rfc1952.txt; a version of the deflate compression facility can be found at www.ietf.org/rfc/rfc1951.txt. Many web servers and many web browsers on a client device can automatically support the deflate or the gzip facilities.

**[0150]** In one embodiment, a client device can periodically request an updated playlist; for example, the client device can request, from a server, an updated playlist every few seconds (e.g. every 10, 20, or 30 seconds or some other period of time). A growing playlist, such as a playlist for a live on-going baseball game that allows a client to start viewing from the beginning of the live game at any time during the live game, can become large enough that use of compression can limit the consumption of a network's bandwidth as the growing playlist is repeatedly sent through the network.

**[0151]** In one embodiment, a client device can optionally specify, when it requests a playlist (such as an updated playlist), what compression techniques it can support (such as deflate or gzip); support for these techniques means that the client device can decompress or decode the compressed or encoded content. The client device's request for a playlist, with the optional specification of a compression technique, is received by a web server which, in one embodiment, is not required to support a compression technique for a playlist but can send an uncompressed playlist. The web server can respond to the client device's request by sending, to the client device, an uncompressed playlist or a playlist compressed using one of the compression techniques specified in the client device's request for the playlist. The client device receives the playlist and uses it as described herein; if the playlist

is compressed, it is decoded using a decoder on the client device such as a decoder in a web browser on the client device.

**[0152]** Figure 8 is a block diagram of one embodiment of an electronic system. The electronic system illustrated in Figure 8 is intended to represent a range of electronic systems (either wired or wireless) including, for example, desktop computer systems, laptop computer systems, cellular telephones, personal digital assistants (PDAs) including cellular-enabled PDAs, set top boxes, entertainment systems or other consumer electronic devices. Alternative electronic systems may include more, fewer and/or different components. The electronic system of Figure 8 may be used to provide the client device and/or the server device.

**[0153]** Electronic system 800 includes bus 805 or other communication device to communicate information, and processor 810 coupled to bus 805 that may process information. While electronic system 800 is illustrated with a single processor, electronic system 800 may include multiple processors and/or co-processors. Electronic system 800 further may include random access memory (RAM) or other dynamic storage device 820 (referred to as main memory), coupled to bus 805 and may store information and instructions that may be executed by processor 810. Main memory 820 may also be used to store temporary variables or other intermediate information during execution of instructions by processor 810.

**[0154]** Electronic system 800 may also include read only memory (ROM) and/or other static storage device 830 coupled to bus 805 that may store static information and instructions for processor 810. Data storage device 840 may be coupled to bus 805 to store information and instructions. Data storage device 840 such as flash memory or a magnetic disk or optical disc and corresponding drive may be coupled to electronic system 800.

**[0155]** Electronic system 800 may also be coupled via bus 805 to display device 850, such as a cathode ray tube (CRT) or liquid crystal display (LCD), to display information to a user. Electronic system 800 can also include an alphanumeric input device 860, including alphanumeric and other keys, which may be coupled to bus 805 to communicate information and command selections to processor

810. Another type of user input device is cursor control 870, such as a touchpad, a mouse, a trackball, or cursor direction keys to communicate direction information and command selections to processor 810 and to control cursor movement on display 850.

**[0156]** Electronic system 800 further may include one or more network interface(s) 880 to provide access to a network, such as a local area network. Network interface(s) 880 may include, for example, a wireless network interface having antenna 885, which may represent one or more antenna(e). Electronic system 800 can include multiple wireless network interfaces such as a combination of WiFi, Bluetooth and cellular telephony interfaces. Network interface(s) 880 may also include, for example, a wired network interface to communicate with remote devices via network cable 887, which may be, for example, an Ethernet cable, a coaxial cable, a fiber optic cable, a serial cable, or a parallel cable.

**[0157]** In one embodiment, network interface(s) 880 may provide access to a local area network, for example, by conforming to IEEE 802.11b and/or IEEE 802.11g standards, and/or the wireless network interface may provide access to a personal area network, for example, by conforming to Bluetooth standards. Other wireless network interfaces and/or protocols can also be supported.

**[0158]** In addition to, or instead of, communication via wireless LAN standards, network interface(s) 880 may provide wireless communications using, for example, Time Division, Multiple Access (TDMA) protocols, Global System for Mobile Communications (GSM) protocols, Code Division, Multiple Access (CDMA) protocols, and/or any other type of wireless communications protocol.

**[0159]** Reference in the specification to "one embodiment" or "an embodiment" means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the invention. The appearances of the phrase "in one embodiment" in various places in the specification are not necessarily all referring to the same embodiment.

**[0160]** In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that

various modifications and changes can be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

# APPENDIX

The following Appendix is a draft specification of a protocol according to a particular embodiment of the invention. It will be understood that the use of certain key words (e.g. MUST, MUST NOT, SHALL, SHALL NOT, etc.) in this Appendix apply to this particular embodiment and do not apply to other embodiments described in this disclosure.

Abstract

This document describes a protocol for transmitting unbounded streams of multimedia data over HTTP. It specifies the data format of the files and the actions to be taken by the server (sender) and the clients (receivers) of the streams. It describes version 1.0 of this protocol.

**Table of Contents**

1. Introduction
2. Summary
3. The Playlist file
    3.1. New Tags
        3.1.1. EXT-X-TARGETDURATION
        3.1.2. EXT-X-MEDIA-SEQUENCE
        3.1.3. EXT-X-KEY
        3.1.4. EXT-X-PROGRAM-DATE-TIME
        3.1.5. EXT-X-ALLOW-CACHE
        3.1.6. EXT-X-ENDLIST
        3.1.7. EXT-X-STREAM-INF
        3.1.8. EXT-X-DISCONTINUITY
4. Media files
5. Key files
    5.1. IV for AES-128
6. Client/Server Actions
    6.1. Server Process
        6.1.1. Sliding Window Playlists
        6.1.2. Encrypting media files
        6.1.3. Providing variant streams
    6.2. Client Process
        6.2.1. Loading the Playlist file
        6.2.2. Playing the Playlist file
        6.2.3. Reloading the Playlist file
        6.2.4. Determining the next file to load

6.2.5. Playing encrypted media files

7. Examples

7.1. Simple Playlist file

7.2. Sliding Window Playlist, using HTTPS

7.3. Playlist file with encrypted media files

7.4. Variant Playlist file

8. Security Considerations

9. References

Normative References

Informative References

1. Introduction

This document describes a protocol for transmitting unbounded streams of multimedia data over HTTP [RFC2616]. The protocol supports the encryption of media data, and the provision of alternate versions (e.g. bitrates) of a stream. Media data can be transmitted soon after it is created, allowing it to be received in near real-time.

External references that describe related standards such as HTTP are listed in Section 9.

2. Summary

A multimedia presentation is specified by a URI [RFC3986] to a Playlist file, which is an ordered list of additional URIs. Each URI in the Playlist file refers to a media file which is a segment of a single contiguous stream.

To play the stream, the client first obtains the Playlist file and then obtains and plays each media file in the Playlist. It reloads the Playlist file as described in this document to discover additional segments.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. The Playlist file

Playlists MUST be Extended M3U Playlist files [M3U]. This document extends the M3U file format by defining additional tags.

An M3U Playlist is a text file that consists of individual lines. Lines are terminated by either a single LF character or a CR character followed by an LF character. Each line is a URI, a blank, or starts with the comment character '#'. URIs identify media files to be played. Blank lines are ignored.

Lines that start with the comment character '#' are either comments or tags. Tags begin with #EXT. All other lines that begin with '#' are comments and SHOULD be ignored.

M3U Playlist files whose names end in .m3u8 and/or have the HTTP Content-Type "application/vnd.apple.mpegurl" are encoded in UTF-8 [RFC3629]. Files whose names end with .m3u and/or have the HTTP Content-Type [RFC2616] "audio/mpegurl" are encoded in US-ASCII [US_ASCII].

Implementations SHOULD produce Playlist files whose names end in .m3u8 or, if transmitted over HTTP, have the Content-Type "application/vnd.apple.mpegurl". For compatibility, implementations MAY produce Playlist files whose names end in .m3u and/or have the HTTP Content-Type type "audio/mpegurl".

The Extended M3U file format defines two tags: EXTM3U and EXTINF. An Extended M3U file is distinguished from a basic M3U file by its first line, which MUST be #EXTM3U.

EXTINF is a record marker that describes the media file identified by the URI that follows it. Each media file URI MUST be preceded by an EXTINF tag. Its format is:

#EXTINF:<duration>,<title>

"duration" is an integer that specifies the duration of the media file in seconds. Durations SHOULD be rounded to the nearest integer. The remainder of the line following the comma is the title of the media file.

3.1. New Tags

This document defines seven new tags: EXT-X-TARGETDURATION, EXT-X-MEDIA-SEQUENCE, EXT-X-KEY, EXT-X-PROGRAM-DATE-TIME, EXT-X-ALLOW-CACHE, EXT-X-STREAM-INF, and EXT-X-ENDLIST.

3.1.1. EXT-X-TARGETDURATION

The EXT-X-TARGETDURATION tag indicates the approximate duration of the next media file that will be added to the main presentation. It MUST appear in the Playlist file. Its format is:

#EXT-X-TARGETDURATION:<seconds>

The actual duration of the media file MAY differ slightly from the target duration.

3.1.2. EXT-X-MEDIA-SEQUENCE

Each media file URI in a Playlist has a unique sequence number. The sequence number of a URI is equal to the sequence number of the URI that preceded it plus one. The EXT-X-MEDIA-SEQUENCE tag indicates the sequence number of the first URI that appears in a Playlist file. Its format is:

#EXT-X-MEDIA-SEQUENCE:<number>

If the Playlist file does not contain an EXT-X-MEDIA-SEQUENCE tag then the sequence number of the first URI in the playlist SHALL be considered to be 1.

See Section 6.2.1 and Section 6.2.4 for information on handling the EXT-X-MEDIA-SEQUENCE tag.

3.1.3. EXT-X-KEY

Media files MAY be encrypted. The EXT-X-KEY tag provides information necessary to decrypt media files that follow it. Its format is:

#EXT-X-KEY:METHOD=<method>[,URI="<URI>"]

The METHOD parameter specifies the encryption method. The URI parameter, if present, specifies how to obtain the key.

Version 1.0 of the protocol defines two encryption methods: NONE and AES-128. An encryption method of NONE means that media files are not encrypted.

An encryption method of AES-128 means that media files are encrypted using the Advanced Encryption Standard [AES_128] with a 128-bit key and PKCS7 padding [RFC3852].

A new EXT-X-KEY supersedes any prior EXT-X-KEY.

If no EXT-X-KEY tag is present then media files are not encrypted.

See Section 5 for the format of the key file, and Section 5.1,

Section 6.1.2 and Section 6.2.5 for additional information on media file encryption.

3.1.4. EXT-X-PROGRAM-DATE-TIME

The EXT-X-PROGRAM-DATE-TIME tag associates the beginning of the next media file with an absolute date and/or time. The date/time representation is ISO/IEC 8601:2004 [ISO_8601] and SHOULD indicate a time zone. For example:

#EXT-X-PROGRAM-DATE-TIME:<YYYY-MM-DDThh:mm:ssZ>

3.1.5. EXT-X-ALLOW-CACHE

The EXT-X-ALLOW-CACHE tag indicates whether the client MAY cache downloaded media files for later replay. Its format is:

#EXT-X-ALLOW-CACHE:<YES|NO>

3.1.6. EXT-X-ENDLIST

The EXT-X-ENDLIST tag indicates that no more media files will be added to the Playlist file. Its format is:

#EXT-X-ENDLIST

3.1.7. EXT-X-STREAM-INF

The EXT-X-STREAM-INF tag indicates that the next URI in the Playlist file identifies another Playlist file. Its format is:

#EXT-X-STREAM-INF:[attribute=value][,attribute=value]*
<URI>

The following attributes are defined for the EXT-X-STREAM-INF tag:

BANDWIDTH=<n>

where n is an approximate upper bound of the stream bitrate, expressed as a number of bits per second.

PROGRAM-ID=<i>

where i is a number that uniquely identifies a particular presentation within the scope of the Playlist file.

A Playlist file MAY contain multiple EXT-X-STREAM-INF URIs with the same PROGRAM-ID to describe variant streams of the same presentation.

CODECS="[format][,format]*"

where each format specifies a media sample type that is present in a media file in the Playlist file.

Valid format identifiers are those in the ISO File Format Name Space defined by RFC 4281 [RFC4281].

3.1.8. EXT-X-DISCONTINUITY

The EXT-X-DISCONTINUITY tag indicates that the media file following it has different characteristics than the one that preceded it. The set of characteristics that MAY change is:

- o file format
- o number and type of tracks
- o encoding parameters

o encoding sequence

o timestamp sequence

Its format is:

#EXT-X-DISCONTINUITY

4. Media files

Each media file URI in a Playlist file MUST identify a media file which is a segment of the overall presentation. Each media file MUST be formatted as an MPEG-2 Transport Stream, an MPEG-2 Program Stream, or an MPEG-2 audio elementary stream [ISO_13818]. All media files in a presentation MUST have the same format.

Transport Stream files MUST contain a single MPEG-2 Program. There SHOULD be a Program Association Table and a Program Map Table at the start of each file. A file that contains video SHOULD have at least one key frame and enough information to completely initialize a video decoder.

Clients SHOULD be prepared to handle multiple tracks of a particular type (e.g. audio or video) by choosing a reasonable subset. Clients MUST ignore private streams inside Transport Streams that they do not recognize.

The encoding parameters for samples within a stream inside a media file and between corresponding streams across multiple media files SHOULD remain consistent. However clients SHOULD deal with encoding changes as they are encountered, for example by scaling video content to accommodate a resolution change.

5. Key files

An EXT-X-KEY tag with the URI parameter identifies a Key file. A Key

file contains the cipher key that MUST be used to decrypt subsequent media files in the Playlist.

The AES-128 encryption method uses 16-octet keys. The format of the Key file is simply a packed array of these 16 octets in binary format.

5.1. IV for AES-128

128-bit AES requires the same 16-octet Initialization Vector (IV) to be supplied when encrypting and decrypting. Varying this IV increases the strength of the cipher.

When using the encryption METHOD AES-128, implementations SHALL use the sequence number of the media file as the IV when encrypting or decrypting media files. The big-endian binary representation of the sequence number SHALL be placed in a 16-octet buffer and padded (on the left) with zeros.

6. Client/Server Actions

This section describes how the server generates the Playlist and media files and how the client should download and play them.

6.1. Server Process

The production of the MPEG-2 stream is outside the scope of this document, which simply presumes a source of a continuous stream containing the main presentation.

The server MUST divide the stream into individual media files whose duration is approximately equal. The server SHOULD attempt to divide the stream at points that support effective decode of individual

media files, e.g. on packet and key frame boundaries.

The server MUST create a URI for each media file that will allow its clients to obtain the file.

The server MUST create a Playlist file. The Playlist file MUST conform to the format described in Section 3. A URI for each media file that the server wishes to make available MUST appear in the Playlist in the order in which it is to be played. The entire media file MUST be available to clients if its URI is in the Playlist file.

The Playlist file MUST contain an EXT-X-TARGETDURATION tag. It MUST indicate the approximate duration of the next media file to be added to the main presentation. This value MUST remain constant for the entire presentation. A typical target duration is 10 seconds.

The server MUST create a URI for the Playlist file that will allow its clients to obtain the file.

Changes to the Playlist file MUST be made atomically from the point of view of the clients.

Every media file URI in a Playlist MUST be prefixed with an EXTINF tag indicating the approximate duration of the media file.

The server MAY associate an absolute date and time with a media file by prefixing its URI with an EXT-X-PROGRAM-DATE-TIME tag. The value of the date and time is arbitrary.

If the Playlist contains the final media file of the presentation then the Playlist file MUST contain the EXT-X-ENDLIST tag.

If the server wishes to remove an entire presentation, it MUST make the Playlist file unavailable to clients. It SHOULD ensure that all media files in the Playlist file remain available to clients for at least the duration of the Playlist file at the time of removal.

6.1.1. Sliding Window Playlists

The server MAY limit the availability of media files to those which have been most recently added to the Playlist. To do so the Playlist file MUST ALWAYS contain exactly one EXT-X-MEDIA-SEQUENCE tag. Its value MUST be incremented by 1 for every media file URI that is removed from the Playlist file.

Media file URIs MUST be removed from the Playlist file in the order in which they were added.

When the server removes a media file URI from the Playlist, the media file MUST remain available to clients for a period of time equal to the duration of the media file plus the duration of the longest Playlist file in which the media file has appeared. The duration of a Playlist file is the sum of the durations of the media files within it.

If a server plans to remove a media file, it SHOULD ensure that an HTTP Expires header reflects the planned time-to-live when it is delivered to clients.

The server MUST maintain at least three main presentation media files in the Playlist at all times unless the EXT-X-ENDLIST tag is present.

6.1.2. Encrypting media files

If media files are to be encrypted the server MUST define a URI which

will allow authorized clients to obtain a Key file containing a decryption key. The Key file MUST conform to the format described in Section 5.

The server MAY set the Expires header in the key response to indicate that the key may be cached.

If the encryption METHOD is AES-128, AES-128 CBC encryption SHALL be applied to individual media files. The entire file MUST be encrypted. Cipher Block Chaining MUST NOT be applied across media files. The sequence number of the media file MUST be used as the IV as described in Section 5.1.

The server MUST encrypt every media file in a Playlist using the method specified by the EXT-X-KEY tag that most immediately precedes its URI in the Playlist file. Media files preceded by an EXT-X-KEY tag whose METHOD is NONE, or not preceded by any EXT-X-KEY tag, MUST NOT be encrypted.

The URI of every EXT-X-KEY tag must be distinct from the URI of every other EXT-X-KEY tag that appears or has appeared in the Playlist file, unless its METHOD is NONE. An EXT-X-KEY tag with a METHOD of NONE MUST NOT contain a URI parameter.

The server MUST NOT remove an EXT-X-KEY tag from the Playlist file if the Playlist file contains a URI to a media file encrypted with that key.

6.1.3. Providing variant streams

A server MAY offer multiple Playlist files to provide different encodings of the same presentation. If it does so it SHOULD provide

a variant Playlist file that lists each variant stream to allow clients to switch between encodings dynamically.

Variant Playlists MUST contain an EXT-X-STREAM-INF tag for each variant stream. Each EXT-X-STREAM-INF tag for the same presentation MUST have the same PROGRAM-ID attribute value. The PROGRAM-ID value for each presentation MUST be unique within the variant Playlist.

If an EXT-X-STREAM-INF tag contains the CODECS attribute, the attribute value MUST include every format defined by [RFC4281] that is present in any media file that appears or will appear in the Playlist file.

The server MUST meet the following constraints when producing variant streams:

Each variant stream MUST consist of the same content, including content which is not part of the main presentation.

The server MUST make the same period of content available for all variant streams, within an accuracy of the smallest target duration of the streams.

Matching content in variant streams MUST have matching timestamps. This allows clients to synchronize the streams.

Elementary Audio Stream files MUST signal the timestamp of the first sample in the file by prepending an ID3 PRIV tag [ID3] with an owner identifier of "com.apple.streaming.transportStreamTimestamp". The binary data MUST be a 33-bit MPEG-2 Program Elementary Stream timestamp

expressed as a big-endian eight-octet number.

In addition, all variant streams SHOULD contain the same encoded audio bitstream. This allows clients to switch between streams without audible glitching.

6.2. Client Process

How the client obtains the URI to the Playlist file is outside the scope of this document; it is presumed to have done so.

The client MUST obtain the Playlist file from the URI. If the Playlist file so obtained is a variant Playlist, the client MUST obtain the Playlist file from the variant Playlist.

This document does not specify the treatment of variant streams by clients.

6.2.1. Loading the Playlist file

Every time a Playlist file is loaded or reloaded from the Playlist URI:

The client SHOULD check that the Playlist file begins with #EXTM3U and refuse to continue if it does not. The client SHOULD ignore any tags it does not recognize.

The client MUST determine the next media file to load as described in Section 6.2.4.

If the Playlist contains the EXT-X-MEDIA-SEQUENCE tag, the client SHOULD assume that each media file in it will become unavailable at the time that the Playlist file was loaded plus the duration of the

Playlist file. The duration of a Playlist file is the sum of the durations of the media files within it.

6.2.2. Playing the Playlist file

The client SHALL choose which media file to play first from the Playlist when playback starts. If the Playlist file contains the EXT-X-ENDLIST tag, any file in the Playlist MAY be played first. If the EXT-X-ENDLIST tag is not present, any file except for the last and second-to-last files in the Playlist MAY be played first.

Once the first media file to play has been chosen, subsequent media files in the Playlist MUST be loaded in the order that they appear and played in the order that they are loaded.

The client SHOULD attempt to load media files in advance of when they will be required for uninterrupted playback to compensate for temporary variations in latency and throughput.

If the Playlist file contains the EXT-X-ALLOW-CACHE tag and its value is NO, the client MUST NOT cache downloaded media files after they have been played. Otherwise the client MAY cache downloaded media files indefinitely for later replay.

The client MAY use the value of the EXT-X-PROGRAM-DATE-TIME tag to display the program origination time to the user. If the value includes time zone information the client SHALL take it into account, but if it does not the client MUST NOT infer an originating time zone.

The client MUST NOT depend upon the correctness or the consistency of the value of the EXT-X-PROGRAM-DATE-TIME tag.

6.2.3. Reloading the Playlist file

The client MUST periodically reload the Playlist file unless it contains the EXT-X-ENDLIST tag.

However the client MUST NOT attempt to reload the Playlist file more frequently than specified by this section.

When a client loads a Playlist file for the first time or reloads a Playlist file and finds that it has changed since the last time it was loaded, the client MUST wait for a period of time before attempting to reload the Playlist file again. This period is called the initial minimum reload delay. It is measured from the time that the client began loading the Playlist file.

The initial minimum reload delay is the duration of the last media file in the Playlist or 3 times the target duration, whichever is less. Media file duration is specified by the EXTINF tag.

If the client reloads a Playlist file and finds that it has not changed then it MUST wait for a period of time before retrying. The minimum delay is three times the target duration or a multiple of the initial minimum reload delay, whichever is less. This multiple is 0.5 for the first attempt, 1.5 for the second, and 3.0 thereafter.

6.2.4. Determining the next file to load

The client MUST examine the Playlist file every time it is loaded or reloaded to determine the next media file to load.

The first file to load MUST be the file that the client has chosen to play first, as described in Section 6.2.2.

If the first file to be played has been loaded and the Playlist file does not contain the EXT-X-MEDIA-SEQUENCE tag then the client MUST verify that the current Playlist file contains the URI of the last loaded media file at the offset it was originally found at, halting playback if it does not. The next media file to load MUST be the first media file URI following the last-loaded URI in the Playlist.

If the first file to be played has been loaded and the Playlist file contains the EXT-X-MEDIA-SEQUENCE tag then the next media file to load SHALL be the one with the lowest sequence number that is greater than the sequence number of the last media file loaded.

6.2.5. Playing encrypted media files

If a Playlist file contains an EXT-X-KEY tag that specifies a Key file URI, the client MUST obtain that key file and use the key inside it to decrypt all media files following the EXT-X-KEY tag until another EXT-X-KEY tag is encountered.

If the encryption METHOD is AES-128, AES-128 CBC decryption SHALL be applied to individual media files. The entire file MUST be decrypted. Cipher Block Chaining MUST NOT be applied across media files. The sequence number of the media file MUST be used as the IV as described in Section 5.1.

If the encryption METHOD is NONE, the client MUST treat all media files following the EXT-X-KEY tag as cleartext (not encrypted) until another EXT-X-KEY tag is encountered.

7. Examples

This section contains several example Playlist files.

7.1. Simple Playlist file

```
#EXTM3U
#EXT-X-TARGETDURATION:10
#EXTINF:5220,
http://media.example.com/entire.ts
#EXT-X-ENDLIST
```

7.2. Sliding Window Playlist, using HTTPS

```
#EXTM3U
#EXT-X-TARGETDURATION:8
#EXT-X-MEDIA-SEQUENCE:2680
#EXTINF:8,
https://priv.example.com/fileSequence2680.ts
#EXTINF:8,
https://priv.example.com/fileSequence2681.ts
#EXTINF:8,
https://priv.example.com/fileSequence2682.ts
```

7.3. Playlist file with encrypted media files

```
#EXTM3U
#EXT-X-MEDIA-SEQUENCE:7794
#EXT-X-TARGETDURATION:15

#EXT-X-KEY:METHOD=AES-
128,URI="https://priv.example.com/key.php?r=52"

#EXTINF:15,
http://media.example.com/fileSequence7794.ts
#EXTINF:15,
http://media.example.com/fileSequence7795.ts
```

#EXTINF:15,
http://media.example.com/fileSequence7796.ts

#EXT-X-KEY:METHOD=AES-
128,URI="https://priv.example.com/key.php?r=53"

#EXTINF:15,
http://media.example.com/fileSequence7797.ts

7.4. Variant Playlist file

#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1280000
http://example.com/low.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=2560000
http://example.com/mid.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=7680000
http://example.com/hi.m3u8
#EXT-X-STREAM-INF:PROGRAM-
ID=1,BANDWIDTH=65000,CODECS="mp4a.40.5"
http://example.com/audio-only.m3u8

8. Security Considerations

Since the protocol generally uses HTTP to transmit data, most of the same security considerations apply. See section 15 of RFC 2616 [RFC2616].

Media file parsers are typically subject to "fuzzing" attacks. Clients SHOULD take care when parsing files received from a server so that non-compliant files are rejected.

Playlist files contain URIs, which clients will use to make network

requests of arbitrary entities. Clients SHOULD range-check responses to prevent buffer overflows. See also the Security Considerations section of RFC 3986 [RFC3986].

Clients SHOULD load resources identified by URI lazily to avoid contributing to denial-of-service attacks.

HTTP requests often include session state ("cookies"), which may contain private user data. Implementations MUST follow cookie restriction and expiry rules specified by RFC 2965 [RFC2965]. See also the Security Considerations section of RFC 2965, and RFC 2964 [RFC2964].

Encryption keys are specified by URI. The delivery of these keys SHOULD be secured by a mechanism such as HTTP over TLS [RFC5246] (formerly SSL) in conjunction with a secure realm or a session cookie.

## 9. References

### Normative References

[AES_128] U.S. Department of Commerce/National Institute of Standards and Technology, "Advanced Encryption Standard (AES), FIPS PUB 197", November 2001, <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.

[ISO_13818] International Organization for Standardization, "ISO/IEC International Standard 13818; Generic coding of moving pictures and associated audio information", November 1994, <http://www.iso.org/iso/catalogue_detail?csnumber=44169>.

[ISO_8601]

International Organization for Standardization, "ISO/IEC International Standard 8601:2004; Data elements and interchange formats -- Information interchange -- Representation of dates and times", December 2004, <http://www.iso.org/iso/catalogue_detail?csnumber=40874>.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3852] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 3852, July 2004.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4281] Gellens, R., Singer, D., and P. Frojdh, "The Codecs Parameter for "Bucket" Media Types", RFC 4281, November 2005.

[US_ASCII]

American National Standards Institute, "ANSI X3.4-1986, Information Systems -- Coded Character Sets 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII)", December 1986.

Informative References

[ID3] ID3.org, "The ID3 audio file data tagging format", <http://www.id3.org/Developer_Information>.

[M3U] Nullsoft, Inc., "The M3U Playlist format, originally invented for the Winamp media player", <http://wikipedia.org/wiki/M3U>.

CLAIMS

What is claimed is:

1. A method comprising:

dividing a stream of data into multiple media files each containing a portion of the stream of data on the server device, the multiple media files to be stored as individual files in a memory on the server device in a non-streaming transfer protocol compliant format; and

generating a playlist file having a plurality of tags and a plurality of Universal Resource Indicators (URIs), the plurality of URIs indicating an ordering of the multiple media files to recreate the stream of data.

2. The method of claim 1 wherein the non-streaming transfer protocol comprises a hypertext transfer protocol (HTTP)-compliant protocol and wherein the method further comprises:

transmitting the playlist file to a client device using the non-streaming transfer protocol; and

transferring one or more of the multiple media files to the client device using the non-streaming transfer protocol in response to requests from the client device utilizing one or more of the plurality of URIs.

3. The method of claim 1 further comprising:

generating an updated playlist file corresponding to changes to the multiple media files, the updated playlist file comprising a plurality of updated URIs indicating an ordering of the updated multiple media files to recreate the stream of data.

4. The method of claim 3 wherein the changes to the multiple media files comprises at least one of: (a) additions of one or more media files and the updated playlist file includes URIs to the multiple media files and the added one or more media files; (b) modifications to the one or more media files and the

updated playlist file comprises URIs to the multiple media files, and one or more of the tags in the plurality of tags is updated to reflect the modifications to the one or more media files: or (c) removal of one or more selected media files from the multiple media files and the addition of further media files to result in one or more remaining media files and the updated playlist file comprises URIs to the remaining media files.

5. The method of claim 4 wherein the updated playlist is generated at the expiration of a selected period of time based on an attribute of one of the tags in the playlist file.

6. The method of claim 1 further comprising:

determining an approximate duration of a next media file to be added to the playlist file;

causing the playlist file to include a tag indicating the approximate duration.

7. The method of claim 1 further comprising:

determining an encryption method to be applied to the multiple media files;

causing the playlist file to include a tag to indicate the encryption method to be applied to the multiple media files and wherein the tag comprises a URI to a security key.

8. The method of claim 1 further comprising:

determining a time and/or data associated with a beginning of a next media file; and

causing the playlist to include a tag indicating the time and/or date associated with the beginning of the next media file.

9. The method of claim 1 further comprising:

determining whether a client device to receive the media files is authorized to store the media files after playback;

causing the playlist to include a tag that indicates whether the client device is authorized to store the media files after playback.

10. The method of claim 1 further comprising:

determining whether additional URIs corresponding to additional media files will be added to the playlist file;

causing the playlist to include a tag that indicates whether additional URIs corresponding to additional media files will be added to the playlist file wherein the tag comprises an EXT-X-ENDLIST tag.

11. The method of claim 1 further comprising:

determining whether a next URI in the playlist file indicates another playlist file;

causing the playlist to include a tag to indicate that the next URI in the playlist indicates another playlist wherein the tag indicates a bandwidth associated with the another playlist.

12. An article comprising a computer-readable medium having stored thereon executable instructions that, when executed, cause one or more processors to:

divide a stream of data into multiple media files each containing a portion of the stream of data on the server device, the multiple media files to be stored as individual files in a memory on the server device in a non-streaming transfer protocol compliant format; and

generate a playlist file having a plurality of tags and a plurality of Universal Resource Indicators (URIs), the plurality of URIs indicating an ordering of the multiple media files to recreate the stream of data.

13. The article of claim 12 wherein the non-streaming transfer protocol comprises a hypertext transfer protocol (HTTP)-compliant protocol and wherein further instructions cause the one or more processors to:

transmit the playlist file to a client device using the non-streaming transfer protocol;

transfer one or more of the multiple media files to the client device using the non-streaming transfer protocol in response to requests from the client device utilizing one or more of the plurality of URIs.

14. The article of claim 12 further comprising instructions that, when executed, cause the one or more processors to:

generate an updated playlist corresponding to changes to the multiple media files, the updated playlist comprising a plurality of updated URIs indicating an ordering of the updated multiple media files to recreate the stream of data.

15. The article of claim 14 wherein the changes to the multiple media files comprises at least one of: (a) additions of one or more media files and the updated playlist file includes URIs to the multiple media files and the added one or more media files; (b) modifications to the one or more media files and the updated playlist file comprises URIs to the multiple media files, and one or more of the tags in the plurality of tags is updated to reflect the modifications to the one or more media files; or (c) removal of one or more selected media files from the multiple media files and the addition of further media files to result in one or more remaining media files and the updated playlist file comprises URIs to the remaining media files.

16. The article of claim 14 wherein the updated playlist is generated at the expiration of a selected period of time based on an attribute of one of the tags in the playlist file.

17. The article of claim 12 further comprising instructions that, when executed, cause the one or more processors to:

determine an approximate duration of a next media file to be added to the playlist file;

cause the playlist file to include a tag indicating the approximate duration.

18. The article of claim 12 further comprising instructions that, when executed, cause the one or more processors to:

determine an encryption method to be applied to the multiple media files;

cause the playlist file to include a tag to indicate the encryption method to be applied to the multiple media files, and wherein the tag comprises a URI to a security key.

19. The article of claim 12 further comprising instructions that, when executed, cause the one or more processors to:

determine a time and/or data associated with a beginning of a next media file; and

cause the playlist to include a tag indicating the time and/or date associated with the beginning of the next media file.

20. The article of claim 12 further comprising instructions that, when executed, cause the one or more processors to:

determine whether a client device to receive the media files is authorized to store the media files after playback;

cause the playlist to include a tag that indicates whether the client device is authorized to store the media files after playback.

21. The article of claim 12 further comprising instructions that, when executed, cause the one or more processors to:

determine whether additional URIs corresponding to additional media files will be added to the playlist file;

cause the playlist to include a tag that indicates whether additional URIs corresponding to additional media files will be added to the playlist file wherein the tag comprises an EXT-X-ENDLIST tag.

22. The article of claim 12 further comprising instructions that, when executed, cause the one or more processors to:

determine whether a next URI in the playlist file indicates another playlist file;

cause the playlist to include a tag to indicate that the next URI in the playlist indicates another playlist wherein the tag indicates a bandwidth associated with the another playlist.

23. A method comprising:

requesting, with a client device, a playlist file over a network using a non-streaming transfer protocol;

receiving, in response to the request and with the client device, the playlist file from a server device, the playlist file having Universal Resource Indicators (URIs) indicating a plurality of media files and a plurality of tags having parameters related to playback of the plurality of media files;

requesting one or more of the media files in an order indicated by the playlist file;

receiving the one or more requested media files over the network using the non-streaming transfer protocol.

24. The method of claim 23 further comprising:

generating an audio and/or video output representing the stream of content by playing the media files with the client device in the order indicated by the playlist file, and wherein the non-streaming transfer protocol comprises a hypertext transfer protocol (HTTP)-compliant protocol.

25. The method of claim 23 further comprising:

sending repeatedly a subsequent request for the playlist file to the server device;

receiving an updated playlist file in response to the subsequent request;

using the updated playlist file received in response to the subsequent request.

26. The method of claim 25 further comprising:

comparing the playlist file, received in response to the request, and the updated playlist file;

determining whether the updated playlist file received in response to the subsequent request includes a URI of a last loaded media file;

halting playback if the URI of the last loaded media file is not included in the updated playlist file;

utilizing a tag in the updated playlist file received in response to the subsequent request to access a next media file.

27. The method of claim 26 wherein the tag comprises an EXT-X-MEDIA-SEQUENCE tag.

28. The method of claim 26 further comprising:

sending a second subsequent request for the playlist file to the server device after a first period of time if the playlists differ; and

sending the second subsequent request for the playlist file to the server device after a second period of time if the playlists do not differ, wherein the first period of time is based on a first multiple of attribute of one of the tags in the playlist file and the second period of time is based on a second multiple of the attribute.

29. The method of claim 23 further comprising:

determining, with the client device, whether the playlist file includes a tag indicating whether additional URIs corresponding to media files may be added to the playlist file;

selecting, with the client device, any one of the URIs from the playback file if the tag indicates that additional URIs may not be added to the playlist file or selecting any one of the URIs from the playlist file except for the second-to-last URI and the last URI if the tag indicates that additional URIs may be added to the playlist file;

accessing the media files in an order indicated by the URIs in the playlist file starting with the selected URI and wherein the tag indicates the end of a program presented through the media files.

30. The method of claim 23 further comprising:

determining, with the client device, whether a tag in the playlist file indicates whether the client device is authorized to store media files after playback;

storing the media files on the client device after playback if the tag indicates that the client is authorized to store the media files after playback;

preventing access to the media files after playback if the tag indicates that the client is not authorized to store the media files after playback.

31. An article comprising a computer-readable medium having stored thereon instructions that, when executed, cause one or more processors to:

request, with a client device, a playlist file from a server device over a network using a non-streaming transfer protocol;

receive, in response to the request and with the client device, the playlist file from the server device, the playlist file having Universal Resource Indicators (URIs) indicating a plurality of media files and a plurality of tags having parameters related to playback of the plurality of media files;

request one or more of the media files in an order indicated by the playlist file;

receive the one or more requested media files over the network using the non-streaming transfer protocol.

32. The article of claim 31 further comprising instructions that, when executed, cause the one or more processors to:

generate an audio and/or video output representing the stream of content by playing the media files with the client device in the order indicated by the playlist file, and wherein the non-streaming transfer protocol comprises a hypertext transfer protocol (HTTP)-compliant protocol.

33. The article of claim 31 further comprising instructions that, when executed, cause the one or more processors to:

send repeatedly a subsequent request for the playlist file to the server device;

receive an updated playlist file in response to the subsequent request;

use the updated playlist file received in response to the subsequent request.

34. The article of claim 33 further comprising instructions that, when executed, cause the one or more processors to:

compare the playlist file, received in response to the request, and the updated playlist file;

determine whether the updated playlist file received in response to the subsequent request includes a URI of a last loaded media file;

halt playback if the URI of the last loaded media file is not included in the updated playlist file;

utilize a tag in the updated playlist file received in response to the subsequent request to access a next media file.

35. The article of claim 33 further comprising instructions that, when executed, cause the one or more processors to:

send a second subsequent request for the playlist file to the server device after a first period of time if the playlists differ; and

send the second subsequent request for the playlist file to the server device after a second period of time if the playlists do not differ, wherein the first period of time is based on a first multiple of attribute of one of the tags in the playlist file and the second period of time is based on a second multiple of the attribute.

36. The article of claim 33 further comprising instructions that, when executed, cause the one or more processors to:

determine, with the client device, whether the playlist file includes a tag indicating whether additional URIs corresponding to media files may be added to the playlist file;

select, with the client device, any one of the URIs from the playback file if the tag indicates that additional URIs may not be added to the playlist file or selecting any one of the URIs from the playlist file except for the second-to-last URI and the last URI if the tag indicates that additional URIs may be added to the playlist file;

access the media files in an order indicated by the URIs in the playlist file starting with the selected URI and wherein the tag indicates the end of a program presented through the media files.

37. A server device comprising:

a network interface;

a memory, the memory to store a playlist file and multiple media files;

a processing system coupled with the network interface and the memory, the processing system configured to divide a stream of data to generate the multiple media files where each media file stores a portion of the stream of data on the server device, the processing system configured to store the media files in the memory, the processing system configured to generate the playlist file having a plurality of tags and a plurality of Universal Resource Indicators (URIs) indicating an ordering of the multiple media files to recreate the stream of data.

38. The server device of claim 37 wherein the non-streaming transfer protocol comprises a hypertext transfer protocol (HTTP)-compliant protocol and wherein the processing system is configured to cause to be transmitted the playlist file to a client device using a non-streaming transfer protocol, and to cause to be transferred one or more of the multiple media files to the client device using the non-streaming transfer protocol in response to requests from the client device utilizing one or more of the plurality of URIs.

39. The server device of claim 37 wherein the processing system further generates an updated playlist file corresponding to changes to the multiple media files, the updated playlist file comprising a plurality of updated URIs indicating an ordering of the updated multiple media files to recreate the stream of data and causes to be transmitted the updated playlist file to the client device.

40. A client device comprising:

a network interface;

a memory coupled with the processor, the memory to store a playlist file and media files received via the network interface,

a processor coupled with the network interface, the processor to request the playlist file via the network interface using a non-streaming transfer protocol, to receive the playlist file from a server device, the playlist file having Universal Resource Indicators (URIs) indicating a plurality of media files and a plurality of tags having parameters related to playback of the plurality of media files, to request one or more of the media files in an order indicated by the playlist file, to receive the one or more requested media files over the network using the non-streaming transfer protocol, and to generate an audio and/or video output representing the stream of content by playing the media files in the order indicated by the playlist file.

41. The client device of claim 40 wherein the non-streaming transfer protocol comprises a hypertext transfer protocol (HTTP)-compliant protocol.

42. The client device of claim 40, the processor further to send a subsequent request for the playlist file to the server device, to receive the playlist file from server in response to the subsequent request, to compare the playlist file received in response to the request and the playlist file received in response to the subsequent request, to use the playlist file received in response to the subsequent request if the playlist files differ.

43. A machine readable storage medium storing executable instructions which when executed by a data processing system cause the data processing system to perform a method comprising:

transmitting a playlist file to a client device using a non-streaming transfer protocol, the playlist file having a plurality of tags and a plurality of Universal Resource Indicators (URIs), the plurality of URIs indicating an ordering of multiple media files which have been divided out of a stream of data to recreate the stream of data by sequential presentation of the multiple media files; and

transferring one or more of the multiple media files to the client device using the non-streaming transfer protocol in response to requests from the client device utilizing one or more of the plurality of URIs.

44. The medium of claim 43 wherein the non-streaming transfer protocol comprises a hypertext transfer protocol (HTTP)-compliant protocol and wherein the method further comprises:

transmitting an updated playlist file corresponding to changes to the multiple media files, the updated playlist file comprising a plurality of updated URIs indicating an ordering of the updated multiple media files to recreate the stream of data.

45. A method comprising:

dividing a stream of data into multiple media files, wherein the multiple media files comprise a first group of media files and a second group of media files, wherein the first group of media files are configured to be stored with a first encoding in a transfer protocol compliant format and wherein the second group of media files are configured to be stored with a second encoding in the transfer protocol compliant format;

generating a first playlist file having a plurality of tags and a plurality of Universal Resource Indicators (URIs) indicating an ordering of the first group of media files to recreate the stream of data;

generating a second playlist file having a plurality of tags and a plurality of URIs indicating an ordering of the second group of media files to recreate the stream of data; and

generating a variant playlist file having a plurality of tags and a plurality of URIs indicating the first playlist file and the second playlist file.

46. The method of claim 45 wherein the transfer protocol comprises a hypertext transfer protocol (HTTP)-compliant protocol and wherein the first encoding and the second encoding are different in that they are one of: encodings with the same codec but at different data rates or encodings with different codecs at the same or different data rates.

47. The method of claim 45 further comprising:

generating an updated variant playlist file corresponding to an updated first playlist file or an updated second playlist file, wherein the updated first playlist file or the updated second playlist file includes at least one updated URI indicating at least one change to the first or second group of media files.

48. The method of claim 47 wherein the at least one change to the first or second group of media files comprises adding one or more media files and the updated first or second playlist file includes URIs to the media files and the added one or more media files.

49. The method of claim 47 wherein the at least one change to the first or second group of media files comprises modifications to the one or more media files and the updated first or second playlist file comprises URIs to the media files, and one or more of the tags in the plurality of tags is updated to reflect the modifications to the one or more media files.

50. The method of claim 47 wherein the at least one change to the first or second group of media files comprises removal of one or more selected media

files from the media files and the addition of one or more further media files to result in one or more remaining media files and the updated first or second playlist file comprises URIs to the remaining media files and wherein the stream of data is a contiguous time based stream of content for a single program.

51. An article comprising a computer-readable medium having stored thereon executable instructions that, when executed, cause one or more processors to:

divide a stream of data into multiple media files, wherein the multiple media files comprise a first group of media files and a second group of media files, wherein the first group of media files are configured to be stored with a first encoding in a transfer protocol compliant format and wherein the second group of media files are configured to be stored with a second encoding in the transfer protocol compliant format;

generate a first playlist file having a plurality of tags and a plurality of Universal Resource Indicators (URIs) indicating an ordering of the first group of media files to recreate the stream of data;

generate a second playlist file having a plurality of tags and a plurality of URIs indicating an ordering of the second group of media files to recreate the stream of data;

generate a variant playlist file having a plurality of tags and a plurality of URIs indicating the first playlist file and the second playlist file.

52. The article of claim 51 wherein the transfer protocol comprises a hypertext transfer protocol (HTTP)-compliant protocol and wherein the first encoding and the second encoding are different in that they are one of: encodings with the same codec but at different data rates or encodings with different codecs at the same or different data rates.

53. The article of claim 51 further comprising instructions that, when executed, cause the one or more processors to:

generate an updated variant playlist file corresponding to an updated first playlist file or an updated second playlist file, wherein the updated first playlist file or the updated second playlist file includes at least one updated URI indicating at least one change to the first or second group of media files.

54. The article of claim 53 wherein the at least one change to the first or second group of media files comprises adding one or more media files and the updated first or second playlist file includes URIs to the media files and the added one or more media files.

55. The article of claim 53 wherein the at least one change to the first or second group of media files comprises modifications to the one or more media files and the updated first or second playlist file comprises URIs to the media files, and one or more of the tags in the plurality of tags is updated to reflect the modifications to the one or more media files.

56. The article of claim 53 wherein the at least one change to the first or second group of media files comprises removal of one or more selected media files from the media files and the addition of one or more further media files to result in one or more remaining media files and the updated first or second playlist file comprises URIs to the remaining media files and wherein the stream of data is a contiguous time based stream of content for a single program.

57. An apparatus comprising:

means for dividing a stream of data into multiple media files, wherein the multiple media files comprise a first group of media files and a second group of media files, wherein the first group of media files are configured to be stored with a first encoding in a transfer protocol compliant format and wherein the second group of media files are configured to be stored with a second encoding in the transfer protocol compliant format;

means for generating a first playlist file having a plurality of tags and a plurality of Universal Resource Indicators (URIs) indicating an ordering of the first group of media files to recreate the stream of data;

means for generating a second playlist file having a plurality of tags and a plurality of URIs indicating an ordering of the second group of media files to recreate the stream of data; and

means for generating a variant playlist file having a plurality of tags and a plurality of URIs indicating the first playlist file and the second playlist file.

58. A machine implemented method comprising:

requesting, with a client device, a playlist file over a network using a transfer protocol;

receiving, in response to the requesting, a variant playlist comprising a plurality of tags and a plurality of Universal Resource Indicators (URIs) indicating a first playlist file and a second playlist file, wherein one or more of the plurality of tags indicates a first bandwidth parameter associated with a first playlist file and a second bandwidth parameter associated with a second playlist file wherein the first bandwidth parameter is less than the second bandwidth parameter;

selecting, with the client device, the first playlist file or the second playlist file;

using the transfer protocol, requesting the selected first or second playlist file with the client device, wherein the requesting uses a URI from the variant playlist file corresponding to the selected playlist file;

receiving, with the client device, the requested playlist file, the requested playlist file having URIs indicating a plurality of media files and a plurality of tags having parameters related to playback of the plurality of media files;

using the transfer protocol, requesting one or more of the media files in an order indicated by the requested playlist file;

receiving the one or more requested media files over the network using the transfer protocol.

59. The method of claim 58 further comprising:

generating an audio and/or video output representing the stream of content by playing the media files with the client device in the order indicated by the playlist file, and wherein selecting, with the client device, the first playlist file or the second playlist file comprises selecting the first playlist file having a lower associated bandwidth parameter.

60. The method of claim 58 further comprising:

monitoring available bandwidth on the client device while receiving the one or more requested media files over the network wherein the one or more requested media files is received over the transfer protocol;

comparing the available bandwidth and the first bandwidth parameter associated with the selected playlist file;

wherein the requesting the first or the second playlist file comprises requesting the second playlist file if the second bandwidth parameter is within the available bandwidth and the available bandwidth exceeds the first bandwidth parameter;

wherein the requesting of the one or more media files comprises requesting media files utilizing URIs in the second bandwidth playlist file.

61. An article comprising a computer-readable medium having stored thereon executable instructions that, when executed, cause one or more processors to perform a method comprising:

requesting, with a client device, a playlist file over a network using a transfer protocol;

receiving, in response to the requesting, a variant playlist comprising a plurality of tags and a plurality of Universal Resource Indicators (URIs) indicating a first playlist file and a second playlist file, wherein one or more of the plurality of tags indicates a first bandwidth parameter associated with a first playlist file and a second bandwidth parameter associated with a second playlist

file wherein the first bandwidth parameter is less than the second bandwidth parameter;

selecting, with the client device, the first playlist file or the second playlist file;

using the transfer protocol, requesting the selected first or second playlist file with the client device, wherein the requesting uses a URI from the variant playlist file corresponding to the selected playlist file;

receiving, with the client device, the requested playlist file, the requested playlist file having URIs indicating a plurality of media files and a plurality of tags having parameters related to playback of the plurality of media files;

using the transfer protocol, requesting one or more of the media files in an order indicated by the requested playlist file;

receiving the one or more requested media files over the network using the transfer protocol.

62. The article of claim 61 further comprising generating an audio and/or video output representing the stream of content by playing the media files with the client device in the order indicated by the playlist file, and wherein selecting, with the client device, the first playlist file or the second playlist file comprises selecting the first playlist file having a lower associated bandwidth parameter.

63. The article of claim 61 further comprising instructions that, when executed, cause the one or more processors to perform a further method comprising:

monitoring available bandwidth on the client device while receiving the one or more requested media files over the network wherein the one or more requested media files is received over the transfer protocol;

comparing the available bandwidth and the first bandwidth parameter associated with the selected playlist file;

wherein the requesting the first or the second playlist file comprises requesting the second playlist if the second bandwidth parameter is within the

available bandwidth and the available bandwidth exceeds the first bandwidth parameter;

wherein the requesting of the one or more media files comprises requesting media files utilizing URIs in the second bandwidth playlist file.

64. An apparatus comprising:

means for requesting, with a client device, a playlist file over a network using a transfer protocol;

means for receiving, in response to the request, a variant playlist comprising a plurality of tags and a plurality of Universal Resource Indicators (URIs) indicating a first playlist file and a second playlist file, wherein one or more of the plurality of tags indicates a first bandwidth parameter associated with a first playlist file and a second bandwidth parameter associated with a second playlist file wherein the first bandwidth parameter is less than the second bandwidth parameter;

means for selecting, with the client device, the first playlist file or the second playlist file;

means for requesting, using the transfer protocol, the selected first or second playlist file with the client device, wherein the requesting uses a URI from the variant playlist file corresponding to the selected playlist file;

means for receiving, with the client device, the requested playlist file, the requested playlist file having URIs indicating a plurality of media files and a plurality of tags having parameters related to playback of the plurality of media files;

means for requesting, using the transfer protocol, one or more of the media files in an order indicated by the requested playlist file;

means for receiving the one or more requested media files over the network using the transfer protocol;

means for generating an audio and/or video output representing the stream of content by playing the media files with the client device in the order indicated by the requested playlist file.

65. The apparatus of claim 64 further comprising:

means for monitoring available bandwidth on the client device while receiving the one or more requested media files over the network wherein the one or more requested media files is received through the non-streaming transfer protocol;

means for comparing the available bandwidth and the first bandwidth parameter associated with the selected playlist file;

means for requesting the second playlist if the second bandwidth parameter is within the available bandwidth and the available bandwidth exceeds the first bandwidth parameter;

means for requesting media files utilizing URIs in the second bandwidth playlist file.

66. A method comprising:

transmitting a variant playlist file to a client device using a transfer protocol, the variant playlist file having a plurality of tags and a plurality of URIs indicating a first playlist file and a second playlist file and wherein the first playlist file has a plurality of tags and a plurality of Universal Resource Indicators (URIs) indicating an ordering of a first group of media files, having a first encoding, to recreate a stream of data for a contiguous time based stream of content of a program, and wherein the second playlist file has a plurality of tags and a plurality of URIs indicating an ordering of a second group of media files, having a second encoding, to recreate the stream of data;

transmitting, using the transfer protocol, either the first playlist file or the second playlist file to the client device in response to a request from the client device utilizing a URI from the variant playlist file;

transferring, using the transfer protocol, at least one of: (a) media files from the first group of media files to the client device in response to one or more requests from the client device utilizing URIs from the first playlist file, or (b) media files from the second group of media files to the client device in response

to one or more requests from the client device utilizing URIs from the second playlist file.

67. The method of claim 66 wherein the transfer protocol comprises a hypertext transfer protocol (HTTP)-compliant protocol.

68. The method of claim 66 further comprising:

transmitting an updated variant playlist file to the client device, the updated variant playlist file including updates to the first playlist file and the second playlist file, and wherein the updated first playlist file includes at least one updated URI indicating at least one change to the first group or second group of media files.

69. The method of claim 68 wherein the at least one change to the first or second group of media files comprises adding one or more media files and the updated first or second playlist file includes URIs to the media files and the added one or more media files.

70. The method of claim 68 wherein the at least one change to the first or second group of media files comprises modifications to the one or more media files and the updated first or second playlist file comprises URIs to the media files, and one or more of the tags in the plurality of tags is updated to reflect the modifications to the one or more media files.

71. The method of claim 68 wherein the at least one change to the first or second group of media files comprises removal of one or more selected media files from the media files and the addition of one or more further media files to result in one or more remaining media files and the updated first playlist file comprises URIs to the remaining media files.

72. An article comprising a computer readable medium having stored thereon executable instructions that, when executed, cause one or more processors to:

transmit a variant playlist file to a client device using a transfer protocol, the variant playlist file having a plurality of tags and a plurality of URIs indicating a first playlist file and a second playlist file, and wherein the first playlist file has a plurality of tags and a plurality of Universal Resource Indicators (URIs) indicating an ordering of the first group of media files, having a first encoding, to recreate a stream of data for a contiguous time based stream of content of a program, and wherein the second playlist file has a plurality of tags and a plurality of URIs indicating an ordering of a second group of media files, having a second encoding, to recreate the stream of data;

transmit, using the transfer protocol, either the first playlist file or the second playlist file to the client device in response to a request from the client device utilizing a URI from the variant playlist file;

transfer, using the transfer protocol, at least one of (a) media files from the first group of media files to the client device in response to one or more request from the client device utilizing URIs from the first playlist file; or (b) media files from the second group of media files to the client device in response to one or more requests from the client device utilizing URIs from the second playlist file.

73. The article of claim 72 wherein the transfer protocol comprises a hypertext transfer protocol (HTTP)-compliant protocol.

74. The article of claim 72 further comprising instructions that, when executed, cause the one or more processors to:

transmit an updated variant playlist file to the client device, the updated variant playlist file including updates to the first playlist file and updates to the second playlist file and wherein the updates to the first playlist file include a plurality of updated URIs indicating changes to the first group of media files stored with the first encoding in the transfer protocol compliant format and the

updates to the second playlist file include changes to the second group of media files stored with the first encoding in the transfer protocol compliant format.

75. The article of claim 74 wherein the changes to the media files comprises adding one or more media files and the updates to the first playlist file include URIs to the media files and the added one or more media files.

76. The article of claim 74 wherein the changes to the media files comprises modifications to the one or more media files and the updates to the first playlist file comprise URIs to the media files, and one or more of the tags in the plurality of tags is updated to reflect the modifications to the one or more media files.

77. The article of claim 74 wherein the changes to the media files comprises removal of one or more selected media files from the media files and the addition of one or more further media files to result in one or more remaining media files and the updates to the first playlist file comprise URIs to the remaining media files.

78. An apparatus comprising:

means for transmitting a variant playlist file to a client device using a transfer protocol, the variant playlist file having a plurality of tags and a plurality of URIs indicating a first playlist file and a second playlist file and wherein the first playlist file has a plurality of tags and a plurality of Universal Resource Indicators (URIs) indicating an ordering of a first group of media files, having a first encoding, to recreate a stream of data for a contiguous time based stream of content of a program and wherein the second playlist file has a plurality of tags and a plurality of URIs indicating an ordering of a second group of media files, having a second encoding, to recreate the stream of data;

means for transmitting either the first playlist file or the second playlist file to the client device in response to a request from the client device utilizing the URIs from the variant playlist file using the transfer protocol;

means for transferring at least one of (a) media files from the first group of media files to the client device in response to one or more requests from the client device utilizing URIs from the first playlist file using the transfer protocol; or (b) media files from the second group of media files to the client device in response to one or more requests from the client device utilizing URIs from the second playlist file using the transfer protocol.

79. A machine implemented method comprising:

dividing, at a server device, a stream of data into multiple media files, each of the multiple media files to be stored as individual files in a memory in a transfer protocol compliant format;

generating a playlist file having a plurality of tags and a plurality of Universal Resource Indicators (URIs), the plurality of URIs indicating an ordering of the multiple media files to recreate the stream of data; and

generating an updated playlist file corresponding to changes to the multiple media files, the updated playlist file comprising a plurality of updated URIs indicating an ordering of updated multiple media files to recreate a representation of the stream of data.

80. The method of claim 79 wherein the changes to the multiple media files comprises adding one or more media files and the updated playlist file includes URIs to the multiple media files and the added one or more media files.

81. The method of claim 79 wherein the changes to the multiple media files comprises modifications to the one or more media files and the updated playlist file comprises URIs to the multiple media files, and one or more of the tags in the plurality of tags is updated to reflect the modifications to the one or more media files.

82. The method of claim 79 wherein the changes to the multiple media files comprises removal of one or more selected media files from the multiple media

files and adding one or more further media files to result in one or more remaining media files and the updated playlist file comprises URIs to the remaining media files and wherein the stream of data is a contiguous time based stream of content for a single program and wherein the transfer protocol is an HTTP compliant protocol.

83. The method of claim 79 wherein the updated playlist is generated at the expiration of a selected period of time.

84. The method of claim 83 wherein the selected period of time is based on an attribute of one of the tags in the playlist file.

85. The method of claim 79 further comprising:

determining an approximate duration of a next media file to be added to the playlist file;

causing the playlist file to include a tag indicating the approximate duration.

86. The method of claim 85 wherein the tag comprises an EXT-X-TARGETDURATION tag.

87. The method of claim 79 further comprising:

determining sequence numbers for one or more of the URIs in the playlist file; and

causing the playlist file to include one or more tags indicating one or more of the sequence numbers.

88. The method of claim 87 wherein the one or more tags comprise one or more EXT-X-MEDIA-SEQUENCE tags.

89. The method of claim 79 further comprising:

determining whether a client device to receive the media files is authorized to store the media files after playback;

causing the playlist to include a tag that indicates whether the client device is authorized to store the media files after playback.

90. The method of claim 89 wherein the tag comprises an EXT-X-ALLOW-CACHE tag.

91. An article comprising a computer-readable medium having stored thereon executable instructions that, when executed, cause one or more processors to:

divide, at a server device, a stream of data into multiple media files, each of the multiple media files to be stored as individual files in a memory in a transfer protocol compliant format;

generate a playlist file having a plurality of tags and a plurality of Universal Resource Indicators (URIs), the plurality of URIs indicating an ordering of the multiple media files to recreate the stream of data; and

generate an updated playlist file corresponding to changes to the multiple media files, the updated playlist file comprising a plurality of updated URIs indicating an ordering of updated multiple media files to recreate a representation of the stream of data.

92. The article of claim 91 wherein the changes to the multiple media files comprises adding one or more media files and the updated playlist file includes URIs to the multiple media files and the added one or more media files.

93. The article of claim 91 wherein the changes to the multiple media files comprises modifications to the one or more media files and the updated playlist file comprises URIs to the multiple media files, and one or more of the tags in the plurality of tags is updated to reflect the modifications to the one or more media files.

94. The article of claim 91 wherein the changes to the multiple media files comprises removal of one or more selected media files from the multiple media files and adding one or more further media files to result in one or more remaining media files and the updated playlist file comprises URIs to the remaining media files and wherein the stream of data is a contiguous time based stream of content for a single program and wherein the transfer protocol is an HTTP compliant protocol.

95. The article of claim 91 wherein the updated playlist is generated at the expiration of a selected period of time.

96. The article of claim 95 wherein the selected period of time is based on an attribute of one of the tags in the playlist file.

97. The article of claim 91 further comprising:

determining an approximate duration of a next media file to be added to the playlist file;

causing the playlist file to include a tag indicating the approximate duration.

98. The article of claim 97 wherein the tag comprises an EXT-X-TARGETDURATION tag.

99. The article of claim 91 further comprising:

determining sequence numbers for one or more of the URIs in the playlist file; and

causing the playlist file to include one or more tags indicating one or more of the sequence numbers.

100. The article of claim 99 wherein the one or more tags comprise one or more EXT-X-MEDIA-SEQUENCE tags.

101. The article of claim 91 further comprising:

determining whether a client device to receive the media files is authorized to store the media files after playback;

causing the playlist to include a tag that indicates whether the client device is authorized to store the media files after playback.

102. An apparatus comprising:

means, at a server device, for dividing a stream of data into multiple media files, each of the multiple media files to be stored as individual files in a memory in a transfer protocol compliant format;

means for generating a playlist file having a plurality of tags and a plurality of Universal Resource Indicators (URIs), the plurality of URIs indicating an ordering of the multiple media files to recreate the stream of data; and

means for generating an updated playlist file corresponding to changes to the multiple media files, the updated playlist file comprising a plurality of updated URIs indicating an ordering of updated multiple media files to recreate a representation of the stream of data.

103. The apparatus of claim 102 wherein the changes to the multiple media files comprises adding one or more media files and the updated playlist file includes URIs to the multiple media files and the added one or more media files.

104. The apparatus of claim 102 wherein the changes to the multiple media files comprises modifications to the one or more media files and the updated playlist file comprises URIs to the multiple media files, and one or more of the tags in the plurality of tags is updated to reflect the modifications to the one or more media files.

105. The apparatus of claim 102 wherein the changes to the multiple media files comprises removal of one or more selected media files from the multiple media files and adding one or more further media files to result in one or more

remaining media files and the updated playlist file comprises URIs to the remaining media files and wherein the stream of data is a contiguous time based stream of content for a single program and wherein the transfer protocol is an HTTP compliant protocol and wherein the updated playlist is generated at the expiration of a selected period of time and wherein the selected period of time is based on an attribute of one of the tags in the playlist file.

106. A machine implemented method comprising:
storing multiple media files as individual files in a memory in a transfer protocol compliant format, the multiple media files having been divided out of a contiguous time based stream of data;

transmitting a playlist file to a client device using the transfer protocol, the playlist file having a plurality of tags and a plurality of Universal Resource Indicators (URIs), the plurality of URIs indicating an ordering of the multiple media files to recreate the stream of data;

transferring one or more of the multiple media files to the client device using the transfer protocol in response to one or more requests from the client device utilizing one or more of the plurality of URIs;

transmitting an updated playlist file to the client device, the updated playlist file corresponding to changes to the multiple media files, the updated playlist file comprising a plurality of updated URIs indicating an ordering of updated multiple media files to recreate a representation of the stream of data.

107. The method of claim 106 wherein the changes to the multiple media files comprises at least one of: (a) the addition of one or more media files and the updated playlist file includes URIs to the multiple media files and the added one or more media files; (b) modifications to the one or more media files and the updated playlist file comprises URIs to the multiple media files, and one or more of the tags in the plurality of tags is updated to reflect the modifications to the one or more media files; or (c) removal of one or more selected media files from the multiple media files and the addition of one or more further media files to

result in one or more remaining media files and the updated playlist file comprises URIs to the remaining media files and wherein the stream of data is a contiguous time based stream of content for a single program and wherein the transfer protocol is an HTTP compliant protocol.

108. The method of claim 106 wherein the updated playlist is generated at the expiration of a selected period of time and wherein the selected period of time is based on an attribute of one of the tags in the playlist file and wherein the transfer protocol is an HTTP compliant protocol.

109. An article comprising a computer readable medium having stored thereon executable instructions that, when executed, case one or more processors to:

store multiple media files as individual files in a memory in a transfer protocol compliant format, the multiple media files having been divided out of a contiguous time based stream of data;

transmit a playlist file to a client device using the non-streaming transfer protocol, the playlist file having a plurality of tags and a plurality of Universal Resource Indicators (URIs), the plurality of URIs indicating an ordering of the multiple media files to recreate the stream of data;

transfer one or more of the multiple media files to the client device using the transfer protocol in response to one or more requests from the client device utilizing one or more of the plurality of URIs;

transmit an updated playlist file to the client device, the updated playlist file corresponding to changes to the multiple media files, the updated playlist file comprising a plurality of updated URIs indicating an ordering of the updated multiple media files to recreate a representation of the stream of data.

110. The article of claim 109 wherein the changes to the multiple media files comprises at least one of: (a) the addition of one or more media files and the updated playlist file includes URIs to the multiple media files and the added one or more media files; (b) modifications to the one or more media files and the

updated playlist file comprises URIs to the multiple media files, and one or more of the tags in the plurality of tags is updated to reflect the modifications to the one or more media files; or (c) removal of one or more selected media files from the multiple media files and the addition of one or more further media files to result in one or more remaining media files and the updated playlist file comprises URIs to the remaining media files and wherein the stream of data is a contiguous time based stream of content for a single program and wherein the transfer protocol is an HTTP compliant protocol.

111. The article of claim 109 wherein the updated playlist is generated at the expiration of a selected period of time and wherein the selected period of time is based on an attribute of one of the tags in the playlist file and wherein the transfer protocol is an HTTP compliant protocol.

112. An apparatus comprising:

means for storing multiple media files as individual files in a memory in a transfer protocol compliant format, the multiple media files having been divided out of a contiguous time based stream of data;

means for transmitting a playlist file to a client device using the transfer protocol, the playlist file having a plurality of tags and a plurality of Universal Resource Indicators (URIs), the plurality of URIs indicating an ordering of the multiple media files to recreate the stream of data;

means for transferring one or more of the multiple media files to the client device using the transfer protocol in response to one or more requests from the client device utilizing one or more of the plurality of URIs;

means for transmitting an updated playlist file to the client device, the updated playlist file corresponding to changes to the multiple media files, the updated playlist file comprising a plurality of updated URIs indicating an ordering of the updated multiple media files to recreate a representation of the stream of data.

113. A machine implemented method comprising:

requesting, with a client device, a playlist file using a transfer protocol;

receiving, in response to the requesting, the playlist file having (1) Universal Resource Indicators (URIs) indicating a plurality of media files for a contiguous time based stream of data of a program and (2) a plurality of tags having parameters related to the plurality of media files;

requesting and receiving one or more of the media files in an order indicated in the playlist;

receiving an updated playlist file corresponding to changes to the media files, the updated playlist file comprising a plurality of updated URIs indicating an ordering of an updated plurality of media files.

114. The method as in claim 113 further comprising:

generating a presentation of the program from the updated plurality of media files.

115. The method as in claim 114 wherein the changes to the media files comprises at least one of: (a) the addition of one or more media files and the updated playlist file includes URIs to the updated plurality of media files; (b) modifications of one or more of the plurality of media files; or (c) removal of a first set of media files from the playlist file and the addition of one or more further media files to the playlist file and wherein the transfer protocol is an HTTP compliant protocol.

116. The method as in claim 115 wherein the updated playlist is requested by the client device at the expiration of a time based on an attribute of one of the tags in the playlist file.

117. A machine readable storage medium having stored thereon executable instructions that, when executed, cause one or more processors to perform a method comprising:

requesting, with a client device, a playlist file using a transfer protocol;

receiving, in response to the requesting, the playlist file having Universal Resource Indicators (URIs) indicating a plurality of media files for a contiguous time based stream of data of a program and a plurality of tags having parameters related to the plurality of media files;

requesting and receiving one or more of the media files in an order indicated in the playlist;

receiving an updated playlist file corresponding to changes to the media files, the updated playlist file comprising a plurality of updated URIs indicating an ordering of an updated plurality of media files.

118. The storage medium as in claim 117 wherein the method further comprises:

generating a presentation of the program from the updated playlist.

119. The storage medium as in claim 118 wherein the changes to the media files comprises at least one of: (a) the addition of one or more media files and the updated playlist file includes URIs to the updated plurality of media files; (b) modifications of one or more of the plurality of media files; or (c) removal of a first set of media files from the playlist file and the addition of one or more further media files to the playlist file and wherein the transfer protocol is an HTTP compliant protocol.

120. The storage medium as in claim 119 wherein the updated playlist is requested by the client device at the expiration of a time based on an attribute of one of the tags in the playlist file.

121. A data processing system comprising:

means for requesting, with a client device, a playlist file using a transfer protocol;

means for receiving, in response to the requesting, the playlist file having Universal Resource Indicators (URIs) indicating a plurality of media files for a contiguous time based stream of data of a program and a plurality of tags having parameters related to the plurality of media files;

means for requesting and receiving one or more of the media files in an order indicated in the playlist;

means for receiving an updated playlist file corresponding to changes to the media files, the updated playlist file comprising a plurality of updated URIs indicating an ordering of an updated plurality of media files.

122. A computer readable storage medium for use with an electronic device, the medium to store data in a digital, random access format that is configured to provide a playlist file having a plurality of tags and a plurality of URIs, each of the URIs indicating a respective media file, wherein the media files provide segments of a single contiguous presentation such that when played in order the media files provide playback of the single contiguous presentation, the order of the URIs in the playlist file indicating an order of retrieval by the electronic device of the media files through a network using a non-streaming hypertext transfer protocol and the order of the URIs in the playlist indicating an order of playback for the media files, the tags in the playlist providing information related to playback of the media files.

123. The storage medium of claim 122 wherein the playlist file comprises a tag that indicates an approximate duration of a media file that follows the tag in the playlist file.

124. The storage medium of claim 123 wherein the tag comprises an EXT-X-TARGETDURATION tag.

125. The storage medium of claim 122 wherein the playlist file comprises a tag that indicates a unique sequence number for a first media file indicated by the playlist file.

126. The storage medium of claim 125 wherein the tag comprises an EXT-X-MEDIA-SEQUENCE tag.

127. The storage medium of claim 122 wherein the playlist file comprises a tag that indicates a location of an encryption key used to encrypt at least one of the media files.

128. The storage medium of claim 127 wherein the tag comprises an EXT-X-KEY tag.

129. The storage medium of claim 122 wherein the playlist file comprises a tag that associates a beginning of a media file, that follows the tag, with a date and time.

130. The storage medium of claim 129 wherein the tag comprises an EXT-X-PROGRAM-DATE-TIME tag.

131. The storage medium of claim 122 wherein the playlist file comprises a tag to indicate whether a client device may store a media file after playback.

132. The storage medium of claim 131 wherein the tag comprises an EXT-X-ALLOW-CACHE tag.

133. The storage medium of claim 122 wherein the playlist file comprises a tag that indicates a location of a subsequent playlist file that follows the tag.

134. The storage medium of claim 133 wherein the tag comprises an EXT-X-STREAM-INF tag.

135. A computer readable storage medium for use with an electronic device, the medium to store data in a digital, random access format that is configured to provide a variant playlist file having a plurality of tags and a plurality of URIs, each of the URIs indicating a respective media playlist file, wherein each of the media playlist files specify segments of a single contiguous presentation such that when played in order the segments provide playback of the presentation, the order of the URIs in each of the media playlist files indicating an order of playback for the segments, the tags in the variant playlist file providing information related to a connection bandwidth for each of the URIs indicating a respective media playlist file.

136. The storage medium of claim 135 wherein the variant playlist file comprises a tag proceeding each URI that indicates a respective media playlist file wherein each tag indicates a program identifier, wherein the program identifier is the same for each of the URIs indicating a respective media playlist file, and wherein the program identifier identifies the single contiguous presentation.

137. The storage medium of claim 136 wherein at least one of the tags comprises an identifier of a CODEC.

138. The storage medium of claim 137 wherein corresponding segments from each media playlist file contain the same audio encoding.

139. The storage medium of claim 137 wherein a tag in the variant playlist file comprises an EXT-X-STREAM-INF tag.

140. A machine readable storage medium containing executable instructions which, when executed by a first processing system, cause the first processing system to perform a method comprising:

determining from a tag in a text file that the text file specifies a plurality of URIs (Universal Resource Indicators) of a plurality of segments of a time based stream to be received at the first processing system through a network, wherein the plurality of segments form a single contiguous media presentation;

determining from the text file a duration of at least one of the segments;

retrieving from the text file each of the plurality of URIs to cause the first processing system to retrieve each of the plurality of segments, wherein the plurality of segments are retrieved in the order of the plurality of URIs in the text file, wherein the retrieving allows the first processing system to present the segments in the order of the plurality of URIs in the text file.

141. The medium as in claim 140, the method further comprising:

determining from a tag in the text file a location of a decryption key to decrypt at least one of the segments; and

determining when to reload the text file.

142. The medium as in claim 141, the method further comprising: determining from a tag in the text file that the end of the time based stream has been reached.

143. A computer readable storage medium for use with an electronic device, the medium storing a file comprising:

a first identifier in the file, the first identifier indicating that the file specifies a plurality of URIs (Universal Resource Indicators) of a plurality of segments of a time based stream to be received by the electronic device through a network, wherein the plurality of segments form a single contiguous presentation; and

at least one second identifier that indicates a duration of at least one of the segments;

wherein the plurality of URIs provide addresses for each of the segments, wherein the plurality of segments are retrieved in the order of the plurality of URIs in the file and wherein the segments provide playback of the single contiguous presentation when the segments are played back in the order of the plurality of URIs in the file.

144. The medium as in claim 143 further comprising:

at least one third identifier which specifies a URI of a decryption key to decrypt at least one of the segments.

145. The medium as in claim 144 further comprising:

a tag indicating that the single contiguous presentation has ended.

146. A machine implemented method comprising:

requesting, with a client device, a playlist file over a network using a transfer protocol;

receiving, in response to the requesting, a variant playlist comprising a plurality of tags and a plurality of Universal Resource Indicators (URIs) indicating a first playlist file and a second playlist file, wherein one or more of the plurality of tags indicates a first URI associated with a first playlist file and specifying a first location for content and a second URI associated with a second playlist file and specifying a second location for the content, wherein the second location is redundant of the first location;

selecting, with the client device, the first playlist file or the second playlist file;

using the transfer protocol, requesting the selected first or second playlist file with the client device, wherein the requesting uses a URI from the variant playlist file corresponding to the selected playlist file;

receiving, with the client device, the requested playlist file, the requested playlist file having URIs indicating a plurality of media files and a plurality of tags having parameters related to playback of the plurality of media files;

using the transfer protocol, requesting one or more of the media files in an order indicated by the requested playlist file;

receiving the one or more requested media files over the network using the transfer protocol.

147. The method of claim 146 further comprising:
generating an audio and/or video output representing the stream of content by playing the media files with the client device in the order indicated by the playlist file, and wherein selecting, with the client device, the first playlist file or the second playlist file comprises selecting the first playlist file having a lower associated bandwidth parameter.

148. A machine readable storage medium storing executable program instructions that when executed by a data processing system cause the data processing system to perform a method comprising:

requesting, with a client device, a playlist file over a network using a transfer protocol, wherein the requesting specifies at least one compression protocol;

receiving, with the client device, the playlist file, the playlist file having URIs indicating a plurality of media files and a plurality of tags having parameters related to playback of the plurality of media files and decoding, with the client device, the playlist file;

using the transfer protocol, requesting one or more of the media files in an order indicated by the requested playlist file;

receiving the one or more requested media files over the network using the transfer protocol.

149. The medium of claim 148, wherein the method further comprises:

generating an audio and/or video output representing the stream of content by playing the media files with the client device in the order indicated by the playlist file.

150. The medium of claim 149 wherein the compression protocol is compatible with a standards based HTTP implementation of the transfer protocol.

151. The medium of claim 150 wherein the compression protocol is one of deflate or gzip.

152. A machine readable storage medium storing executable program instructions that when executed by a data processing system cause the data processing system to perform a method comprising:

receiving, through a network using a transfer protocol, a request for a playlist file, the request specifying at least one compression protocol;

transmitting, to a client device, the playlist file in a compressed format using the at least one compression protocol.

153. The medium of claim 152 wherein the at least one compression protocol is compatible with a standards based HTTP implementation of the transfer protocol.